Bulletin of the Technical Committee on

Data Engineering

June 2021 Vol. 45 No. 1

IEEE Computer Society

Letters

Special Issue on Knowledge Management in E-Commerce Applications

Conference and Journal Notices

The TC on Data Engineering

Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems. The TCDE web page is http://tab.computer.org/tcde/index.html.

The Data Engineering Bulletin

The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

The Data Engineering Bulletin web site is at http://tab.computer.org/tcde/bull_about.html.

Letter from the Editor-in-Chief

The March issue of the Data Engineering Bulletin focuses on the intricate interplay as well as a significant gap between data management and machine learning when it comes to supporting real-life business applications.

The opinion piece of this issue features a group of distinguished researchers and their assessment and prognosis of machine learning's current and future roles in building database systems. Besides highlighting several specific potentials and challenges such as using machine learning to optimize database indices and query optimization, the article also gives a great overview of how databases, data analytics and machine learning, system and infrastructure, work together to support today's business needs. It is clear that the business needs, the volume, velocity, and variety of the data, the latency and throughput requirements have evolved dramatically and in consequence, data management systems must adapt. The opinion pieces described four disruptive forces underneath the evolution, which are likely to influence future data systems.

Our associate editor Sebastian Schelter put together the current issue–Data Validation for Machine Learning Models and Applications–that consists of six papers from leading researchers in industry and academia. The papers focus on data validation, which is a critical component in end-to-end machine learning pipelines that many business applications rely on.

Haixun Wang
Instacart

Letter from the Special Issue Editor

The global e-Commerce market size is valued at USD 9.09 trillion with an annual growth rate of 14.7

This special issue presents some recent work from both industry companies and academy on the issues in E-Commerce. The first two papers how structured data and knowledge about product can help improve product understanding and ranking. The next two papers talk about how to model user preference to build tailored and personalized experience in E-Commerce. Deep Learning is the new trend. The last two papers highlights how classical problems are solved with new deep learning models now.

Working on this issue has been a privilege for me, and I would like to thank the authors for their contributions.

Bing Yin
Amazon.com

Using Product Meta Information For Bias Removal In E-Commerce Grid Search

Apoorva Balyan
Walmart Global Tech India
Apoorva.Balyan@walmart.com

Atul Singh
Walmart Global Tech India
Atul.Singh@walmart.com

Praveen Reddy Suram
Walmart Global Tech India
Praveen.Suram@walmart.com

Deepak Arora
Walmart Global Tech India
Deepak.arora1@walmart.com

Varun Srivastava
Walmart Global Tech India
varun.srivastava@walmart.com

Abstract

In e-commerce, product search plays a crucial role in helping customers discover and purchase products. Most IR algorithms focus on creating a relationship between product and customer intent. These techniques stand on two pillars of information primarily- first, what information the seller provides about a product i.e description, title, taxonomy etc, and second, the implicit feedback data that is collected from the search logs which comprises of millions of user activity events. This data is inherently biased in nature as the user activity is not only dependent on the quality of query-item match but also on how probable the user is to observe that item in the first place. In the IR theory and past research efforts, discounting based on position in evaluation methods have been introduced to address this bias. However, these bias reduction methods cannot be applied efficiently to e-commerce cases because of the difference in the search results displayed to the user. The user behaviour in the list view search differs from that in the grid view search and is composed of three major factors- (1) middle bias (2) slower decay (3) row skipping. There have been efforts to model the user attention probability in a grid-based search. However, these efforts have not been considered item meta-information hence, we propose a method to incorporate the item attributes in the user attention estimation model. The idea is to add a non-positional aspect in the propensity model of the user behaviour patterns. This is based on a simple argument that the user attention is not only dependent on the row and column of the item(aka position) but also on the features of product tile. These features can be shipping promise, price, reviews, ratings etc. Some of these feature tags have been designed to attract the attention of customers. Through various experiments on Walmart search logs data, We show that the proposed framework outperforms the debias baseline algorithms. The results reflect better on how different taxonomies, product categories can impact the user behaviour in grid-based product search.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

1 Introduction

There is an inherent difference between traditional search engines and e-commerce or product search engines. In traditional search engines like Google, the search results are displayed in a list view whereas in the e-commerce search engines like www.walmart.com, the results are displayed in a grid view. The difference in the display of search results can impact the customer's attention and behaviour. Researchers in a previous study [5] were able to show that in the Image Search Engine Result Pages(SERPs), the user's attention can be defined by three factors: (1) User's attention within a row is more in the middle, so there is a bias towards the products in the middle (2) Users tend to skip rows while scrolling on a SERP (3) The decay of user attention is slower in a grid view compared to list view, where the decay happens drastically. This idea was further applied in the e-commerce search [6] and developed an inverse propensity bias model to correct the learning to rank algorithm. While position/presentation of the products on a SERP has been considered in the modelling of the bias, the impact of the product's meta-information on the user attention has still not been addressed.

Intuitively, the SERPs of the e-commerce search engines are not only different in the presentation but also in the fact that e-commerce is a marketplace which has a direct monetary benefit for the sellers maintaining the product information on the page. Thus, the information present for each product tile such as 'Top Picks', 'Reviews' and 'Ratings' would have a very strong impact on the customers navigating this web space to get the best product suited for their requirements. Figure 1 and 2 shows the difference in results displayed in list vs grid view on walmart web and application.

The impact of these qualitative product features on bias would be different for different categories of products. For example : On a category of Milk with clear intent, the new customer might not be exploring so much on the view presented whilst on a category of clothing, the user's attention would be influenced by these product tile tags as discussed above. Hence, we extend the framework to model the product meta-information like Ratings, Reviews, Shipping Promise in the propensity model. Incorporating these features into the model will show the impact on the features displayed on the product tile. To model these features, we use a scoring algorithm to assign a score to each product and this score in modelled with the propensity scoring model. Our aim is to learn a ranker and evaluate the impact of the model using an evaluation metric. We also look at the different product taxonomies like Entertainment, Fashion etc because the customer's intent is different in each product taxonomy.
We organize the rest of the paper in the following way: Section 2 contains related work on grid based e-commerce search, Section 3 contains our proposed framework with details on how the item feature scores are calculated and how the propensity score for modelling is derived, Section 4 mentions the experimental settings, evaluation metric and the results, along with the dataset description, and Section 5 presents the conclusion and the future work. Each section is further divided into subsections.

2 RELATED WORK

We will start with a brief introduction of the background and knowledge on the prior research work for analyzing the differences in the user behavior patterns in the list view vs the grid view search, and how this behaviour pattern is used in the propensity score modelling with a brief introduction to LambdaMART ranker.

2.1 GRID SEARCH USER BEHAVIOUR PATTERNS

Products in an e-commerce are placed in different view as compared to the majority of search engines that show the list view for the results. The different placement of products leads to a change in the interaction mechanisms of items. Previously, a number of studies have been done on the user behaviour on image or grid search engines [5, 8, 9]. All these studies focused on comparing the user interaction behaviour in grid search setting to list view display setting by using search logs and eye tracking methods. Important differences in user behaviour, user being more exploratory and more time spent by the user are some of the findings that were

Figure 1: List view on Walmart App

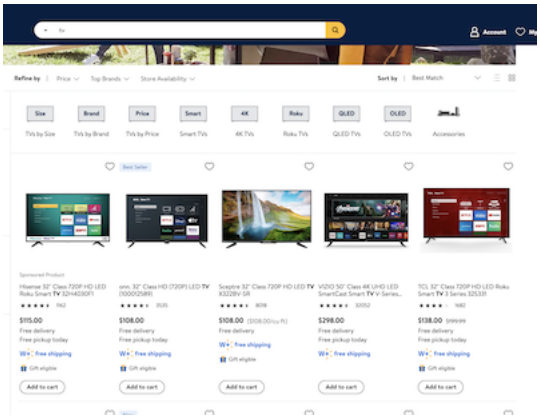


Figure 2: Grid view on Walmart web

observed. We list the major findings in grid based search as follows:

(a) Middle Bias: There exists a "middle position bias" in the user's examination behaviour. Customers allocate their attention more to the items placed in middle positions in a row. The probability of the user examining items placed on left and right corner in a grid search is less compared to the items in the center. In [5], the researchers consider the dwell time as the factor which is defined as the examination duration of an image and they confirm the middle bias phenomenon. Also, the user behaviour at each position in a row follows normal distribution. Researchers have not used this phenomenon in Learning to rank models of e-commerce yet, and we do not consider it in this research because in the dataset that we use, the number of columns is less than 4 or 5, and our hypothesis is that there are other details in e-commerce search that draw the user attention. So, the middle bias phenomenon is not very effective here.

(b) Row Skipping: The customers do not examine each and every product in a grid from top to bottom, they

tend to skip some rows. Users ignore particular rows and directly jump to some distant rows. The probability of the users skipping the first row is very low as compared to the other rows. In [6], row skipping bias in the user behavior was considered in getting an Unbiased Learning to rank model. The probability of the user skipping a particular row was considered as a bias factor when calculating the lambda gradient for a query. The observation was that the row skipping models performed better in taxonomies where user intent was very clear and users were looking for specific set of items. In such cases, users were probable to skip the rows.
(c) Slower Decay: In a list-view web SERP, the attention of the customer decays in monotonically decreasing fashion and is also dramatic but in grid-view, the decay is much slower. The user behavior in the grid search is more exploratory and there is more browsing as there are a lot of items displayed in a page and users don't have to go to next page to view the next set of items as in the list-view. This leads to the user spending more time in the grid search, concentrating on the items, and thus the decay is slow. In [6], the slower decay user behaviour bias was considered in training unbiased learning to rank models.
These findings are incorporated and Normalized Discounted Cumulative Gain(NDCG) evaluation method is also changed based on grid view search user behaviour.

2.2 Click Models

A lot of prior work has been done on-click modelling in the web search by extracting useful information from search logs of any search engine, mainly the click data. Cascade click model is one of the widely adopted technique in list based search that takes multiple types of user feed backs into account and models the user behavior as a sequence of actions [7, 10]. There are different types of biases in the click-models, mainly the position bias. Learning from biased data without taking them into consideration leads to a less effective ranking function. In another study, unbiased learning to rank model was introduced where position bias was considered. Inverse propensity weighting is a well known technique adopted to address any kind of bias and is used for unbiased learning [7]. Most of the work in this area assumes that the propensity scores are present in the logs and they study how to reduce the model variance while remaining unbiased. The unbiased learning-to-rank framework is different, because in that, the propensity is not explicitly logged and is buried in the user behavior data implicitly. Unbiased learning to rank does debiasing of the click data and uses it to train the ranker. Unbiased LambdaMART framework [4] was introduced for training an unbiased ranker by jointly estimating the biases at click positions and the biases at non-click positions.

2.3 LambdaMART Ranker

LambdaMART is a widely used Learning to rank algorithm to solve ranking problems in search engines. LambdaMART [2, 3, 11] uses gradient boosting and the gradient function of the loss function is called lambda function. The minimization of the objective function is performed using the lambda function on the training dataset. In LambdaMART, the lambda gradient of any document is calculated as

$$\lambda_i = \sum_{j:(d_i,d_j)} \lambda_{ij} - \sum_{j:(d_j,d_i)} \lambda_{ji} \tag{1}$$

λ_{ij} is defined as following :

$$\lambda_{ij} = \frac{-2}{1 + exp(2(f(x_i) - f(x_j)))} \left|\Delta_{ij}\right|$$

where λ_{ij} is the lambda gradient defined on a pair of documents d_i and d_j, σ is a constant,f(x_i) and f(x_j) are the scores of the two documents given by LambdaMART, Δ_{ij} denotes the difference between NDCG scores if documents d_i and d_j are swapped in the ranking list.

3 Item Propensity Scoring Model

We will start with a brief introduction of the background and knowledge on propensity estimation and joint optimization within the learning to rank models. Then we provide description of the changes to the proposed propensity scoring model using item features.

3.1 Estimating Propensity using Product Meta Information

We hypothesize that the user is not only influenced by the presentation of search query results but also with certain attributes or tags that have been designed to get customer's attention by sellers.
We consider four such attributes about an item which can impact user attention:
(1) **Two Day Shipping**: In the fast paced world that we are in currently, every user wants their favorite products to be delivered at the earliest. Users tend to compare delivery options with other online competitors and place their order accordingly. Thus, to get user attention on items that are eligible for super fast delivery, a message is shown on product tile that this item is eligible for delivery within 2 days. Fig 3,4 shows a two day shipping tag for products in the result set.
(2) **Ratings**: Ratings are given by previous users to the products on any e-commerce platform to indicate how satisfied or dissatisfied previous customers were for that product. This attributes tells what their experience was like and how they rate that experience on a scale of 1 to 5. Fig 3,4 shows the rating tag for products in the result set.
(3) **Reviews**: Reviews is the number of reviews a product has received till date. Reviews is a factor in fetching the rating score for an item. Users are attentive towards number of reviews as well along with rating, as an item with a high rating and very low number of user reviews is highly biased. Products with significant number of reviews gains user trust for that products rating.
(4) **Relative Price**: Price is an important information displayed to the user on the product tile. A set of expensive items for a query may not attract user attention and user may not look in details with much attention. This will counter the slower decay factor. With relative price score, we try to get the expensive and cheap products for a query that is displayed to user. This feature is the relative score among price of all products fetched in the recall set for a particular query.

Based on the above features, we estimate a score and experiment with it in the propensity model. In scoring method, We assign a binary value in the scoring functions for 2-day shipping, ratings and reviews.

According to the formulation:

$$S(I) = \gamma \sum_{i=0}^{3} K_i \tag{2}$$

where $K_i \epsilon [-1, 1]$ represents binary vote of the item property.
How to assign the vote for score estimation of an item? To assign a vote for scoring, we look at the quantiles for item features in the data. The threshold for assigning a positive vote or a negative vote is decided using quantile values and then the final score is estimated by the summation of all the votes. The scoring algorithm is mentioned in Algorithm 1.
For example:
If ratings >= 75th quantile-value, we assign a +1 score to the item or If ratings <= 25th quantile-value, we assign

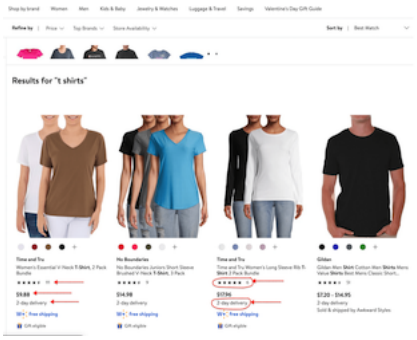


Figure 3: SERP with 4-column view with product meta information like Ratings, Reviews, Two day shipping

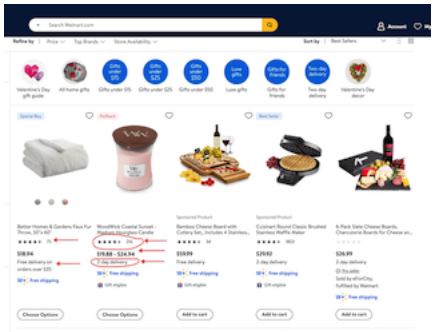


Figure 4: SERP with 5-column view with product meta information like Ratings, Reviews, Two day shipping

a -1 score to the item etc. In this similar fashion, we compute a score for each item based on these features.

Algorithm 1: Score estimation using product meta information

Initialize $score_i$ as 0
if $twoDayShipping_i > 0.0$ **then**
    $score_i \leftarrow score_i + 1$
end if
if $rating_i >= 75thQuantileValue$ **then**
    $score_i \leftarrow score_i + 1$
end if
if $rating_i <= 25thQuantileValue$ **then**
    $score_i \leftarrow score_i - 1$
end if
if $reviews_i >= 75thQuantileValue$ **then**
    $score_i \leftarrow score_i + 1$
end if
if $reviews_i <= 25thQuantileValue$ **then**
    $score_i \leftarrow score_i - 1$
end if
return $score_i$

3.2 Changing the Slower Decay Model

As mentioned before, in a SERP with a grid view the user attention decreases more slowly when compared to the results in a list view. According to the earlier work [6], the probability of examination of an item at position, i is

estimated as:

$$P(o^i = 1) = \prod_{j=0}^{i-1} min(\beta^{row(j)} * \alpha), 1.0) \tag{3}$$

where row(i) is the row-number for the item at position i, α is likeness of customer browsing the next item, β models the customer's patience in grid view. As we can see, the probability does not account for item meta-information shown on the item tile in SERP. If an user finds any item attractive in SERP, the user tends to spend more time on it and the probability of examination is more. There are multiple details shown on search page that may attract user attention viz. image quality or attractiveness, user ratings of that particular item, number of reviews on it, special offer tag, price of the product or the delivery date of the product. These factors will also contribute to the probability of user examining any particular product in Grid based SERP. Considering this, the decay factor will change accordingly. Based on this hypothesis, we model the estimated score above into the probability of examination at position, i as:

$$P(o^i = 1) = \prod_{j=0}^{i-1} min(\beta^{row(j)+S(I)} * \alpha), 1.0) \tag{4}$$

We train a ranker f based on loss function L with the propensity score of slower decay model computed using α, β and γ hyper parameters. We use LambdaMART with the ranker, gradient boosting trees as explained before. Following on the same lines, we calculate the lambda gradients by applying inverse propensity scoring. Hence, the lambda gradient for k^{th} product can be re-written as:

$$\frac{\delta L}{\delta x_k} = \lambda_k = \sum_q \left(\sum_{y_q^i = k \cap (i,j) \epsilon I_q} \frac{\lambda_{ij}}{P(o^i)} - \sum_{y_q^i = k \cap (j,i) \epsilon I_q} \frac{\lambda_{ij}}{P(o^j)} \right) \tag{5}$$

where i, j are the positions on the SERP, x_q is the feature vector for query-product pair, y_q^i means that the k^{th} product is at position, i and λ_{ij} is defined as following :

$$\lambda_{ij} = \frac{-2}{1 + exp(2(f(x_q^i) - f(x_q^j)))} \left| \Delta_{ij} \right|$$

Δ_{ij} is the value difference in the NDCG evaluation metric if i and j are to be swapped.

Relative Price Feature : Price is another information displayed on the product tile in an E-commerce search. Most of the users have a budget in mind when they visit to shop any product. If ,for a particular query, most of the items in search results are expensive compared to the range user is looking for, the user may not spend much time on that query viewing the results thus the decay will be comparatively faster. Among the set of items displayed, if there is a relatively cheaper product that also has other attractive tile meta-information, the probability of examining it will be higher. As a part of this work, we consider relative price feature separately to observe the impact of the price on the user behaviour and to check if there is any price bias . Relative price calculation is done as follows:

$$relp(i) = 1.0 - k * abs(func(log(x/mean\_price))) \tag{6}$$

where relp(i) is the relative price score for an item i and x is the price of that particular item , k is a constant and mean_price is the mean of prices of all items in the recall set of the query, abs is the absolute function to get the absolute value, func can be any activation function [14] like tanh or sigmoid . For example , tanh is defined as

$$tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

Sigmoid is defined as

$$sig(x) = \frac{1}{1 + e^{-x}}$$

Thus, in our dataset, we have the relative price score for all query-item pairs. Based on this, we redefine the probability of examination at position i in a GRID based SERP as

$$P(o^i = 1) = \prod_{j=0}^{i-1} min(\beta^{row(j)+S(I)+\eta*relp(i)} * \alpha), 1.0) \tag{7}$$

where η is the hyper-parameter introduced to tune the relative price feature in the slower decay model.

For training slower decay model including price feature, we calculate the lambda gradient in the gradient function of LambdaMART as above in (7).

4 Experiments

In this section, we conduct a series of experiments to compare our proposed method with the baseline methods for the removal of bias in grid-based e-commerce search. First, we explain our experimental settings, the datasets that we used, how we prepared them, and the metrics we used to evaluate the models. We then report the results obtained in our unbiased learning to rank setting.

4.1 Datasets

In this subsection, we provide a brief overview of the search log datasets that we used for our experiments. We used the data from the search logs of e-commerce website at Walmart, an American multinational retail corporation with a significant online presence. In particular, we use the data from three product taxonomies viz. Fashion, Entertainment (ENT), and Every Day Living (EDL), and by considering them for capturing and understanding the user behaviour and we eventually obtain four datasets, one for each category. Our datasets included queries, items, positions, and item features. The item features include raw features like item title match, item popularity, partial match score, along with engagement features like click rates, 'add to cart' rates, order rates and attribute features like the number of reviews, ratings, the relative price of the products among the recall set, and shipping promise.

4.2 Experimental Settings

In this subsection, we describe the experimental setup. The most common way to evaluate the learning to rank model is by its performance on a holdout set that has unseen data. We perform experiments on each product taxonomy, as mentioned in the subsection above. In our datasets, we perform a random split into a training set (80%), validation set (10%), and test set (10%). We carry out a fast grid search to obtain the best hyper-parameter values for the models. To keep the probability value in a reasonable range, we search alpha in {0.8, 0.85 , 0.9 , 0.95}, beta in {1.05 , 1.1 , 1.15. 1.2}, and gamma in {0.001 , 0.003 , 0.005 , 0.007 , 0.01 , 0.03 , 0.05 , 0.07 , 0.1}. For the LAMBDAMART ranker, we search for the learning rate in {0.01 , 0.05 , 0.1}, the maximum depth in {3 , 5, 7}, and the minimum data in leaf in {20 , 30 , 40 , 50 , 60}. We also tune for boosting fraction and feature fraction values in the range 0.7, 0.8, 0.9, 1.0 and 0.8, 0.9, 1.0 respectively. The other parameters are the default settings of Unbiased LambdaMART, while we keep the settings consistent for all the baseline and feature models. We consider the slower decay model[25] as the baseline method. Here, we optimize for the position 10 and consider the metric on the validation set to get the optimal parameter values.

4.3 Evaluation Metrics

Here, we explain the evaluation metric considered for the experiment. We consider the traditional evaluation method of Information Retrieval Systems, Normalized Discounted Cumulative Gain (NDCG) as the evaluation metric [1, 12, 13]. Experiments are performed in offline settings and we obtain the NDCG metric at different positions to compare the proposed framework with the baseline method.

$$NDCG_k = \frac{DCG_k}{IDCG_k} \tag{8}$$

where

$$IDCG_k = \sum_{i=1}^{|REL|} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

Here, IDCG@K is the normalizer and $|REL|$ is the list of documents ordered by engagement in the set up to position k. To explain the slower decay of user attention we consider K = 1, 3, 5, 10 for the NDCGs.

On the held-out dataset of Walmart Search logs, we calculate the prediction scores of our models. Using the prediction labels, we apply this evaluation method to get the metrics at different positions across different categories.

4.4 Experimental Results

In this section, we provide an overview of the experimental results that we obtain. We share insights on: (1) how e-commerce search results get better with the proposed framework, and (2)how the user behaviour varies across the product taxonomies. We mention the results in Table 2 and summarize our observations as follows:

- Our proposed method for slower decay using product tile information outperforms the baseline method in Fashion vertical by significant margin. This clearly shows the effectiveness of our proposed framework and assumption that user attention decay is slower when user is attracted by any of the item features.
- Performance of our proposed framework in Fashion affirms our initial assumption that users are more inclined on item tile information when shopping for clothing accessories and decay is much slower in such a case as user looks for detailed information.
- The proposed framework of using item attributes in slower decay probability computation shows improvement in Entertainment and Every Day living category too compared to the baseline method of slower decay. Performance improvement is not much compared to Fashion, and this may be because the users have a specific intent when they shop in such categories.
- Adding relative price feature in slower decay probability computation for fashion gave slight improvement over the proposed framework model without price feature. This suggests that price is not a major factor in the users' attempt to click on that product and view the details.

Taxonomy	α	β	γ
Fashion	0.8	1.05	0.01
EDL	0.95	1.15	0.001
ENT	0.9	1.15	0.03

Table 1: Tuned hyperparameters for item propensity scoring model for different taxonomies

Taxonomy	Framework	NDCG@1	NDCG@3	NDCG@5	NDCG@10
Fashion (w/o price)	Benchmark Slower Decay	0.408	0.486	0.535	0.584
	Proposed framework	**0.419**	**0.492**	**0.539**	**0.590**
Fashion (with price)	Benchmark Slower Decay	0.408	0.486	0.534	0.583
	Proposed framework(with price)	**0.420**	**0.494**	**0.540**	**0.590**
Every Day Living	Benchmark Slower Decay	0.553	0.627	0.669	0.711
	Proposed framework	0.553	**0.628**	0.669	0.711
Entertainment	Benchmark Slower Decay	0.503	0.580	0.624	0.669
	Proposed framework	0.503	**0.581**	0.624	**0.670**

Table 2: Experimental Results on different taxonomies using held-out dataset.

We train separate models for different taxonomies to show that the user behaviour patterns differ across categories and the users react differently to the product tile features.

Hyper Parameter Values: Here, we report the values that we obtain for the hyper parameters (α , β , γ) and that achieve the optimal performance for the proposed framework. For the Fashion category, the proposed slower decay model with α = 0.8, β = 1.05 and γ = 0.01 outperforms the others. Similarly, for the Every Day Living category, the proposed slower decay model with α = 0.95, β = 1.15 and γ = 0.001 works best. α = 0.9, β = 1.15 and γ = 0.03 are the optimal hyper-parameter values for the Entertainment category with the proposed slower decay framework. We experimented on Fashion with the price feature and α = 0.95 , β = 1.15 , γ = 0.05 , η = 0.01 are the optimal hyper-parameter values for our dataset.

5 Conclusion and Future work

In this paper, we tried to understand the impact of meta-information on the products displayed on the SERPs for user attention. We extended the solution of debiasing the grid product search by modelling the meta-information of the products, like rating, reviews, shipping promise, price in the propensity model. We showed through extensive experimentation how the different product taxonomies have different user behaviour.

As part of future work, we plan to extend this work to incorporate more product tile features and also use the image quality feature to model the user attention in Grid-based E-commerce search engines. We plan to try out different scoring methods as well for tile features score calculation and model middle bias , row skipping behaviour based on product tile features.

References

[1] K. Järvelin and J. Kekäläinen. Cumulated Gain-Based Evaluation of IR Techniques. *ACM Trans. Inf. Syst*, 10.1145/582415.582418, 2002.

[2] C. JC Burges. From RankNet to LambdaRank to LambdaMART: An Overview. *https://www.microsoft.com/en-us/research/publication/from-ranknet-to-lambdarank-to-lambdamart-an-overview/*, MSR-TR-2010-82, 2010.

[3] Z. Cao, T. Qin, T.Y. Liu, M.F. Tsai and H. Li. Learning to Rank: From Pairwise Approach to Listwise Approach. *Proceedings of the 24th International Conference on Machine Learning*, 10.1145/1273496.1273513, 2007.

[4] Z. Hu, Y. Wang, Q. Peng and H. Li. Unbiased LambdaMART: An Unbiased Pairwise Learning-to-Rank Algorithm. *The World Wide Web Conference*, 10.1145/3308558.3313447, 2019.

[5] X. Xie, J. Mao, Y. Liu, M. de Rijke, Y. Shao, Z.-Ye, M. Zhang, S. Ma. Grid-Based Evaluation Metrics for Web Image Search. *The World Wide Web Conference*, 10.1145/3308558.3313514, 2019.

[6] R. Guo, X. Zhao, A. Henderson, L. Hong and H. Liu. Debiasing Grid-Based Product Search in E-Commerce. *AProceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*,10.1145/3394486.3403336, 2020.

[7] X. Wang, N. Golbandi, M. Bendersky, D. Metzler and M.-Najork. Position Bias Estimation for Unbiased Learning to Rank in Personal Search. *Association for Computing Machinery*, 10.1145/3159652.3159732, 2018.

[8] B. Geng, L. Yang, C. Xu, X.S. Hua and S.-Li. CThe Role of Attractiveness in Web Image Search. *AProceedings of the 19th ACM International Conference on Multimedia*, 10.1145/2072298.2072308, 2011.

[9] X. Xie, J. Mao, Y. Liu, M. de Rijke, H. Chen, M. Zhang, S. Ma. Preference-Based Evaluation Metrics for Web Image Search. *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 10.1145/3397271.3401146, 2020.

[10] N. Craswell, O. Zoeter, M. Taylor and B. Ramsey. CAn Experimental Comparison of Click Position-Bias Models. *Proceedings of the 2008 International Conference on Web Search and Data Mining*, 10.1145/1341531.1341545, 2008.

[11] S K. K.Santu, P. Sondhi and C. Zhai. On Application of Learning to Rank for E-Commerce Search. *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 10.1145/3077136.3080838, 2017.

[12] Sham. S. Discounted Cumulative Gain. *https://machinelearningmedium.com/2017/07/24/discounted-cumulative-gain/*, 2017.

[13] Chandekar. Pranay. Evaluate your Recommendation Engine using NDCG. *ttps://towardsdatascience.com/evaluate-your-recommendation-engine-using-ndcg-759a851452d1*, 2020.

[14] A. Sharma V. Understanding Activation Functions in Neural Networks. *https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0*, 2017.

Graph Neural Networks for Inconsistent Cluster Detection in Incremental Entity Resolution

Robert A. Barton, Tal Neiman, Changhe Yuan

Abstract

Online stores often utilize product relationships such as bundles and substitutes to improve their catalog quality and guide customers through myriad choices. Entity resolution using pairwise product matching models offers a means of inferring relationships between products. In mature data repositories, the relationships may be mostly correct but require incremental improvements owing to errors in the original data or in the entity resolution system. It is critical to devise incremental entity resolution (IER) approaches for improving the health of relationships. However, most existing research on IER focuses on the addition of new products or information into existing relationships. Little research has been done for detecting low quality within current relationships and ensuring that incremental improvements affect only the unhealthy relationships.

This paper fills the void in IER research by developing a novel method for identifying inconsistent clusters (IC), existing groups of related products that do not belong together. We propose to treat the identification of inconsistent clusters as a supervised learning task which predicts whether a graph of products with similarities as weighted edges should be partitioned into multiple clusters. In this case, the problem becomes a classification task on weighted graphs and represents an interesting application area for modern tools such as Graph Neural Networks (GNNs). We demonstrate that existing Message Passing neural networks perform well at this task, exceeding traditional graph processing techniques. We also develop a novel message aggregation scheme for Message Passing Neural Networks that further improves the performance of GNNs on this task. We apply the model to synthetic datasets, a public benchmark dataset, and an internal application. Our results demonstrate the value of graph classification in IER and the ability of graph neural networks to develop useful representations for graph partitioning.

1 Introduction

Entity resolution, the problem of grouping related entities together, is an important problem in computer science with many application domains, such as databases (when linking records from different databases that represent the same object) or natural language processing (disambiguating nouns that refer to the same entity) [2, 6, 12]. Much of the research in entity resolution focuses on static systems where the resolution is performed as a one-time exercise. In many applications, however, a large repository of existing relationships must be continually updated in order to maintain correct groupings of entities. Incremental entity resolution, or dynamic entity resolution, studies the problem of how to update repositories of existing relationships over time [15, 24, 33]. Studies of incremental entity resolution typically focus on how to adapt to new entities and information as they are received.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

However, another important goal in practice is to make the resolved entities incrementally better while not decreasing quality [32]. Where cluster quality is important, we must detect incorrect clusters of relationships with high precision to ensure that healthy clusters are maintained. Incorrect clusters of related entities can arise, for example, from poor quality in the original data, prior incorrect clustering decisions, changing item data, or malicious actors that fool existing entity resolution systems. We refer to clusters or families of incorrectly resolved entities as inconsistent clusters (IC).

As an example of where Inconsistent Cluster detection is important, consider entity resolution in the domain of e-commerce products. Stores such as Amazon, AliExpress, WalMart, and eBay provide huge selections of products for customers. These platforms often utilize product relationships to link relevant products together or link them to customers. Examples of product relationships include duplicates, complements, substitutes, and accessories. Many entity resolution approaches have been proposed for these product matching problems [21, 23, 27]. Another useful but less researched relationship is product variations, which refer to families of products that are functionally the same but differ in specific attributes. As an example, shoes that are identical in all aspects other than size and/or color are variations of one another and are shown to customers on a single page to simplify selection. In this case, the entire resolved cluster is shown to the customer, and the inclusion of any shoe that does not belong will make the whole family inconsistent. In this case, IC defects are directly visible to customers and are critical to fix. More generally, inconsistent clusters of items can be detrimental in any entity resolution system.

Conventional entity resolution methods were not developed to detect ICs with high precision for IER. In the conventional approach, a matching step generates the probability for a relationship between all pairs of items. Matching is followed by a clustering step, wherein clusters are formed by applying a graph processing algorithm such as transitive closure to all edges above a threshold value [2, 19]. To determine whether existing clusters should be divided, one could apply matching and clustering within all existing families. However, it is possible for pairwise scores to conflict [26], leading to incorrect cluster formation and preventing existing Inconsistent Families from being detected. It has been observed that different clustering algorithms have different strengths and weaknesses [19]. We propose instead a classification task to evaluate cluster quality that generates a score for any given cluster being inconsistent. In addition to allowing us to escape the weaknesses of any particular graph processing algorithm, this approach has the benefit that the score can be tuned to ensure high precision identification of ICs. Before either forming a new cluster or disaggregating an existing cluster, IC detection can be used to ensure high confidence that every cluster is incrementally improved. This ability is especially useful when the quality of each individual cluster is important such as in the product variations problem above.

We formulate the IC detection problem as a graph classification problem on weighted undirected graphs, where edges represent the similarities between each node. We then develop a binary graph classifier to predict whether the candidate items jointly form a consistent cluster. Although we have motivated it from an entity resolution perspective, the problem represents a rich area of study for graph classification since the data are well represented in graph form with node features associated with entities and pairwise similarity scores associated with edges. We apply graph neural networks [5, 14, 16, 20, 30] to the problem of scoring a group of entities for whether they belong together. We demonstrate that existing Graph Neural Network architectures perform well at graph classification on weighted graphs, and we develop a novel message representation that concatenates multiple node aggregation functions together to improve performance. We tested our methods in a variety of problems, including an e-commerce application, open data, and synthetic datasets, and demonstrate the general superiority of the proposed Graph Neural Network approach.

To summarize, we make the following major contributions in this work. First, we introduce the Inconsistent Cluster detection problem as an important problem in Incremental Entity Resolution and demonstrate its usefulness on a real-world product variations dataset. Second, we develop a Graph Neural Network-based binary classifier for solving the Inconsistent Cluster detection problem. We demonstrate a novel message aggregation scheme that significantly improves the performance of a GNN classifier on this task, and show that its performance exceeds state of the art. Third, our extensive evaluations shed light on the superior performance of the proposed method

against several existing methods. The results show that existing methods may still work well when positive and negative edges are reasonably separated. But when the scores become more mixed and noisy, the advantage of our graph classifier is clear. Finally, we demonstrate that Graph Neural Networks can learn robust representations of weighted graphs for reasoning about partitioning decisions, which we believe will be a valuable area of study for graph representation learning.

2 Background

In this section, we review several areas of research that are closely related to this work.

2.1 Product matching

Much existing research on product relationships in e-commerce focuses on product matching, e.g., duplicates, complements and substitutes. Product matching is usually formulated as an entity resolution problem [6, 12]. Konda et al. described an end-to-end entity resolution system that includes steps from data preprocessing to product matching [21]. More recently, many deep learning-based methods have been developed for entity resolution. Mudgal et al. evaluated several deep learning-based approaches for entity matching and found that deep learning-based methods do not outperform existing approaches in more structured domains, but can significantly outperform them on textual and noisy data [23]. Shah et al. discussed two approaches for product matching, including a classification-based shallow neural network and a similarity-based on deep siamese network [28]. Ristoski et al. used both word embeddings from neural language models and image embeddings from a deep convolutional neural network for product matching [27]. Li et al. studied using product titles and attributes for product matching and evaluate their method in two application scenarios, either as a classification problem or as a ranking problem [22]. Datasets that are specifically targeted for product matching have been published. For example, a WDC dataset has been published by [25] for large-scale product matching. Its English language subset consists of 20 million pairs of duplicate products that were extracted from 43 thousand e-commerce websites.

Another closely related area of research is product graphs [9, 18]. A product graph is a framework to capture all aspects of entities in a unified representation, including both product attributes and the relations between them. Such a unified view is beneficial because identifying relevant entity attributes and learning relationships are tightly coupled and can benefit from being treated jointly.

2.2 Incremental entity resolution (IER)

IER has been studied previously with different objectives. [33, 34] address the problem from the perspective of evolving matching rules, and point out that the IC problem arises in cases where a stricter matching rule requires all existing clusters to be reevaluated. [15] discusses IER following record insertion, deletion, and change and allows existing clusters to split apart in response to these operations. The main difference between these works and our own is that we focus on the precision of updates to individual clusters rather than relying on the ER clustering algorithm to cluster optimally. These goals are important in an IER system that may already have relatively high cluster quality that needs to be preserved.

Repairing existing relationships has also been studied from an ER perspective [32] via the creation of a provenance index that tracks how clustering has changed over time. In this work, we identify clusters that need repairs with high precision when the provenance may not be known.

2.3 Collective entity resolution

Collective entity resolution, broadly defined, treats clusters of products jointly in entity resolution [2]. Many existing approaches to collective entity resolution can be described as a two-step process: matching and clustering.

The matching step can be solved with the methods reviewed in the previous section and outputs scores of all edges in a cluster. The clustering step can also be solved using many methods. One popular method is *transitive closure* (TC) [2, 19]. In this method, edges below a critical threshold t are removed and connected components are computed for remaining edges. *K-Core* is another common method for the clustering step. In graph theory, k-cores of the graph are connected components that remain after all vertices with degrees less than k are recursively removed. These methods can be evaluated on our task by whether they split a cluster into more than one partition after their application.

Finally, a more recent method called *SuperPart* is a cluster scoring model [26]. SuperPart directly formulates classification of a weighted graph as having one or multiple partitions as a binary classification problem. The SuperPart model featurizes a graph of pairwise edge scores by calculating graph-level features such as the mean external edge (mean of the edges removed by transitive closure at a fixed threshold), the average edge score in the cluster, and the number of connected components after filtering out edges below various thresholds. A random forest is used to extract a cluster-level score from the table of features that represents a likelihood for the cluster to remain as a single partition. To our knowledge, SuperPart is the only prior algorithm that assigns scores to evaluate cluster quality. We will use it as a baseline algorithm in this work.

2.4 Graph neural networks

Graph neural networks (GNNs) have been an increasingly active research area in recent years [17]. GNNs represent relationships between a set of entities as a graph, allowing the incorporation of both entity-level features as well as edge-level features. Many real-world problems can be represented as graphs, such as social networks [20], chemical and biological compounds [10, 13, 30] and product graphs [5]. One widely applicable formulation of graph neural networks, Message Passing Neural Networks [13], uses message propagation to iteratively update embedding representations of nodes, such that the embeddings capture the common patterns between node features directly communicated via edge features. Graph representation learning has been studied extensively, e.g., [5, 14, 16, 20, 30]; however, GNN research has focused more on learning node embeddings than on graph classification. Gilmer et al. performed graph-level regression to predict 13 chemical properties on a large graph dataset, whereas here we have a graph-level classification task related to the graph partitioning [13] . Chen et al. perform graph partitioning at the node level given a fixed number of partitions [5]. This task is similar to the classification task described here in that it involves graph partitioning, but is dissimilar in that we have weighted rather than binary edges and are classifying entire graphs rather than nodes. Graph Neural Networks have been explored for entity linkage in knowledge bases in Neural Collective Entity Linking (NCEL) [4], wherein the best candidate edge for a given node is found using an existing clustering. Overall, graph neural network models are particularly compelling for entity resolution because of their flexibility: they can learn from both node features and edge features and make predictions for nodes, edges, or entire graphs. To our knowledge, graph neural networks have not yet been applied to evaluating clusters in entity resolution, for which many applications still use the traditional approach of calculating similarity scores followed by algorithmic clustering approaches.

3 Method

3.1 Problem

Product variations are increasingly used by e-commerce websites to group similar products into families so that each family can be shown on a single detail page. Again taking shoes as an example, the shoes that share key aspects such as brand and style would be displayed on the same page; non-key aspects such as size and color are shows as selectable (dropdown) options. Such aggregated views of products allow customers to select specific products much more easily. Since all varied items are shown together, even a single item incorrectly grouped with others creates a visibly inconsistent family. Importantly, stores such as WalMart, Amazon, and Shopify offer

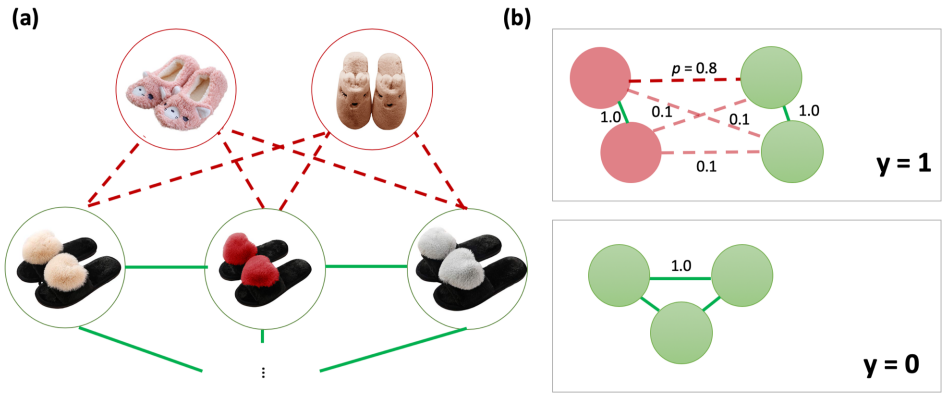


Figure 1: (a) Example of an IC in the context of product variations. Two items representing animal-themed slippers should be separated from traditional Womens' slippers. (b) Generic binary graph classification problem. Existing fully connected graphs of items contain both correct relationships (solid green lines), and incorrect relationships (dotted red lines) according to auditor labels. The edge labels encode a partitioning of the graph into one subgraph in the case of healthy families ($y = 0$) or multiple subgraphs in the case of IC ($y = 1$). The goal of the classifier is to find the correct graph label y without knowing the true partitioning. The classifier relies on a probability for edge existence p derived from pairwise item similarity. As in the IC family in (b), the task is challenging for algorithmic approaches because p might vary across two true clusters.

sellers the opportunity to upload information about variations directly, allowing for the possibility that incorrect clusters can arise as a result of input data rather than as the output of an entity resolution system.

In this paper, we focus on the problem of detecting whether a family of items form a true cluster, which we define as the *Inconsistent Cluster* (IC) problem. Due to the need to treat product families collectively, IC detection can be much more challenging than simple pairwise entity matching. Imagine a family of products with 10 members, and only one of them does not belong. The family as a whole is clearly *defective*. However, if we consider the problem as pairwise product matching, a majority of the 45 product pairs (10 choose 2) are correct matches, with only 9 of them as mismatches. Although we can add a clustering step to further decide whether the family is defective or not, classic clustering approaches such as transitive closure or k-core may not work well for real world problems where noisy data are prevalent [26]. For example, one or two of the 9 scores for mismatched edges may be high.

Importantly, because we use an independent algorithm to classify clusters into consistent and inconsistent clusters, our method can detect errors in the decision made by an entity resolution system. This is an important contrast with other IER approaches because, as our data will show, clustering methods used in entity resolution do not always make correct decisions. Even in this context of noisy pairwise scores, our method can be used to enforce high precision for resolved clusters.

3.2 Formulation

We solve the IC detection problem by formulating it as a binary graph classification problem. Given an input graph $G(V, E)$, a signal $y \in \{0, 1\}$ is to be reported for each graph (Figure 1). A graph with a negative label corresponds to a healthy family and a positive label corresponds to an IC. As noted before, this graph classification task arises not only in finding existing families that are IC, but in confirming that items to be combined have a healthy score. The nodes represent items and all candidate families of items are fully connected with edge scores p_{ij} that represent item similarity between items i and j.

For maximum generality, in this work, we assume only that there is a means available to produce the pairwise

node similarity feature p between all candidate related items. We create these pairwise edge scores for an internal e-commerce application, the inconsistent variation family detection problem, by using a random forest on edge features generated from each pair of records; these edge features are generated by a general record matching platform similar to [8]. We also test the method on open data from [26] which contains weighted graphs with ground truth partitionings. Finally, we can test our method on synthetic data. For this purpose, we adopt the Weighted Stochastic Block Model [1], wherein scores between two partitions come from one exponential family distribution, and scores within a given partition come from another.

3.3 Baseline Methods

One common method for making partitioning decisions in entity resolution is transitive closure (TC). In this method, edges below a critical threshold t are removed and connected components are computed for remaining edges. When enough edges fall below t to divide a candidate family into two or more connected components, we predict $y = 1$ for the family-level label; otherwise, we predict $y = 0$. Unlike other models studied here, this method does not provide a continuous score for how the cluster is likely to be classified. However, we note that at a given threshold t, TC is a binary classifier which considers more families to be inconsistent as t increases. We can therefore create a precision-recall curve for the purpose of comparing to other graph classifiers by sweeping t. To do this, we construct a family-level score $s_i^{TC} \equiv 1 - t_i^{min}$ where t_i^{min} is minimum pairwise threshold at which family i will have multiple connected components. If t_i^{min} is low (s_i^{TC} is high), then the cluster breaks apart easily; however, if t_i^{min} is high (s_i^{TC} is low), even edges with high scores must be filtered out to divide the cluster. We use the score s_i^{TC} as a binary graph classification score for the purpose of constructing a precision / recall curve, where only families with score greater than or equal to a family-level threshold t_f will be considered IC.

For k-core, we allow tunable $k \in [3, 9]$, which is chosen in the validation job to optimize the F_1 metric we report. The k-core algorithm recursively prunes nodes with degree less than k. As for TC, we can calculate a PR-curve for k-core using the minimum pairwise threshold at which k-core would separate or reduce the size of a family.

We also implement the SuperPart algorithm from [26], which explicitly solves the classification task of deciding whether a weighted graph should be broken into multiple partitions. To our knowledge, SuperPart is the only existing graph classification method intended to classify clusters according to whether they should be partitioned.

3.4 Proposed Model

Going beyond prior work, we propose to solve the graph classification problem by applying graph neural network models (GNNs). In particular, we adopt the message passing neural network (MPNN) framework outlined in [13] but propose a new message aggregation scheme that significantly improves the classification performance. MPNNs can be described by their message passing, update, and readout functions. Given a graph G with node features x_v and edge features e_{vw}, the MPNN works by aggregating (AGG) messages M_t at each node v from neighboring nodes $N(v)$ at each timestep t. These collected message m_v^t can depend both on the neighboring node features as well as the edges. The messages are then used to update the hidden state h_v of the node according to an update function U_t. Finally, after T timesteps, the graph label is read from all node states according to a readout function R:

$$m_v^{t+1} = \text{AGG}_{w \in N(v)}[M_t(h_v^t, h_w^t, e_{vw})] \tag{9}$$

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1}) \tag{10}$$

$$\hat{y} = R(\{h_v^T | v \in G\}) \tag{11}$$

Table 3: Performance on real and synthetic data.

Dataset	TC	K-Core	SuperPart	GCN-E	MAG-GCN (Ours)
Softlines	0.716 (0.006)	0.624 (0.014)	**0.721** (0.005)	0.705 (0.005)	**0.721** (0.009)
Hardlines	0.873 (0.008)	0.663 (0.002)	0.880 (0.004)	0.877 (0.003)	**0.887** (0.003)
IC-Stratified	0.725 (0.012)	0.746 (0.026)	0.754 (0.023)	0.757 (0.003)	**0.774** (0.003)
$Beta$(1,1), $Beta$(3,1.5)	0.375 (0.047)	0.433 (0.041)	0.619 (0.071)	0.576 (0.061)	**0.687** (0.061)
$Beta$(2.5,4), $Beta$(4,2)	0.706 (0.057)	0.640 (0.067)	0.819 (0.04)	0.737 (0.039)	**0.843** (0.039)
$Beta$(1,2), $Beta$(4,1.5)	0.624 (0.051)	0.762 (0.051)	0.914 (0.028)	0.813 (0.048)	**0.932** (0.029)
$Beta$(1.5,4), $Beta$(3,1)	0.907 (0.03)	0.856 (0.04)	0.95 (0.016)	0.825 (0.037)	**0.967** (0.014)
$Beta$(1.5,4), $Beta$(4,1)	0.936 (0.035)	0.924 (0.028)	**0.976** (0.019)	0.907 (0.036)	0.973 (0.009)
$Beta$(1,4), $Beta$(4,1)	0.979 (0.01)	0.972 (0.016)	0.981 (0.02)	0.93 (0.025)	**0.989** (0.014)

We report mean F_1 score with parentheses enclosing standard error.

Of the MPNNs, we draw on the design of the Graph Convolutional Network (GCN) [20] and ideas from [35] to produce an architecture that performs well at the task of deciding whether a weighted graph should be divided into one or more partitions. The message passing function for our proposed approach, the MAG-GCN (Mixture AGgregation Graph Convolutional Network), is

$$m_v^{t+1} = (\text{MEAN}(h_w, e_{vw}), \text{MAX}(e_{vw}), \text{MIN}(e_{vw})), \tag{12}$$

where $(\cdot, \cdot)$ denotes concatenation. We found the concatenation of the edge feature to the node features be important, as observed in [11]. Moreover, the inclusion of the min and max aggregation represents a significant departure from the message passing function of the GCN: $m_v^{t+1} \sim \text{MEAN}(h_w)$. We found that for reasoning about partitioning in a weighted graph, it was helpful for the message passing function to carry more information about the distribution of incoming edge weights than just the mean. To demonstrate these points, we compare the MAG-GCN to an architecture that we refer to as GCN-E [11], which is the same but with message passing

$$m_v^{t+1} = \text{MEAN}(h_w, e_{vw}). \tag{13}$$

We also tried adding a sum aggregator to our mixed aggregation, but found that this did not yield any benefit. We also note that for the graphs studied here, where all of the information is contained in the edge weights, it is not obvious how to adopt a vanilla GCN architecture such as that in [20], where the focus is on learning from node features. After completion of this work, we became aware of [7], which demonstrates that a similar mixed aggregation scheme enhances GNN performance in a broad array of applications.

For all of the GNN architectures we studied, the update function consists of a fully connected neural network f_U with ReLU activation, and the readout function consists of a fully connected network that operates on the mean of the hidden states $\bar{h}_v$, $R = f_R(\bar{h}_v)$. Following readout the loss is calculated with binary cross-entropy. We utilized learning rates between 10^{-2} and 10^{-3}. Initial node features were set to a vector of zeros and the edge weights e_{vw} were set to the pairwise edge similarity. Hyperparameters were tuned on synthetic datasets. Our graph neural network models were built with Deep Graph Library [31], a general framework for building Message Passing Neural Networks.

4 Results

4.1 Experimental design

To evaluate model performance, we choose transitive closure, k-core, and SuperPart as baseline methods. For evaluating performance on a test set, we use F1 score. For the F1 score metric, a threshold is first chosen to

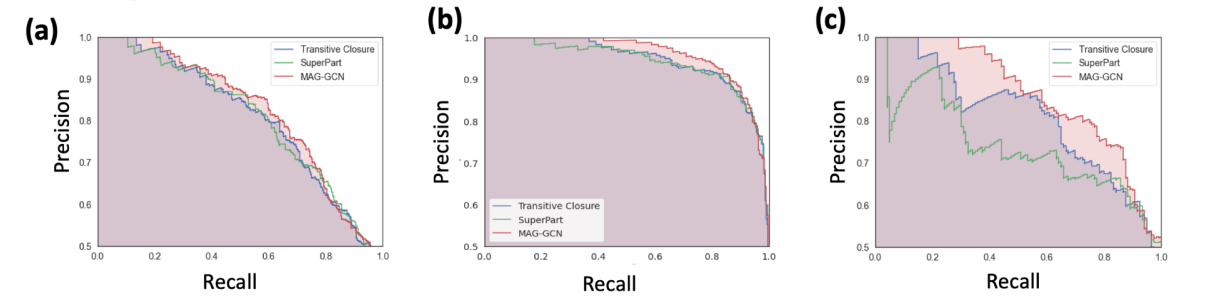


Figure 2: Performance of cluster scoring on: (a) Softlines , (b) Hardlines, (c) An open dataset introduced in [26] and referred to in that work as IC-Stratified. F1 scores for the same datasets can be found in Table 3.

maximize F1 score on a validation set, and the F1 score is evaluated on a test set.

We first evaluated all of the methods in an internal application. Several audited datasets are created as follows. A group of items was shown to auditors on a single page, and the auditors were given the task of partitioning it into one or more Variation families. One audited dataset consists of approximately 8,500 such families from a Softlines product line containing 76,000 items which have been partitioned according to auditor labels; approximately 3,800 of these families were split by the auditors and are therefore IC. The second audited dataset consists of approximately 6,500 families from a Hardlines product line containing 124,000 items; 2,700 families are IC.

We also use a publicly available dataset released in [26] called IC-Stratified; this dataset contains a training set with 2649 nodes and a test set with 2424 nodes with ground truth partitionings into one or more families.

Finally, we evaluated our methods on synthetic datasets. When generating the datasets, we tested different degrees of separation between positive and negative pairs, as well as the specific distributions for generating the weights of these pairs. The systematic study on synthetic datasets allow us to gain deep insights on the advantages and disadvantages of the different methods under different conditions. We generate synthetic data by creating graphs of 1 or 2 ground truth families, where the latter is an IC example. Each node had probability of 0.97 to be part of the first true family and 0.03 to from the second. All edges within true families were drawn from one Beta distribution, and all edges between true families were drawn from another.

4.2 Real-World Datasets

For the Softlines and Hardlines datasets, we trained and tested the models on a fixed training, validation, and test set 5 times, each with a different random seed for the pairwise classifier training. For IC-Stratified, pairwise scores are provided and we split the training set into training and validation 5 different ways. A comparison of the cluster scoring models can be found in Table 3, with PR-AUC curves shown in Figure 2. Overall, we find that a model-based cluster scoring mechanism helps with the task of identifying inconsistent clusters, with both SuperPart and MAG-GCN outperforming the TC baseline. We find that the MAG-GCN model performs the best overall in Softlines, Hardlines, and the public dataset IC-Stratified from [26]. The GNN model is particularly effective at improving recall at high precisions, an important metric for applications. An example precision-recall comparison between the MAG-GCN and the TC baseline can be found in Figure 2.

To understand the origin of the improved performance, we plot the GNN score against the TC score in Figure 3. We observe that the MAG-GCN derives its improved performance by boosting the scores of TC's false negatives more than reducing the scores of false positives. We also observe that larger families tend to benefit more often from the GNN than their smaller counterparts. This pattern can be better understood by examining some examples. The lower portion of Figure 3 shows example families that were correctly labeled positive by the GCN but incorrectly by transitive closure. We see that while the pairwise model correctly drew low-probability edges between clusters in many cases, there is often a structural component in the graph that makes it difficult to

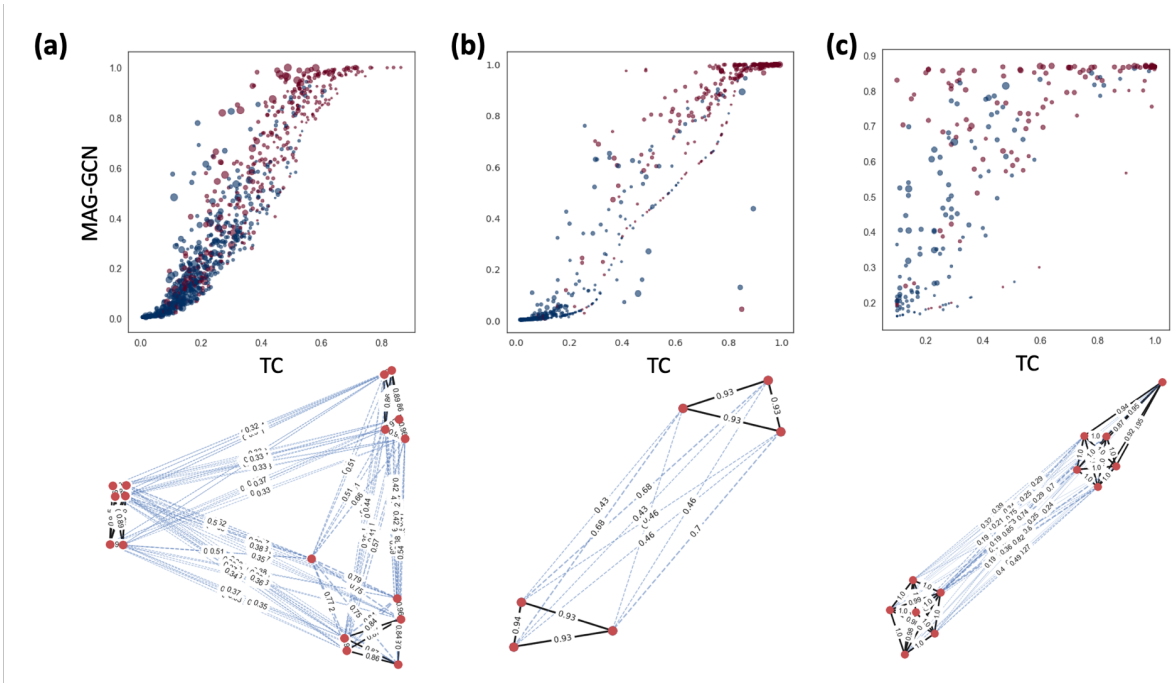


Figure 3: Study of improved performance. Upper: MAG-GCN score is displayed against the TC score for (a) Softlines, (b) Hardlines, (c) IC-Stratified. Blue points are ground truth negative ICs, red points are ground truth positive ICs, and size represents the number of items in the family. The upper left hand corner of the graph represents the area where the GCN increased the score compared to TC. The primary effect of the GCN is to boost the scores of positive examples more than negative examples. Lower: example families of products that were classified as non-IC by TC but were correctly classified as IC by MAG-GCN. The red points represent items, black edges represent true positive edges, and blue dotted lines represent true negative edges. The width of the line is proportional to the pairwise model prediction p. For example, in the rightmost plot, owing to a single edge with weight 0.85 between two true clusters, TC did not suggest partitioning into two families with high likelihood (score = 0.15); however, MAG-GCN correctly did (score was 0.78, true label was 1.0)

partition by transitive closure. For example, in the example from Softlines (a), there is a central node that holds three separate clusters together. In the examples from Hardlines (b) and IC-Stratified (c) there are widely varying edges between two true clusters.

4.3 Synthetic datasets

A study of synthetic data is shown in Figure 4. As mentioned previously, we produced artificial families of items ranging from size 5 to 15 using a different distribution for true positive edges and true negative edges. By varying the parameters of the distribution, we were able to determine how methods of cluster scoring perform across different levels of difficulty. The distributions and corresponding performances are shown in Figure 4 and Table 3. We find that MAG-GCN consistently outperforms or is competitive with the other methods, with particular performance increases for noisy data. For example, when positive edges are drawn from $Beta(3, 1.5)$ and negative edges are drawn from a uniform distribution – a noisy setting – MAG-GCN has an F1-score of 0.687 vs. 0.375 for TC and 0.619 for Superpart - an 83% and 11% improvement over the baselines, respectively. However, when both positive and negative distributions are well-separated, most methods do well. This points to the usefulness of our method in the noisy data setting, which is common in applied settings such as our internal task.

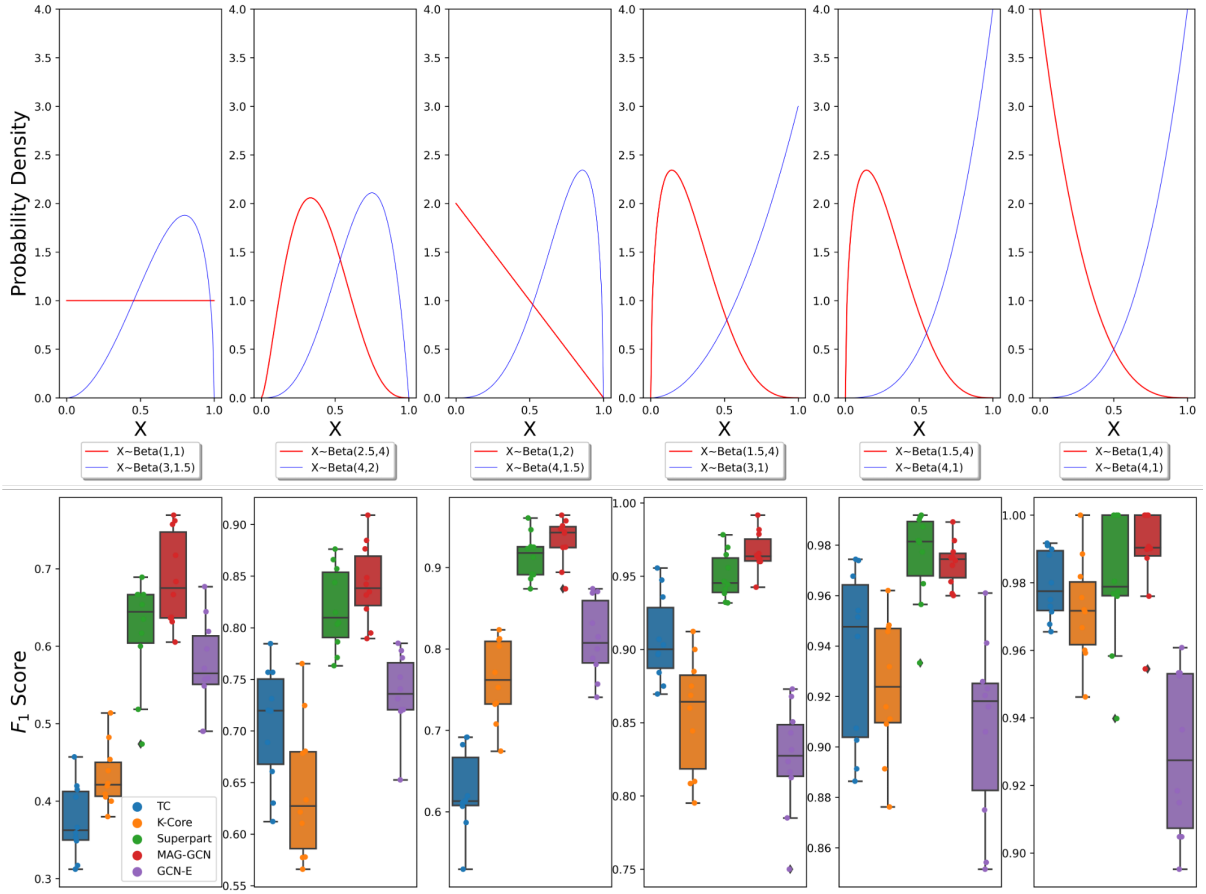


Figure 4: Performance on synthetic data. Top: beta distributions used generate synthetic data. Negative edges are drawn from the red distributions while positive edges are drawn from the blue distribution. Bottom: different cluster scoring techniques are applied to each distribution. We utilize the synthetic datasets from Table 3 (N = 1000).

We also tested all methods against the GCN-E, which is similar to MAG-GCN but uses only mean node aggregation rather than the mixture of mean, min, and max. We note that the concatenation of the edge feature into a GCN alone is insufficient to learn this task as successfully as our method.

5 Discussion

As in previous work [26], we find that by applying an additional cluster scoring model we can improve performance on the joint task of classifying an entire family as inconsistent. Of the cluster scoring mechanisms we tested, we find that the MAG-GCN is most effective, outperforming its immediate competitor SuperPart. We also found that the benefit of the MAG-GCN is most dramatic for recall in the high precision regime, which is a relevant metric for production use cases where we wish to fix only high-confidence IC families. The performance of the MAG-GCN in comparison to SuperPart was also interesting. We noticed that when little data was available, SuperPart performed well and once better than MAG-GCN (second to last plot in Figure 4), suggesting that the SuperPart model may be better regularized. However, MAG-GCN appears to be more flexible and able to learn effective representations of a wide variety of weighted graphs. Understanding the power of Graph Neural Networks relative to featurized representations of graphs is an interesting topic for further study.

We also make note that MAG-GCN is significantly better than GCN-E, where the edge feature is simply concatenated and the aggregation function is the mean. Xu et al. [35] note the mean aggregation function is effective at learning the distribution of its surroundings while the max aggregation function is more effective at distinguishing outliers. We note that the mean aggregation function provides information about the center of the distribution of incoming edges, while the min and max aggregation functions provide some sense of its spread.

To frame the results in terms of our IC detection application, the decision at the cluster level, rather than pairwise, appears to add value over TC primarily by increasing model score for families that would be false negatives under TC because they are tied together by outlier edges. In Figure 3, the benefit of the GNN score appears to increase for larger families, where there is more opportunity to have an outlier edge. We suspect that in real-world applications of inconsistent family detection to product graphs, where there may be hundreds or even thousands of products linked together, the benefits of this technique will grow.

Despite the performance enhancements achieved by the MAG-GCN, further work including raw node features such as images and text is needed to take full advantage of the graph neural network paradigm for product relationships. For the use case of automatically cleaving high confidence families, further work is recommended on using GNNs to learn the correct partitioning within the IC. Models whose loss depends on the categorization of each node such as the Line Graph Neural Network [5] might be well suited for this task.

6 Conclusions

Using graph neural networks that leverage graph structure, we are able to significantly improve the ability of cluster scoring models to detect Inconsistent Clusters for e-commerce applications. More generally, we show that graph neural networks can learn robust graph representations of weighted graphs for partitioning decisions. Future work will focus on incorporating additional node and edge features, as well as studying graph neural network structures that partition the graph explicitly rather than classify the entire graph as either consistent or inconsistent. For the binary classification problem, we hypothesize that methods allowing learnable coarsening of the graph [3, 29] could be useful. Graph neural networks offer an interesting path forward for other entity resolution problems because node features can be included along with the edge weights studied here.

In terms of incremental entity resolution, focusing on the IC problem allows us to handle the important case where we want to enforce that cluster quality does not degrade with ER updates. Having an Inconsistent Cluster detection mechanism is an efficient way to accomplish this regardless of the quality of pairwise scores, since precision and recall for cluster-level updates can be tuned along the curves shown in Figure 2. In this paper, we mainly studied the IC detection problem in the context of existing clusters prior to splitting, but the same cluster quality checks can be applied to clusters prior to being merged or split. Future work will include evaluating how to combine merging and splitting to optimize the IER in the general case.

References

[1] Christopher Aicher, Abigail Z Jacobs, and Aaron Clauset. Learning latent block structure in weighted networks. *Journal of Complex Networks*, 3, 2 221–248, 2015.

[2] Indrajit Bhattacharya and Lise Getoor. 2007. Collective entity resolution in relational data. ACM Transactions on Knowledge Discovery from Data (TKDD) 1, 1 (2007), 5–es.

[3] Cătălina Cangea, Petar Veličković, Nikola Jovanović, Thomas Kipf, and Pietro Liò. 2019. Towards sparse hierarchical graph classifiers. arXiv preprint arXiv:1811.01287 (2019).

[4] Yixin Cao, Lei Hou, Juanzi Li, and Zhiyuan Liu. 2018. Neural collective entity linking. arXiv preprint arXiv:1811.08603 (2018).

[5] Zhengdao Chen, Lisha Li, and Joan Bruna. 2019. Supervised Community De- tection with Line Graph Neural Networks. In 7th International Conference on Learning Representations (ICLR).

[6] William W Cohen and Jacob Richman. 2002. Learning to match and cluster large high-dimensional data sets for data integration. In Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining. 475–480.

[7] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. 2020. Principal Neighbourhood Aggregation for Graph Nets. Advances in Neural Information Processing Systems 33 (2020).

[8] J De Bruin. 2019. Python Record Linkage Toolkit: A toolkit for record linkage and duplicate detection in Python. https://doi.org/10.5281/zenodo.3559043

[9] Xin Luna Dong. 2018. Challenges and innovations in building a product knowledge graph. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2869–2869.

[10] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. In Advances in neural information processing systems. 2224–2232.

[11] Vijay Dwivedi, Chaitanya Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. 2020. Benchmarking Graph Neural Networks. arXiv preprint arXiv:2003.00982 (2020).

[12] Ivan P Fellegi and Alan B Sunter. 1969. A theory for record linkage. J. Amer. Statist. Assoc. 64, 328 (1969), 1183–1210.

[13] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In Proceedings of the 34th International Conference on Machine Learning-Volume 70. JMLR. org, 1263–1272.

[14] Marco Gori, Gabriele Monfardini, and Franco Scarselli. 2005. A new model for learning in graph domains. In Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005., Vol. 2. IEEE, 729–734.

[15] Anja Gruenheid, Xin Luna Dong, and Divesh Srivastava. 2014. Incremental record linkage. Proceedings of the VLDB Endowment 7, 9 (2014), 697–708.

[16] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In Advances in neural information processing systems. 1024–1034.

[17] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. arXiv preprint arXiv:1709.05584 (2017).

[18] Richard Hammack, Wilfried Imrich, and Sandi Klavžar. 2011. Handbook of product graphs. CRC press.

[19] Oktie Hassanzadeh, Fei Chiang, Hyun Chul Lee, and Renée J Miller. 2009. Framework for evaluating clustering algorithms in duplicate detection. Proceedings of the VLDB Endowment 2, 1 (2009), 1282–1293.

[20] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016).

[21] Pradap Konda, Sanjib Das, Paul Suganthan GC, AnHai Doan, Adel Ardalan, Jeffrey R Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeff Naughton, et al. 2016. Magellan: Toward building entity matching management systems. Proceedings of the VLDB Endowment 9, 12 (2016), 1197–1208.

[22] Juan Li, Zhicheng Dou, Yutao Zhu, Xiaochen Zuo, and Ji-Rong Wen. 2020. Deep cross-platform product matching in e-commerce. Information Retrieval Journal 23, 2 (2020), 136–158.

[23] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park,Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep learning for entity matching: A design space exploration. In Proceedings of the 2018 International Conference on Management of Data. 19–34.

[24] Markus Nentwig and Erhard Rahm. 2018. Incremental clustering on linked data. In 2018 IEEE International Conference on Data Mining Workshops (ICDMW). IEEE, 531–538.

[25] Anna Primpeli, Ralph Peeters, and Christian Bizer. 2019. The WDC training dataset and gold standard for large-scale product matching. In Companion Proceedings of The 2019 World Wide Web Conference. 381–386.

[26] Russel Reas, Stephen Ash, Rob Barton, and Andrew Borthwick. 2018. SuperPart: Supervised graph partitioning for record linkage. In International Conference on Data Mining. ICDM.

[27] Petar Ristoski, Petar Petrovski, Peter Mika, and Heiko Paulheim. 2018. A machine learning approach for product matching and categorization. Semantic web 9, 5 (2018), 707–728.

[28] Kashif Shah, Selcuk Kopru, and Jean David Ruvini. 2018. Neural network based extreme classification and similarity models for product matching. In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 3 (Industry Papers). 8–15.

[29] Pawel Swietojanski and Steve Renals. 2015. Differentiable pooling for unsu- pervised speaker adaptation. In 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 4305–4309.

[30] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. arXiv preprint arXiv:1710.10903 (2017).

[31] Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, Ziyue Huang, Qipeng Guo, Hao Zhang, Haibin Lin, Junbo Zhao, Jinyang Li, Alexander J Smola, and Zheng Zhang. 2019. Deep Graph Library: Towards Efficient and Scalable Deep Learning on Graphs. ICLR Workshop on Representation Learning on Graphs and Manifolds (2019). https: //arxiv.org/abs/1909.01315

[32] Qing Wang, Jingyi Gao, and Peter Christen. 2016. A clustering-based frame- work for incrementally repairing entity resolution. In Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, 283–295.

[33] Steven Euijong Whang and Hector Garcia-Molina. 2010. Entity resolution with evolving rules. PVLDB (2010).

[34] Steven Euijong Whang and Hector Garcia-Molina. 2014. Incremental entity resolution on rules and data. The VLDB journal 23, 1 (2014), 77–102.

[35] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? arXiv preprint arXiv:1810.00826 (2018).

Optimizing Email Marketing Campaigns in the Airline Industry using Knowledge Graph Embeddings

Amine Dadoun
EURECOM, Sophia Antipolis, France
Amadeus SAS, Biot, France
amine.dadoun@eurecom.fr

Raphaël Troncy
EURECOM, Sophia Antipolis, France
raphael.troncy@eurecom.fr

Michael Defoin Platel
Amadeus SAS, Biot, France
michael.defoinplatel@amadeus.com

Riccardo Petitti
Amadeus SAS, Biot, France
riccardo.petitti@amadeus.com

Gerardo Ayala Solano
Amadeus SAS, Biot, France
gerardo.ayalasolano@amadeus.com

Abstract

The use of email marketing campaigns in e-commerce is an essential aspect for driving sales and establishing or maintaining good customer service. However, ensuring their success in the airline industry is challenging, and the risk is high that the marketing campaign content is not suited to the needs of specific travelers. The traveler may rapidly feel spammed and the offer content promoted via emails may even be no longer accessed. Consequently, the use of personalised recommender systems for email marketing campaigns becomes essential to build user loyalty and to increase offer conversion. Recently, the use of knowledge graphs has proven to be an effective way to recommend products. In this work, we propose an approach that leverages knowledge graph embeddings to better target the right audience in email marketing campaigns for airline products recommendation. We conduct extensive experiments to compare our approach with the currently in-production rule-based system used by airline marketers and a supervised machine learning model based on handcrafted features as another baseline. The results suggest that the use of knowledge graph embeddings is the most effective approach.

1 Introduction

With the digital transformation of retail commerce from physical sale points to virtual stores, recommender systems have shown their value in easing the search and purchase decision process of customers who are facing an ever increasing amount of products [1]. In order to embrace e-commerce techniques and boost their revenues, some airlines are using the so-called Amadeus Anytime Merchandizing (AAM) Notification System [1] which is an IT solution that allows airline marketers to effectively define, deploy, monitor and adjust airline email marketing

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

[1] https://amadeus.com/en/portfolio/airlines/anytime-merchandising

campaigns sent to the travelers in real-time. Customized notifications can be defined and sent to travelers, after booking a flight, to suggest them additional services to buy (e.g. extra baggage, specific meal, preferred seat) called ancillaries. The solution acts as a bridge between the travelers retailing touchpoints and the airline's service and delivery system. As shown on the left side of Figure 1, when using this solution, the airline marketer can choose the appropriate time **when** to send the notification (e.g. 5 days before departure), **what** product to recommend (e.g. Leg Space Seat), **how** to send the offer (e.g. via an Email) and to **whom** this offer should be sent (matching targeting criteria).

Despite all the functionalities included in the AAM Notification System, it is difficult for an airline to find the optimal audience to target for a given offer. We conducted an analysis of historical sales triggered by some notification campaigns during the period 14 May 2019 - 17 Dec 2019 ran by one of our partner airlines and we observed a poor conversion of the notification offers (Section 3.3). This is partly due to the challenging decision-making process that an airline marketer faces when it comes to deciding which values (belonging to broad ranges) are appropriate for the criteria to be used (e.g. sending time, flight itineraries, flight departure point, etc.). Targeting customers with unsolicited notifications can be counter-productive and lead to adversarial effects on customer loyalty if done incorrectly. It is therefore critical to identify the customers that we expect to react positively to an advertised service in order to avoid spamming them with non-personalized emails. This problem can be seen as an inverse recommendation scenario, i.e. recommending a user to an item.

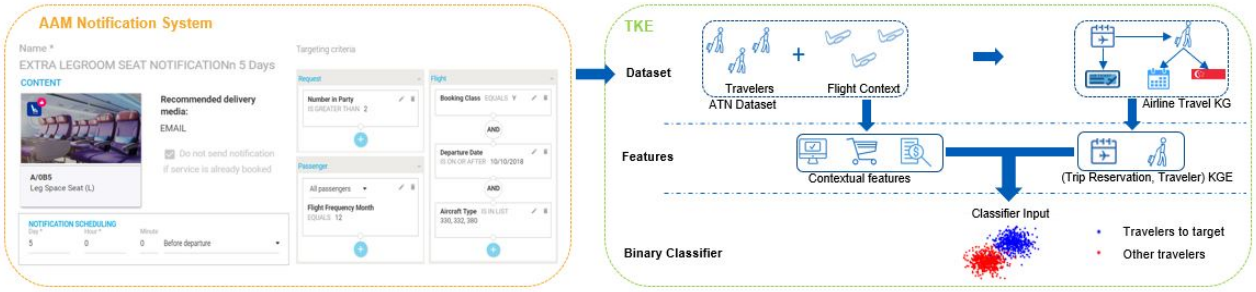


Figure 1: On the left side: AAM Notification System. On the right side: Flowchart of our proposed TKE framework. Notification dataset used in this study is generated from the AAM Notification system. Contextual features include booking context (e.g. number of passengers, date of departure, etc.), notification information (e.g. media used to send the notification, time of notification, etc.).

Inspired by recent works that have illustrated the effectiveness of using knowledge graph embeddings [2–4] and gradient boosting algorithms [5, 6] for item recommendation, we propose the **T**ravel **K**nowledge Graph **E**mbeddings for email marketing campaigns (TKE) framework to better target the audience for a service the airline wishes to recommend through email marketing campaigns (Figure 1). In this paper, we made the following contributions:

1. We design and develop a Knowledge Graph using Semantic Web technologies to represent the travelers past trips as well as to semantically enrich airline products.

2. We learn vector representations of travel entities via knowledge graph embedding algorithms and we leverage gradient boosting algorithms to compute prediction scores to better target the audience in email marketing campaigns.

3. We perform an empirical comparison of our approach with the current in-production rule-based system as well as with an hypothetical classical machine learning approach using handcrafted features on a real-world production dataset.

The paper is organized as follows. Section 2 provides a literature review of the related work. Section 3 introduces some preliminary concepts, the recommendation problem, the dataset and the knowledge graph used to conduct this work. In Section 4, we describe both our TKE approach as well as a machine learning model using handcrafted features as another baseline. In Section 5, we present the experiments we carried out and we demonstrate the effectiveness of our approach. Finally, in Section 6, we give some conclusions and we outline some future works.

2 Related Work

This section provides a literature review of the research conducted on email marketing campaigns in the e-commerce and tourism domains. We also present state-of-the-art methods of knowledge graph embedding algorithms and the use of gradient boosting algorithms for recommender systems.

2.1 Email Marketing Campaigns

Emails allow marketers to send messages to their customers at very low cost. They generally generate faster responses and create an opportunity for interactive communication with customers [7]. In [8], the authors analyze 70 randomized field experiments and find that email promotions not only increase customers' average purchase spending during the promotion window but also carry over to the week after the promotion expires. In our study, we will focus on personalized email marketing campaigns. In [9], the authors performed an empirical comparison of supervised machine learning models based on decision tree and logistic regression algorithms in order to improve the open rate and conversion rate of email marketing campaigns. In [10], the authors propose to use transactional features and instant messaging metadata to train a boosting tree regression algorithm to timely anticipate the needs of consumers in order to increase their level of engagement as well as the rate at which they repurchase products.

In the tourism domain, even if widely used, limited research is conducted on email marketing campaigns. In [11], the authors found that customers' favourite emails contain special offers, discounts and coupons as well as real-time communication tools. When customers perceived these emails as meeting their personal preferences, they developed a strong relationship with the sender. In [12], the authors demonstrated that the personalization, interactivity and price were important predictors of the possibility of revisiting the same accommodation. To the best of our knowledge, we are the first to propose a supervised machine learning approach to enable personalized email marketing in the airline travel industry.

2.2 Knowledge Graph Embeddings

A knowledge graph embedding (KG embedding) is a representation of a knowledge graph (KG) component into a continuous vector space. The objective of learning those embeddings is to ease the manipulation of graph components (entities, relations) for prediction tasks such as entity classification, link prediction or recommender systems.

In [13], the authors classified the learning algorithms into two main categories namely translational distance models and semantic matching models.

For the first category, TransE [14] is often mentioned as the most used translational distance model. TransE represents both entities and relations vectors in the same space R^d. Given a fact (h, r, t), the relation is interpreted as a translation vector r so that the embedded entities h (head) and t (tail) can be connected by r with low error, i.e., $h + r \approx t$ when (h, r, t) holds. TransH [15] introduces relation-specific hyperplanes, each relation r being represented on a hyperplane as w_r its normal vector. TransR [16] follows the same idea as TransH, but instead of projecting the relations into a hyperplane, it proposes to create a specific space per relation.

In the second category, semantic matching models exploit similarity-based scoring functions. In [17], the authors associate each entity with a vector to capture its latent semantics. Each relation is represented as a matrix that models pairwise interactions between latent factors. The score of a fact (h, r, t) is defined by a bi-linear scoring function minimized through tensor factorization based on ALS optimization technique.

Palumbo et al. [18] propose to use property-specific KG embeddings generated from node2vec algorithm [19] in order to compute relatedness scores between items and users. Similarly, we propose to use translational distance and semantic matching models to generate embeddings and to use them as latent features of a gradient boosting algorithm.

2.3 Gradient Boosting Algorithms

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weaker prediction models.

In [5], the authors used multiple additive Regression Trees (Dart) which is an ensemble model that uses boosted regression trees and handles the overspecialization. Their approach for online accommodation recommendation ranked 1^{st} in the famous RecSys 2019 Challenge[2]. In [6], the authors used XGBoost [20] which is an implementation of gradient boosting decision trees to predict tweet engagement, they have intensively used exploratory data analysis to extract and compute relevant features to feed XGBoost algorithm. Their solution ranked 1^{st} in the RecSys 2020 Challenge[3]. In our work, we use XGBoost algorithm as a supervised machine learning algorithm to better target the audience in email marketing campaigns based on the computed embeddings and contextual features (Figure 1).

3 Data & Problem Formulation

In this section, we first provide definitions of some useful concepts. Then, we define the problem and research questions addressed in this work. Finally, we present the dataset and the design of the knowledge graph used for the experiments.

3.1 Preliminaries

We define a Knowledge Graph (KG) similarly to what is done in [21].

Definition 1: A knowledge graph is a set $K = (E, R, O)$, where E is the set of entities, $R \subset E \times \Gamma \times E$ is a set of typed relations between entities and O is an ontology. The ontology O defines the set of relation types ('properties') Γ.

Definition 2: Travelers are a subset of the entities of the KG, $t \in T \subset E$.

Definition 3: Ancillaries are all products offered by the airline beyond air tickets. They can be flight-related (e.g. extra baggage, preferred seat, etc.) or standalone (e.g. lounge access) services. Ancillaries are advertised in the email marketing campaigns. Ancillaries are a subset of the entities of the KG, $a \in A \subset E$.

Definition 4: A notification campaign is a set of notifications sent to an audience of travelers within a given period of time and under some criteria to recommend an ancillary product.

[2] https://recsys.acm.org/recsys19/challenge/
[3] https://recsys.acm.org/recsys20/challenge/

Definition 5: We define the conversion rate of a notification campaign as follows:

$$CR = \frac{1}{N_o} \sum_{i=1}^{N_o} hit(N_i) \tag{14}$$

where N_o is the number of notifications sent through the notification campaign, and $hit(N_i)$ is equal to 1 if the notification N_i triggers a purchase. In our work, we focus on optimizing the conversion rate.

3.2 Problem Formulation

Given a notification campaign aimed at a large audience of travelers who have already booked a flight in a given context, we aim to target the relevant travelers among all the travelers that the notifications will reach. More specifically, we address the following research questions:

RQ 1: How to extract the relevant sample of travelers to target for a given notification campaign?.

RQ 2: How does a supervised machine learning approach perform compared to a rule-based approach to target the relevant audience for a notification campaign?

RQ 3: How does the use of KG embeddings compare to the use of handcrafted features as input of a supervised machine learning model trained to target the relevant audience for a notification campaign?

3.3 Notification Campaign Analysis

We analyzed three notification campaigns summing up to approximately 8.2 million notifications sent by one of our partner airlines to its travelers between 14 May 2019 and 17 December 2019 in order to understand the behavior of travelers in response to the notification campaigns, and to compute the conversion rates of these notification campaigns.

As shown in Table 4, there are three different types of ancillaries that are advertised in three notification campaigns. By analyzing airline sales data over the same period, we can see that only 3 out of 34 different types of purchased ancillaries were offered in the notification campaigns. This shows an untapped sales potential. Moreover, we observed that 50% of sales triggered by a notification happens on the same day (< 24 hours) after receiving the notification. This demonstrates the effect of a notification on the purchases.

Table 4: Conversion rates of notification campaigns: rule-based approach

Notification Campaign	Notification time	Date Range	Number of Notifications	Sales	CR
Extra leg room seat	5 days before Departure	19 May - 23 December 2019	∼355 K	∼2.8K	0.8%
Prepaid baggage	2 days before Departure	14 May - 17 December 2019	∼7.5 M	∼11K	0.15%
Lounge	Right after air ticket purchase	16 October - 17 December 2019	∼338 K	104	0.03%
All Notifications	-	-	-	∼13.8K	0.18%

While the prepaid baggage notification campaign was aimed at all travelers who booked a flight during the period indicated in Table 4, the notification campaigns for Lounge access and Extra leg room seat contain a number of filtering criteria that explain the large discrepancy in terms of the number of notifications sent out. Indeed, for these two notification campaigns, the airline marketer chose to send the notification to a quite restrictive audience by combining a number of criteria (fare family, aircraft type, no chargeable seat in their booking, etc.)

3.4 Airline Travel Notification Dataset

We conducted experiments on a real-world production dataset of bookings from the T-DNA database[4]. Each booking contains one or several air ticket purchases, and is stored using Passenger Name Record (PNR) information. The PNR is created at reservation time by airline reservation system and contains information about the purchased air ticket (e.g. travel itinerary, payment information), traveler demographics and ancillaries information if purchased (comprised in the EMD ticket). The considered dataset contains approximately 2.33 million bookings for approximately 2.85 million unique travelers.

The Airline Travel Notification (ATN) dataset is produced by joining the notification dataset and the historical bookings dataset from T-DNA. This dataset contains information about the shopping and booking context (e.g. search date, number of passenger, departure date, etc.) and information about travelers (e.g. demographics and loyalty membership information). In total, the dataset contains 42 columns and $\sim$ 8.2 million rows. For our experiments, the dataset was broken down into three different sub-datasets that correspond to the three different notification campaigns (Table 4).

3.5 Airline Travel Knowledge Graph

The Airline Travel KG is constructed from the T-DNA database. We develop an ontology which is available in the Turtle format at `https://gitlab.eurecom.fr/amadeus/tke4rec/-/blob/master/ontology/ams_ontology.ttl`. To design the KG, we have defined 7 classes corresponding to top level entities and based on the various tables available in the T-DNA database:

- **Traveler**: A traveler is identified by a T-DNA id. A traveler has a booking history (PNRs) that contain purchase history (air tickets, EMD tickets). An instance of traveler is a `schema:Person`[5].
- **Trip Reservation**: A trip reservation (PNR) represents the booking of all travelers contained in the PNR. It contains information such as the number of passengers, the destination, etc.
- **Journey**: A journey is linked to a trip reservation. Each journey has a stay duration, departure and arrival airports.
- **Air Ticket**: An air ticket is contained in a PNR and contains flight and transactional information. A PNR can have multiple air tickets because of the different flight legs (e.g. Nice-Paris, Paris-New York) or/and the number of passengers.
- **EMD Ticket**: An Electronic Miscellaneous Document (EMD) ticket is linked to an air ticket. It contains information on the ancillary purchased by the traveler (e.g. ancillary type, ancillary price, etc.).
- **Ancillary**: An ancillary is a service purchased by a traveler (associated to a flight) in addition to the air ticket. It is identified by a sub-code (RFISC), labelled by a commercial name, defined by ATPCO[6]. It belongs to a group of ancillaries (Group, RFIC). We propose to model the different ancillaries as SKOS concepts and we create an ancillary thesaurus represented as a concept scheme.
- **Airport**: It represents the airport where the traveler travels to. An airport serves one or several cities.

The KG used to tackle our use case contains 41 different properties (Figure 2), $\sim$ 80 million edges and $\sim$ 9 million nodes.

We present in Table 5 some statistics about the Airline Travel KG. In Figure 3, an excerpt of the KG is

[4]T-DNA: Traveler DNA is a database that contains bookings of travelers over a dozen of airlines

[5]The prefix `schema` is used for concepts defined by `https://schema.org`

[6]ATPCO Ancillary description: `https://www.atpco.net/resource/optional-services-industry-sub-codes`

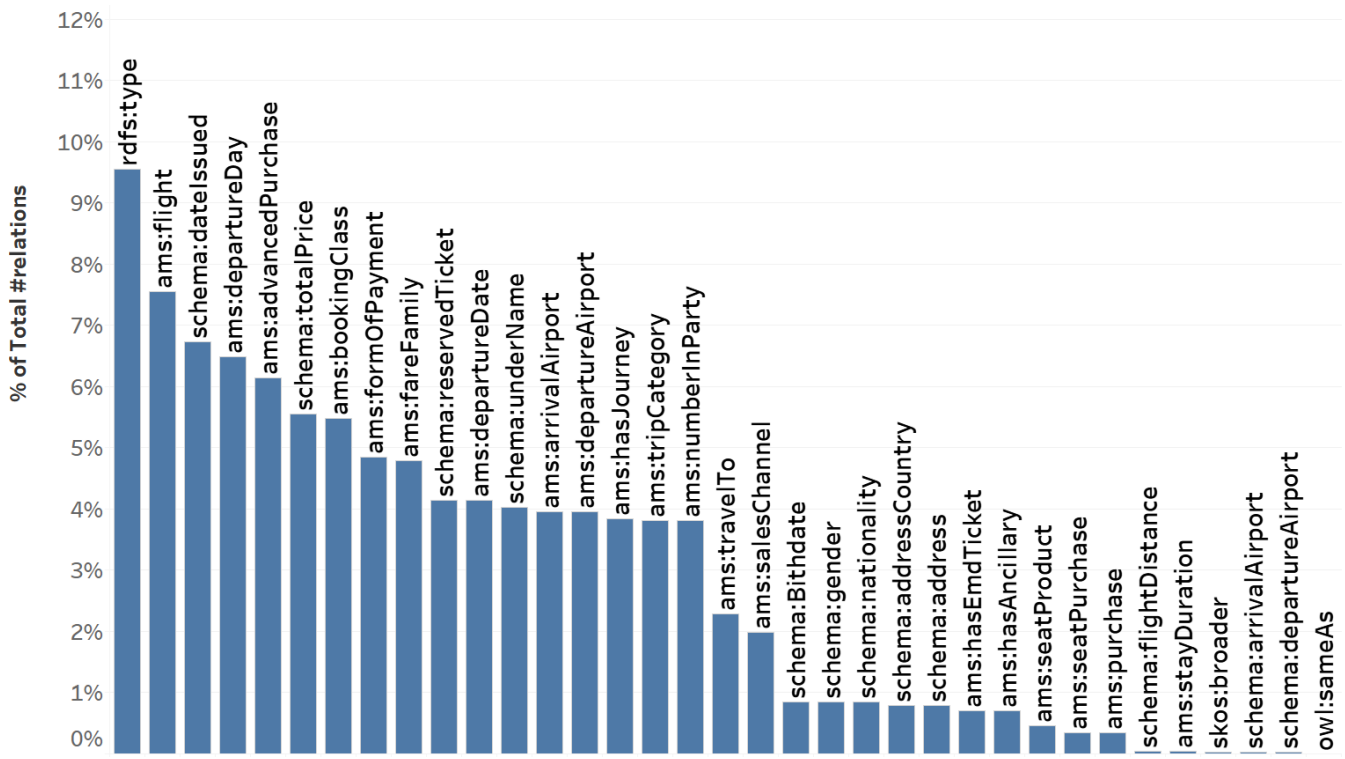


Figure 2: Distribution of #relations of properties in the Airline Travel KG. All prefixes can be found in the ontology definition.

Table 5: Statistics of subgraphs

Subgraph	#Edges	#Nodes	#travelers	#PNRs
Extra leg room seat	7M	800K	67K	205K
Prepaid baggage	64M	7.6M	572K	2.2M
Lounge	6.7M	789K	42K	203K

depicted, where a Malaysian traveler identified by T21354, born on "1988-05-05" has booked a one way flight for two people from Kuala lumpur to Melbourne. The EMD ticket identified by 23143 and linked to the air ticket 21563 represents the purchase of an ancillary (a seat).

4 Approach

Our proposed framework TKE can be seen as a two-stage approach as presented in Figure 1. In the first stage, we extract contextual features from the ATN dataset and compute KG embeddings of travelers and trip reservations from the Airline Travel KG. In the second stage, contextual features and KG embeddings are used as input of an XGBoost classifier in order to predict, for a given user, whether the notification should be sent or not.

We also propose a supervised machine learning model that includes as input contextual features and additional handcrafted travelers' features that capture travelers' preferences which we think could be particularly significant for model accuracy (hypothesis proven in the ablation study) as another baseline to compare with. We describe below the handcrafted travelers' features as well as the KG embeddings used in TKE:

- **Handcrafted travelers' features**: Features designed to capture travelers' preferences for ancillaries, destinations, points of sale, etc. and also customer lifetime value (Section 4.1).

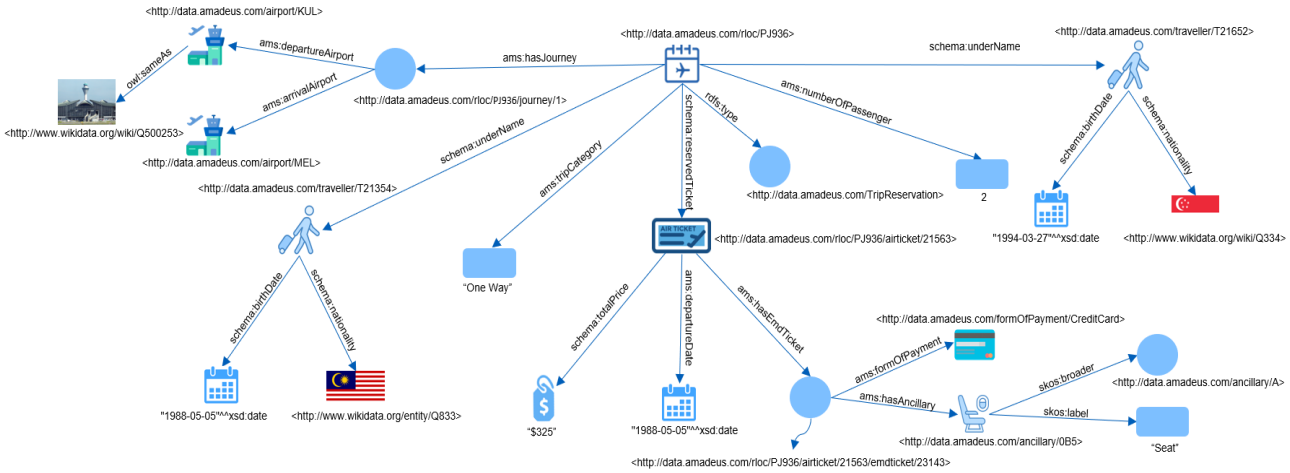


Figure 3: Excerpt of the knowledge graph representing the travelers included in a Trip reservation through the property `schema:underName`, as well as other properties and relations to other entities. Literals are represented in blue rectangle, whereas other entities are represented in blue circle. In this depiction, some properties that links travelers, trip reservations, air tickets and emd tickets are represented as an example, but more properties are included in the graph.

- **Knowledge graph embeddings**: Latent features representation of travelers and trip reservations computed based on KG embedding algorithms such as TransE [14] (Section 4.2).

4.1 Handcrafted Travelers Features

In this section, we present the handcrafted travelers features designed to compare their use with that of KG embeddings. We compute several features based on travelers purchase history, such as preferred ancillary, preferred destination, etc. We list below the features computed that leverage travelers' history:

- *Bookings count*: Number of bookings already purchased by the traveler with the airline.
- *Average flight revenue*: The average booking price tickets for all historical bookings of the traveler.
- *Preferred ancillary*: This feature corresponds to the most sold ancillary to the traveler.
- *Preferred destination*: This feature corresponds to the most visited destination (airport) by the traveler.
- *Preferred seat characteristic*: This feature represents the seat characteristic that is the most purchased by the traveler. There are three types of seat characteristic namely Upper deck, Exit Row, Leg Space.
- *Extra leg room seat already purchase*: For each seat characteristic, we create a binary feature that represents if a traveler has already purchased an Extra leg room seat or not.
- *Seat sales count*: This feature represents the number of times a seat has been purchased by the traveler.
- *Prepaid baggage sales count*: This feature represents the number of times a prepaid baggage has been purchased by the traveler.
- *Lounge sales count*: This feature represents the number of times a lounge access has been purchased by the traveler.

- *Notification response rate*: This feature is equal to the number of sales divided by the number of notifications sent to the traveler (regardless of the recommended service).

The handcrafted features and the features available in the ATN dataset are used as input of a gradient boosting decision tree classifier. We use the official implementation[7] of XGBoost [20] to train a binary classifier to predict if the notified travelers will convert or not. In Section 5, we give more details about the hyper-parameters used in XGBoost.

4.2 TKE Features

In this section, we explain how the KG embeddings used in TKE are computed. We use translational distance models to compute travelers and trip reservations embeddings as shown in Figure 1. More formally, we learn the KG embeddings based on a link prediction task, where some links of ancillary purchases and seat products are hidden in the training set, and put in the test set. Translational distance models are trained under the closed world assumption [13] using a pairwise loss that penalizes negative instances. More concretely, ancillaries that were not purchased by a traveler are considered as negative instances under the closed world assumption. Translational distance models are evaluated using ranking metrics such as hit rate or mean reciprocal rank. Hence, these models will return a high similarity score (low euclidean distance) for the ancillaries that are close in the graph embedding space to the embeddings of the ancillaries historically purchased by the travelers. As an example, we obtain a hit rate of ~ 0.42 with the TransE algorithm on the Airline Travel KG. In addition to translational distance models, we implemented a single-hidden multi-layer perceptron (MLP) as proposed in [22], where each relation (as well as entity) is associated with a single vector. More specifically, given a fact (h, r, t), the vector embeddings of h, r, and t are concatenated in the input layer, and mapped to a non-linear hidden layer. The score is then generated by a linear output layer. The generated embeddings are used as input of XGBoost classifier in addition to the contextual features as shown in Figure 1. We carry out a thorough empirical comparison of the aforementioned KG embedding algorithms and select the KG embeddings that allow the classifier to predict with the highest accuracy.

5 Experiments

The objective of the experiments is to compare the use of handcrafted features (a) with the use of KG embeddings (b). (a) helps in interpreting the results and predictions obtained by the algorithm, while (b) lacks interpretation (latent features), but is easier to compute and maintain. We publish our code as open source in order to ease reproducibility[8].

5.1 Experimental Setup

In this section, we present the different settings and the evaluation protocol (evaluation metrics and split of the dataset) used to conduct the experiments.

Dataset: We experiment both approaches (a) and (b) with the three datasets presented in Table 4. We use the Airline Travel KG presented in Section 3.5 to generate the KG embeddings useful for our main approach TKE.

Training & Test Sets: The three datasets corresponding to the three notification campaigns are split using the same strategy. Each dataset is sorted temporally, and 80% of the first rows of each dataset are used as training/validation sets. We use a cross-fold validation to train and validate all models (k=5, a split of 80% for training and 20% for validation). The remaining 20% are used as test set to evaluate the model. The split between training and validation set is performed randomly in order to avoid a seasonality effect that is usually occurring in

[7] XGBoost:https://XGBoost.readthedocs.io/

[8] https://gitlab.eurecom.fr/amadeus/tke4rec

Table 6: Evaluation results of the different approaches. (a) represents the results of XGBoost for different inputs; (b) represents the results of the TKE approach for different KG embedding algorithms. The average standard deviation (by varying the seed when splitting the dataset) of each metric is as follows: $AUC-ROC : \pm 0.02$, $TPR : \pm 3\%, TNR : \pm 2\%, CR : \pm 0.1\%$

Model	Extra leg room seat				Prepaid baggage				Lounge			
	AUC-ROC	TPR	TNR	CR	AUC-ROC	TPR	TNR	CR	AUC-ROC	TPR	TNR	CR
Rule-based	-	-	-	0.8%	-	-	-	0.15%	-	-	-	0.03%
(a) ADS	0.75	78%	58%	2.2%	0.83	80%	71%	0.38%	0.76	80%	62%	0.18%
(a) HDS	0.79	81%	60%	2.37%	0.85	82%	74%	0.4%	0.84	86%	67%	0.22%
(a) AHDS	0.83	85%	65%	2.8%	0.88	86%	**74%**	0.56%	0.89	88%	65%	0.36%
(b) TransE	0.85	86%	69%	3.1%	0.91	92%	65%	0.6%	0.90	89%	78%	0.35%
(b) TransH	0.84	85%	67%	3%	0.90	91%	65%	0.59%	**0.95**	**96%**	**85%**	**0.59%**
(b) TransR	0.84	85%	67%	2.9%	0.90	91%	65%	0.6%	0.92	92%	80%	0.52%
(b) MLP	**0.87**	**88%**	**69%**	**3.2%**	**0.92**	**94%**	65%	**0.62%**	0.91	90%	81%	0.56%

the travel industry. KG embedding algorithms are often designed to solve a link prediction task. We consider it is appropriate to split the KG by removing some edges that are included in the set of properties that link travelers with ancillaries and consider them as test sets, in order to evaluate the quality of the computed embeddings.

Evaluation metrics: The output of our approach is the probability of purchasing the recommended ancillary a included in the notification N:

$$P(purchase = a|N) = P(purchase|Context, TE, RE) \tag{15}$$

where, TE and RE are the Traveler and Trip reservation embeddings.

To evaluate and compare, the different approaches implemented, we used the conversion rate defined at definition 5 and the three metrics defined as follows:

- **TPR**: The true positive rate is the percentage of correct positive predictions. It represents the ratio of travelers that the algorithm suggests to send the notification and effectively purchase the ancillary. TPR is defined as follows:
$$TPR = \frac{TP}{(TP + FN)} \tag{16}$$
- **TNR**: The true negative rate is the percentage of correct negative predictions. It represents the ratio of travelers that the algorithm suggest to not send the notification and effectively do not purchase the ancillary. TNR is defined as follows:
$$TNR = \frac{TN}{(TN + FP)} \tag{17}$$
- **ROC-AUC**: The area under ROC curve (FPR, TPR) helps to choose what is the optimal probability threshold that maximizes the CR and TPR and is defined as follows:
$$ROC\text{-}AUC = \int_0^1 TPR \, d(FPR) \tag{18}$$
where, $FPR = 1 - TPR$ is the false positive rate

It is noteworthy that the conversion rate was measured offline as well as all the metrics based on the test set. According to equation 14, N_o represents the number of predicted positives and each hit hit_i corresponds to a true positive prediction.

Implementation Framework & Parameter Settings: For KG embedding algorithms, we use the deep learning framework pytorch[9] to implement MLP [22] and the library pykg2vec [23] for all the other KG embedding algorithms. The hyper-parameters of all the models were tuned using a combination of random-search and grid-search algorithms. We apply grid-search algorithm on the implemented algorithms using the following values: the embedding size k $\in \{32, 64, , 96, 128, 256\}$, the batch size $\in \{128, 256, 512, 1024\}$, the number of epochs $\in \{50, 100, 200\}$, the learning rate lr $\in \{0.001,$
$0.003, 0.01, 0.03, 0.1, 0.3\}$ and negative samples Ns $\in [2, 10]$ for MLP algorithm. We also optimize the following hyper-parameters of XGBoost classifier: the max depth of a tree $\in [5, 50]$, the number of trees $\in [10, 100]$, the sub-sample of each tree $\in [0.65, 0.85]$ and the col-sample of each tree $\in [0.65, 0.85]$. In addition to these hyper-parameters, we compute a weighted score (ratio of number of negative class to the positive class) that we use in XGBoost to approach the problem as a cost-sensitive learning problem due to the high class imbalance between positive (purchase) and negative (no purchase) classes (Table 4).

5.2 Results and Discussion

In this section, we discuss the results obtained from the experiments. Results of the experiments conducted are presented in table 6. TPR, TNR and ROC-AUC metrics are not provided for the rule-based approach implemented in AAM Notification System. The reason behind this is that the dataset used in the experiments is generated by the AAM notification system, which is different from the original dataset that contains all travelers used for the rule-based approach to identify the travelers matching the targeting criteria.

Ablation Study: Table 6 shows that using the features from the ATN dataset in addition to the travelers handcrafted features (AHDS: ATN + Handcrafted features) as input of XGBoost performs better than using only one of them (ADS: ATN features or HDS: Handcrafted features) as input for all notification campaigns. We also observe that using only travelers handcrafted features as input information of XGBoost gives better results than using the entire ATN dataset for all notification campaigns. We compute the most important features of the model (a) AHDS for each notification campaign and we report below the three most important ones with their respective information gain:

- Extra Leg Room Seat: {*Preferred Seat Characteristic*: 0.31, *Preferred ancillary*: 0.12, *Ticket amount*: 0.08}.
- Prepaid Baggage: {*Preferred destination*: 0.21, *Destination*: 0.12, *Prepaid Baggage sales Frequency*: 0.10}.
- Lounge: {*Average Flight Revenue*: 0.22, *Destination*: 0.20, *Age*: 0.15}.

Knowledge graph embeddings: We observe in Table 6 that using KG embeddings (concatenation of traveler and reservation KG embeddings) with contextual features as input of XGBoost performs better than using travelers handcrafted features regardless of the algorithm used to compute the embeddings or the notification campaign. Moreover, KG embeddings computed from **MLP** shows to perform better than KG embeddings computed from translational distance models except for the lounge notification campaign, where the use of KG embeddings computed from **TransH** model gives the best results.

6 Conclusions and Future Work

In this work, we have presented a two stage approach to address the problem of audience targeting for email marketing campaigns: first, we compute KG embeddings of travelers and reservations; second, we use these

[9] `Pytorch:https://pytorch.org/`

embeddings in addition to contextual features as input of a XGBoost classifier to learn what is the relevant audience to target for a given notification campaign. We conducted several experiments to address our research questions:

RQ 1: The results of the experiments presented in Table 6 show that extracting the relevant audience for a given notification campaign is not an easy task. Indeed, despite the fact that the conversion rate increases significantly with our approach, it remains relatively small. However, thanks to our approach, notification campaigns are better targeted and we manage to avoid recommending an unsuitable service to at least 65% of passengers.

RQ 2: Experiments have shown that the handcrafted features based supervised machine learning approach (a) gives better results than the rule-based one. Indeed, in Table 6, we can observe that the conversion rate is multiplied by more than 3 for Extra Leg Room Seat, almost 4 for Prepaid Baggage, and 12 for Lounge. Hence, we prove the benefit of using supervised machine learning over a simpler rule-based approach while it is the currently adopted mechanism used by airline marketers. It should be noted that the list of possible criteria available in AAM Notification System (Figure 1) is the same as the list of features used in the supervised machine learning approach.

RQ 3: Experiments show that regardless of the KG embedding algorithm tested, the KG embedding approach is better than the handcrafted features approach. This is very interesting from a scientific point of view, as it shows the added value of having a KG in the travel domain that could be used not only for ancillary recommendation task, but also other recommendation tasks (e.g. Trip recommendation) as the same KG embeddings could be used. It is worth noticing that when dealing with a cold-start problem (new user or item) for on-line usability, a rule-based approach is more appropriate.

Finally, as future work, we expect to tackle the task of personalized ancillary ranking in email marketing campaigns. More specifically, the goal of our future work will be to answer what is the most appropriate service to recommend in a notification campaign. In addition to addressing and optimizing what to recommend to a traveler, it would be interesting to optimize the time when to send the notification as it is an important decision making factor, especially in the airline travel industry [1].

References

[1] A. Dadoun, T. Fiig, M. Defoin-Platel, C. Landra, and R. Troncy, "How recommender systems can transform airline offer construction and retailing," *Journal of Revenue and Pricing Management, Special issue on AI*, 2021.

[2] E. Palumbo, G. Rizzo, R. Troncy, E. Baralis, M. Osella, and E. Ferro, "Translational models for item recommendation," in *The Semantic Web: ESWC 2018 Satellite Events*, A. Gangemi, A. L. Gentile, A. G. Nuzzolese, S. Rudolph, M. Maleshkova, H. Paulheim, J. Z. Pan, and M. Alam, Eds. Cham: Springer International Publishing, 2018, pp. 478–490.

[3] Z. Sun, J. Yang, J. Zhang, A. Bozzon, L.-K. Huang, and C. Xu, "Recurrent Knowledge Graph Embedding for Effective Recommendation," in *12th ACM Conference on Recommender Systems*. New York, NY, USA: ACM, 2018, pp. 297–305.

[4] E. Palumbo, D. Monti, G. Rizzo, R. Troncy, and E. Baralis, "entity2rec: Property-specific knowledge graph embeddings for item recommendation," *Expert Syst. Appl.*, vol. 151, p. 113235, 2020.

[5] P. Jankiewicz, L. Kyrashchuk, P. Sienkowski, and M. Wójcik, "Boosting algorithms for a session-based, context-aware recommender system in an online travel domain," in *Proceedings of the Workshop on ACM Recommender Systems Challenge*, ser. RecSys Challenge '19. New York, NY, USA: Association for Computing Machinery, 2019.

[6] B. Schifferer, G. Titericz, C. Deotte, C. Henkel, K. Onodera, J. Liu, B. Tunguz, E. Oldridge, G. De Souza Pereira Moreira, and A. Erdem, "Gpu accelerated feature engineering and training for recommender systems," in *Proceedings of the Recommender Systems Challenge 2020*, ser. RecSysChallenge '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 16–23. [Online]. Available: https://doi.org/10.1145/3415959.3415996

[7] L. Chittenden and R. Rettie, "An evaluation of e-mail marketing and factors affecting response," *Journal of Targeting, Measurement and Analysis for Marketing*, vol. 11, pp. 203–217, 2003.

[8] N. S. Sahni, D. Zou, and P. K. Chintagunta, "Do targeted discount offers serve as advertising? evidence from 70 field experiments," *Manage. Sci.*, vol. 63, no. 8, pp. 2688–2705, Aug. 2017. [Online]. Available: https://doi.org/10.1287/mnsc.2016.2450

[9] R. Abakouy, E. M. En-Naimi, and A. El Haddadi, "Classification and prediction based data mining algorithms to predict email marketing campaigns," in *Proceedings of the 2nd International Conference on Computing and Wireless Communication Systems*, ser. ICCWCS'17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3167486.3167520

[10] A. Deligiannis, C. Argyriou, and D. Kourtesis, "Predicting the optimal date and time to send personalized marketing messages to repeat buyers," *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 4, 2020. [Online]. Available: http://dx.doi.org/10.14569/IJACSA.2020.0110413

[11] K. Y. Tam and S. Y. Ho, "Web personalization as a persuasion strategy: An elaboration likelihood model perspective," *Info. Sys. Research*, vol. 16, no. 3, pp. 271–291, Sep. 2005. [Online]. Available: https://doi.org/10.1287/isre.1050.0058

[12] K. Yang, J. H. Min, and K. Garza-Baker, "Post-stay email marketing implications for the hotel industry: Role of email features, attitude, revisit intention and leisure involvement level," *Journal of Vacation Marketing*, vol. 25, no. 4, pp. 405–417, 2019.

[13] Q. Wang, Z. Mao, B. Wang, and L. Guo, "Knowledge graph embedding: A survey of approaches and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 12, pp. 2724–2743, Dec 2017.

[14] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Red Hook, NY, USA: Curran Associates, Inc., 2013, pp. 2787–2795. [Online]. Available: http://papers.nips.cc/paper/5071-translating-embeddings-for-modeling-multi-relational-data.pdf

[15] Z. Wang, J. Zhang, J. Feng, and Z. Chen, "Knowledge graph embedding by translating on hyperplanes," in *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, ser. AAAI'14. Québec City, Québec, Canada: AAAI Press, 2014, pp. 1112–1119.

[16] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, "Learning entity and relation embeddings for knowledge graph completion," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, ser. AAAI'15. Austin, Texas: AAAI Press, 2015, pp. 2181–2187.

[17] M. Nickel, V. Tresp, and H.-P. Kriegel, "A three-way model for collective learning on multi-relational data," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ser. ICML'11. Madison, WI, USA: Omnipress, 2011, pp. 809–816.

[18] E. Palumbo, G. Rizzo, and R. Troncy, "Entity2rec: Learning user-item relatedness from knowledge graphs for top-n item recommendation," in *Eleventh ACM Conference on Recommender Systems*. New York, NY, USA: ACM, 2017, pp. 32–36.

[19] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2016, pp. 855–864.

[20] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *22^{nd} ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 785–794.

[21] A. Dadoun, R. Troncy, O. Ratier, and R. Petitti, "Location Embeddings for Next Trip Recommendation," in *9^{th} International Workshop on Location on the Web (LocWeb)*. San Francisco, USA: Association for Computing Machinery, 2019, pp. 896–903.

[22] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang, "Knowledge vault: A web-scale approach to probabilistic knowledge fusion," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '14. New York, NY, USA: Association for Computing Machinery, 2014, pp. 601–610. [Online]. Available: https://doi.org/10.1145/2623330.2623623

[23] S.-Y. Yu, S. R. Chhetri, A. Canedo, P. Goyal, and M. A. A. Faruque, "Pykg2vec: A python library for knowledge graph embedding," *Journal of Machine Learning Research*, vol. 22, no. 16, pp. 1–6, 2021. [Online]. Available: http://jmlr.org/papers/v22/19-433.html

Interpretable Attribute-based Action-aware Bandits for Within-Session Personalization in E-commerce

Xu Liu[1], Congzhe Su[2], Amey Barapatre[2], Xiaoting Zhao[2], Diane Hu[2], Chu-Cheng Hsieh[2], Jingrui He[3]
[1] Arizona State University, [2] Etsy Inc., [3] University of Illinois at Urbana-Champaign
xliu338@asu.edu, {csu, abarapatre, xzhao, dhu, chsieh}@etsy.com, jingrui.he@gmail.com

Abstract

When shopping online, buyers often express and refine their purchase preferences by exploring different items in the product catalog based on varying attributes*, such as color, size, shape, and material. As such, it is increasingly important for e-commerce ranking systems to quickly learn a buyer's fine-grained preferences and re-rank items based on their most recent activity within the session. In this paper, we propose an Online Personalized Attribute-based Re-ranker (*OPAR*), a light-weight, within-session personalization approach using multi-arm bandits (MAB). As the buyer continues on their shopping mission and interacts with different products in an online shop,* OPAR *learns which attributes the buyer likes and dislikes, forming an interpretable user preference profile and improving re-ranking performance over time, within the same session. By representing each arm in the MAB as an attribute, we reduce the complexity space (compared with modeling preferences at the item level) while offering more fine-grained personalization (compared with modeling preferences at the product category level). We naturally extend this formulation to weight attributes differently in the reward function, depending on how the buyer interacts with the item (e.g. click, add-to-cart, purchase). We train and evaluate* OPAR *on a real-world e-commerce search ranking system and benchmark it against 4 state-of-the-art baselines on 8 datasets and show an improvement in ranking performance across all tasks.*

1 Introduction

When buyers shop online, they are often faced with thousands, if not millions, of products to explore and potentially purchase. In recent years, we've seen a growing interest in industrial applications of ranking systems as they help minimize distractions for the buyer and surface a digestible number of products that are most relevant to their shopping mission. These ranking systems take the form of search or recommendation systems, where products are ranked in descending order of relevance to the buyer [13, 17, 21, 31, 33, 37].

Just as a shopper might browse the aisles of a shop, online shoppers also spend time on a retailer's website searching and clicking on items before they decide what they want to buy. This process is an attempt to refine their purchase intent as they learn more about the product catalog. For example, a buyer might be interested in purchasing a ring; however, they often must click on a number of different rings before they understand possible styles, shapes, colors, and materials that are available. Eventually, the buyer might decide that they have a preference for an emerald gemstone, with a circular shape, and a gold band. Shifting to looking for a necklace,

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

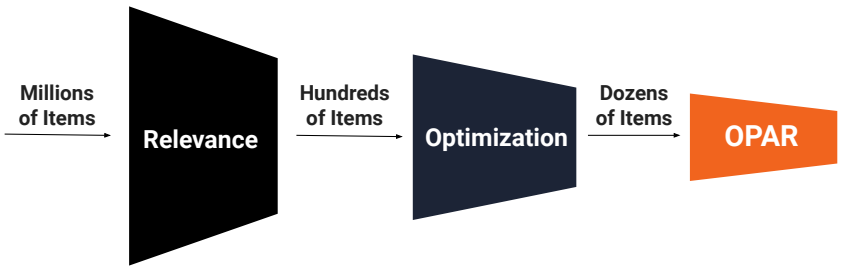


Figure 1: The first two components show a typical 2-stage ranker, where the first-pass narrows down the product catalog to relevant items, while the second-pass performs fine-grained re-ranking to optimize for a business metric. The proposed model, *OPAR*, is responsible for within-session, online personalization that can be effective on its own or as a third-pass ranker on top of a 2-stage ranking system.

the buyer must refine their preference again. Often the buyer's preference for attributes like colors and materials changes quickly over the course of one visit. An intelligent ranking system must continually serve content that stays relevant to the buyer's changing preference, a capability we refer to as *within-session personalization*.

Many production ranking systems today have multiple goals to balance: online retailers not only surface content that is *relevant* to the shopper's buying mission (for example, a search query for "wristwatc" must produce wrist watches), but they also aim show content that is likely to improve a business metric (eg. conversion rate, or GMV). In order to balance these goals, many production ranking systems leverage a 2-stage ranking process (Figure 1): the first pass (commonly referred to as *candidate set selection*) narrows hundreds of millions of items from the product catalog down to a few hundred relevant items [14, 21, 36]; the second pass then re-ranks the top few hundred relevant items in a way that optimizes for specific user action (such as a click or purchase) [9, 10, 22, 31]. In order to maximize prediction accuracy, these systems often train on billions of historical data points that may span over the course of months or years and thus cannot react quickly enough to the buyer's changing preference within a shopping visit.

In this paper, we propose an Online Personalized Attribute-based Re-ranker (*OPAR*) that can respond quickly to the changing preferences of a buyer within their immediate shopping session, while still reaping the benefits of a traditional 2-stage system. In the MAB literature, it is common to address this problem by treating each arm in the bandit to represent a single item [37], product category [28, 33] or a context [13, 15, 16]. In contrast, *OPAR* decomposes each product into a descriptive set of attributes (such as its color, texture, material, and shape), and represents each arm as an *attribute*. As the buyer interacts with different products in an online shop, the bandit learns which attributes the buyer likes and dislikes, forming an interpretable user preference profile that is used to re-rank products in real-time in a personalized manner. By representing each arm as an attribute, we reduce the complexity of the space, while allowing more fine-grained personalization within a product category. We naturally extend this formulation to weight attributes differently in the reward function, depending on how the user interacts with that item (e.g. attributes from a clicked item will be weighted less than attributes from an add-to-cart item).

In our example of searching for a ring, we see in Figure 2 that initially, the same 12 items are shown to two different users. While user 1 might click on items that contain the attributes *crystal, gemstone, ruby, rose gold* (outlined in *green*), user 2 might have different preferences and click on items that contain the attributes *diamond, engagement, oval cut, 14K gold* (outlined in blue). At this point, *OPAR* will begin to differentiate the diverging preferences of these two users based on the different attributes that each user has shown interest in. On a subsequent search page, *OPAR* will rank gemstone rings higher for user 1, while user 2 will see diamond rings at the top of the list. Furthermore, because the learned weights of each attribute can be observed for each user, our model is extremely interpretable.

While *OPAR* can be used as a stand-alone algorithm, we find it to be most effective when deployed as a

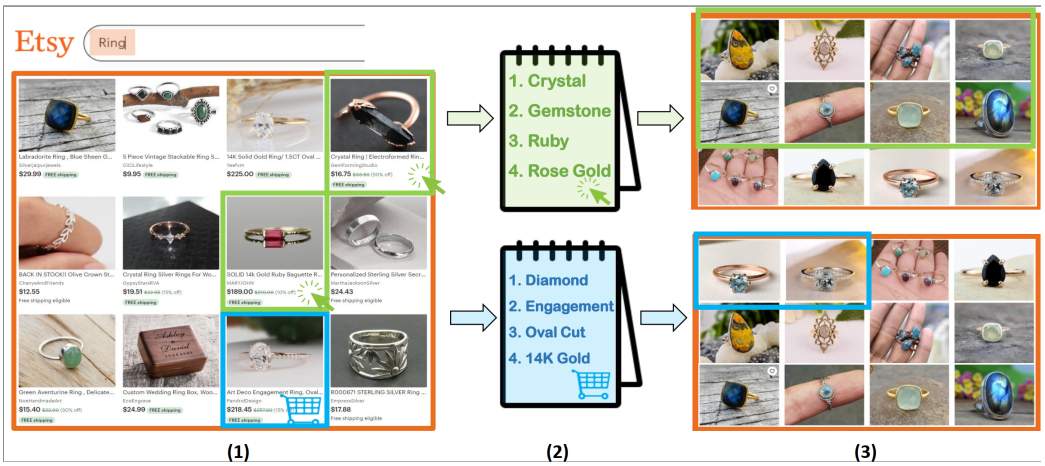


Figure 2: Example of attribute and action-aware re-ranking by *OPAR*. From left to right: (1) shows search results for the query "Ring". User 1 clicked on two gemstone rings (outlined in green), while User 2 adds a diamond ring to their cart (outlined in blue) (2) The attribute of the clicked items are "Crystal", "Gemstone", "Ruby" and "Rose Gold", while the add-to-cart item has the attributes "Diamond", "Engagement", "Oval-Cut" and "14k Gold" (3) On a subsequent search page, *OPAR* re-ranks items based on each user's diverging preferences.

third-pass ranker on top of a traditional two-stage ranking system (see Figure 1). This allows us to leverage the power of traditional 2-pass systems that learn from long-term data aggregated over billions of user and item preferences, while still being nimble enough to personalize a buyer's experience by taking into account their most recent activity.

In the following, we will introduce the proposed model, *OPAR*, and show how we apply it to a search ranking problem on a popular e-commerce platform. Our contributions are as follows:

- **Attribute-level personalization**: *OPAR* performs real-time personalized re-ranking based on user's preferences at the attribute level and reduces the space complexity while offering more fine-grained personalization.
- **Light-weight, online re-ranker**: *OPAR* improves ranking performance with little data and requires us to track a minimal number of variables as arms and can be added on top of the traditional 2-pass ranking systems.
- **Interpretable user preferences**: The learned attribute weights give visibility into attributes that the user likes and dislikes. Top-weighted ones can be used for down-stream personalization tasks.
- **Evaluation on real-world datasets**: *OPAR* is trained and evaluated on real-world e-commerce data and is compared to baselines on 8 datasets from a production e-commerce ranking system. We describe a session-level ranking metric to understand ranking improvements within a session.

2 Related Work

In this section, we summarize the related work from literature and categorize them into two aspects: (1) *Session-based Ranking System*, and (2) *Multi-armed Bandit Ranking System*.

2.1 Within-Session Ranking

The within-session ranking task tries to predict what action the user will take next within the current shopping session, leveraging the temporal nature of their browsing behavior from within the same session [17, 34]. Significant breakthroughs in deep learning (i.e, batch normalization and dropout), have led to its wide adoptions in various communities and applications [35]. In [11], recurrent neural networks (RNNs) were proposed for this within-session ranking task and gained significant attraction given its superior predictive performance for the next-item recommendation. This has been an active research area with various enhancements proposed specifically for predicting short-term user behavior within the same shopping session [11, 12, 24, 27, 34].

Given that a long-term memory models are insufficient to address drift in user interests, [18] proposed a short-term attention priority model to capture users' general (long-term) interest in addition to the users' within-session interest via a short-term memory model based on the recent clicks. In parallel, [17] studied the behavior-intensive neural network for personalized next-item recommendation by considering both users' long-term preference as well as within-session purchase intent. As RNNs have shown and emerged as the powerful technique to model sequential data for this task, [20] argued for an alternative model, inspired by machine translation, by proposing an encoder-decoder neural architecture with an attention mechanism added to capture user session intents and inter session dependencies. In addition to sequential models, [23] leverages graph neural networks by constructing a session graph and then modeling a weighted attention layer when predicting user's preference in session. To tackle uncertainty that arises in a user's within-session behavior, authors in [8] proposed a Matrix Factorization-based attention model to address large-volume and high-velocity session streaming data and [19] handles the missing value issue for the matrix factorization.

Most previous work cited above do not aim for interpretability of its results. In contrast, the model we propose specifically leverages item attributes from the product catalog, resulting in a simple algorithm that learns interpretable user profiles that aid in within-session personalization. The closest related work is [4] that proposes the attribute-aware neural attentive model for the next shopping basket recommendation but does not seem to easily adapt for the real-time scenario due to its complexity.

2.2 Multi-Armed Bandits Ranking System

Requiring a responsive and scalable ranking system that can adapt to the dynamic nature of shifting user preferences (especially in the cold start setting) has led to increasingly wider industry adoption of multi-armed bandit (MAB) in modern day ranking systems. The theoretical foundation and analysis of MABs have been well-studied, with popular approaches include ϵ-greedy [26], Upper Confidence Bounds [2], Thompson sampling [7], EXP3 [3], and others [26]. In the e-commerce [37] setting, the goal is to maximize user satisfaction (i.e., exploitation), while quickly learning (i.e., exploration) users preferences by exploring unseen content.

Hu et al. in [13] proposed to use reinforcement learning to learn an optimal ranking policy that maximizes the expected accumulative rewards in a search session. Yan et al. from [33] built a scalable deep online ranking system (DORS) with MABs as the last pass to dynamically re-rank items based on user real-time feedback and showed significant improvement in both users satisfaction and platform revenue. Furthermore, authors from [25] proposed a multi-armed nearest-neighbor bandit to achieve collaborative filtering for the interactive recommendation, by modeling users as arms and exploring the users' neighborhood. [29] proposed an interactive collaborative topic regression model that infers the clusters of arms via topic models [5] and then utilizes dependent arms for the recommendation.

In this literature, it is common to address this problem by treating each arm in the bandit to represent a single item [37], product category [33] or a context [13, 15, 16]. In contrast, *OPAR* decomposes each product into its descriptive set of attributes (such as its color, texture, material, and shape), represents each arm as an *attribute* and provides great explainability in addition to its performance.

(a) (b)

(c) (d)

Figure 3: Top attribute-value pairs for top categories: (a) Jewelry; (b) Clothing; (c) Craft Supplies and Tools, (d) Home and Living.

3 Problem Formulation

In this section, we provide definitions for commonly used terms such as sessions and attributes. We then explain our model in two parts: (1) how to represent within-session attribute preferences, and (2) how to re-rank items based on these preferences.

3.1 Definitions

Definition 1: A *session* is a sequence of actions that the buyer takes while engaging with an e-commerce platform in trying to fulfill a shopping mission (e.g. search, click, add-to-cart). The session typically ends when the buyer leaves the site with a purchase or abandons after a significant duration of inactivity (e.g., 30 minutes). Note that we focus on product search here but should be generally applicable to other ranking or recommendation problems.

Let us define a session $S = \{[Q_t, I_t, A_t]\}_{t=1}^{T}$ that is a sequence of T user actions within a session, in which T can vary across sessions. The session starts at $t = 1$ and ends at T with a purchase (or becomes inactive). At each time step, item list $I_t \in R^{M\times 1}$ contains M candidate items to be re-ranked for query Q_t, and then how the

user engages with the list of items is represented by A_t:

$$A_t(x_i) = \begin{cases} 0, \text{ no action on } x_i \\ 1, \ x_i \text{ is purchased} \\ 2, \ x_i \text{ is added to cart} \\ 3, \ x_i \text{ is clicked} \end{cases}, \forall x_i \in I_t. \tag{19}$$

Definition 2: An *attribute* is a basic unit (e.g. size, color) that describes the product characteristics of an item. The attributes are determined by taxonomists based on the product category while the value of the attributes (e.g. large, green) are volunteered by the seller, or inferred by machine-learned classifiers. These attribute-value pairs help buyers efficiently navigate through an overwhelmingly large inventory. Thus, each item x_i is represented as the composition of its attributes, with H_{x_i} denoting the total number of attributes associated with $x_i : x_i = \{atr_1, atr_2, \ldots, atr_{H_{x_i}}\}$.

Figure 3 shows four category-specific word clouds of attributes-value pairs exhibited in items from top categories at Etsy, one of the largest e-commerce platform for handmade, vintage, and craft supplies. Some of the most common attributes are universal: size, color, and material. Others are category specific: sleeve length, earring location, and craft type. Lastly, some attributes (e.g. holiday, occasion, recipient) describe how or when the item can be used.

3.2 Problem Statement

Our goal is to (1) formulate each user's within-session preference for product attributes and (2) re-rank a list of candidate items based on the user's inferred within-session preference on item attributes.

Part 1: *How to formulate users' in-session attribute preferences?*

Input: For session S, (1) item lists $\{I_t\}_{t=1}^T$ with each item $x_i = \{atr_{H_{x_i}}\}$ as composition of product attributes; and (2) session-level record of user actions on shown items, $\{A_t\}_{t=1}^T$.

Output User's preference $\mathbf{\Theta}$ on attributes as beta-distributed: $\mathbf{\Theta} = \{\theta_{atr_n}\}_{n=1}^N \sim \{Beta(\alpha_{atr_n}, \beta_{atr_n})\}_{n=1}^N$, where N denotes the total number of attributes encountered in session S.

For a user, we model their within-session preference on an attribute as a latent value $\theta_{atr_n} \in [0, 1]$ denoting the probability that they would like the attribute exhibited in the item. Motivated by Thompson Sampling [1], let θ_{atr_n} be beta-distributed, with $\alpha_{atr_n}, \beta_{atr_n}$ be the two parameters of the distribution. In Section 5.2 we show a method on estimating the parameters of attributes from historical data. From the list of shown items I_t, the user engages on a subset of items (denoted in A_t) to express their preference for item attributes according to $\mathbf{\Theta}$. Given the feedback, we propagate rewards from the user actions to the associated attributes with increments, $\delta_{A_t(x_i)}$, and update the posterior distribution of $\mathbf{\Theta}$, with rewards normalized at x_i by its cardinality (number of associated attributes on that item).

Part 2: *How to sequentially re-rank I_t based on user preference $\mathbf{\Theta}$* to optimize in-session personalization?

Input: At time t, (1) Candidate list of items I_t, and (2) user in-session preference $\mathbf{\Theta}$.

Output: Sequentially learn $f_t : I_t \times \mathbf{\Theta} \rightarrow \tilde{I}_t$.

Below we will present the *OPAR* algorithm to address the problem statement discussed here.

4 Proposed Algorithm, *OPAR*

In this section, we present the details of the proposed *OPAR* model and its extension *OPAR*$_w$ which differentiates different user actions.

Algorithm 2: OPAR Algo: Re-Ranking & Parameter Update

Input:

Given a session $S = \{[Q_t, I_t, A_t]\}_{t=1}^{T}$

$\{\delta_i\}_{i:\text{ actions}}$: action-aware increments on attribute parameters

γ: hyper-parameter to control intensity on negatives

$\mathcal{U}_t$, $\mathcal{V}_t$: the associated attributes from engaged items; the associated attributes from impressed items

$|\cdot|_0$: cardinality operator

for $[Q_t, I_t, A_t] \in S$ **do**

(1) Rerank on the Item List $f : I_t \to \tilde{I}_t$

sample $s_{atr_h} \sim Beta(\alpha_{atr_h}, \beta_{atr_h}), \forall atr_h \in N_S$

for $x_i \in I_t$ **do**

Given $x_i = \{atr_h\}_{h=1}^{H_{x_i}}$ as associated attributes in x_i, set $score(x_i) = \sum_{atr_h \in x_i} g(s_{atr_h})$

end

$\tilde{I}_t = sorted([score(x_i)]_{x_i \in I_t})$

(2) Update attribute parameters given A_t

Let $\mathcal{U}_t = \cup\{atr_h : \forall atr_h \in x_i \, if \, A_t(x_i) \neq 0, \, \forall x_i \in I_t\}$

Let $\mathcal{V}_t = \cup\{atr_h : \forall atr_h \in x_i \, \forall x_i \in I_t\}$

for $x_i \in I_t$ **do**

if $A_t(x_i) \neq 0$, *item* x_i *has positive actions* **then**

$\alpha_{atr_h} += \delta_{A_t(x_i)} \times \{1 - Exp\,(-|\mathcal{U}_t|_0)\}, \forall atr_h \in x_i$

else if $A_t(x_i) = 0$, *no action on item* x_i **then**

$\beta_{atr_h} += \delta_{A_t(x_i)} \times \{1 - Exp\,(-\gamma|\mathcal{V}_t \backslash \mathcal{U}_t|_0)\}, \forall atr_h \in x_i$

end

end

end

Output: All re-ranking results $[\tilde{I}_t]_{t=1}^{T}$

4.1 Scoring and Re-ranking Item List

Given attribute-level bandits with each arm as an item attribute, we describe below our approach on how we score and re-rank items, motivated by the Thompson Sampling approach on [1].

Let N denote the number of attributes associated with item list I_t. For each attribute in $\{atr_h : atr_h \in x_i, \, \forall x_i \in I_t\}$, we randomly sample θ_{atr_h} from its corresponding distribution, denoting the probability that the user is interested in the attribute, atr_h, at time t:

$$\theta_{atr_h} \sim Beta(\alpha_{atr_h}, \beta_{atr_h}). \tag{20}$$

Then, each item $x_i \in I_t$ is scored and ranked by:

$$score(x_i) = \sum_{atr_h \in x_i} g(\theta_{atr_h}), \tag{21}$$

where $g(\theta_{atr_h}) = \frac{1}{rank(\theta_{atr_h})}$ is a harnomic function of the index that θ_{atr_h} is ranked among $[\theta_{atr_h}]_{h=1}^{H_{x_i}}$, with a tie-breaker uniformly at random. A larger $score(x_i)$ indicates higher satisfication with item x_i given users' short in-session preference on the attributes. Lastly, we present the user $\tilde{I}_t$, which is reranked list of the items based on $[score(x_i)]_{x_i \in I_t}$.

4.2 Attribute Parameter Updates

With the feedback gathered from the user action A_t, we do the following updates for the attribute parameters. Let $\mathcal{U}_t$ denote the set of attributes associated from items with positive actions (i.e., click, add-to-cart, purchase), and $\mathcal{V}_t$ be union of all attributes exist in $x_i \in I_t$:

$$\mathcal{U}_t = \cup\{atr_h : \forall atr_h \in x_i \text{ if } A_t(x_i) \neq 0,\ \forall x_i \in I_t\},\ \mathcal{V}_t = \cup\{atr_h : \forall atr_h \in x_i,\ \forall x_i \in I_t\}.$$

For a given atr_h, let $\tilde{\mathcal{Y}}_{t,atr_h}$ and $\tilde{\mathcal{Z}}_{t,atr_h}$ denote the set of items associated with positive user action and no-action, respectively,

$$\tilde{\mathcal{Y}}_{t,atr_h} = \{x_i \in I_t : atr_h \in x_i \text{ and } atr_h \in \mathcal{U}_t\},\ \tilde{\mathcal{Z}}_{t,atr_h} = \{x_i \in I_t : atr_h \in x_i \text{ and } atr_h \in \mathcal{V}_t \backslash \mathcal{U}_t\}.$$

Then, the Beta distribution of each attribute is updated as follows:

$$\begin{aligned}
\alpha_{atr_h} += & \sum_{\tilde{\mathcal{Y}}_{t,atr_h}} \delta_{A_t(x_i)} \left(1 - e^{-|\mathcal{U}_t|_0}\right), \forall atr_h \in \mathcal{U}_t, \\
\beta_{atr_h} += & \sum_{\tilde{\mathcal{Z}}_{t,atr_h}} \delta_{A_t(x_i)} \left(1 - e^{-\gamma|\mathcal{V}_t \backslash \mathcal{U}_t|_0}\right), \forall atr_h \in \mathcal{V}_t \backslash \mathcal{U}_t,
\end{aligned} \tag{22}$$

where $|\cdot|_0$ denotes the cardinality operator and γ controls intensity on implicit no-actions.

4.3 *OPAR* algorithm Procedure

In summary, given a session $S = \{[Q_t, I_t, A_t]\}_{t=1}^{T}$, *OPAR* can be summarized with the following steps, with the pseudo code of *OPAR*$_w$ shown in Algorithm 2.

1. Initialize attribute dictionary $atrDic \in R^{N\times 2}$, which contains N pairs of parameters for attributes, where each row of $atrDic$ denotes the Beta distribution parameter set $(\alpha_{atr}, \beta_{atr})$ for a given attribute. Different initializations have been experimented, including uniform, random or estimated based on held-out historical datasets (shown in Section 5.2).

2. At time t, we score each item $x_i \in I_t$ based on Eq. (21): it first aggregates over the associated attribute preferences sampled in Eq. (20), and then re-rank items based on scores in Eq. (21) and present as $\tilde{I}_t$. More details in Section 4.1.

3. At time t, we receive the obervation A_t on I_t, and then update the distribution of all attributes associated with item x_i in the *atrDic* based on the Eq. (22) described in Section 4.2.

   OPAR: attribute-based bandits with *equal* action-weighting for actions in {click, add-to-cart, purchase}. This means that for positive actions, $\delta_{\text{click}} = \delta_{\text{add-to-cart}} = \delta_{\text{purchase}}$.

   OPAR$_w$: extend *OPAR* to weight action-aware updates as follows, $\delta_{\text{click}} \neq \delta_{\text{add-to-cart}} \neq \delta_{\text{purchase}}$, and hypertune them.

4. We iterate step (2) and (3) until the end of the session.

5 Experiments

In this section, we show how *OPAR* performs on a real-world e-commerce ranking system and benchmark it against 4 baselines on 8 datasets. While *OPAR* can be applied to any content that requires re-ranking, we

Table 7: Etsy Real-world Session-based Dataset Over 3 weeks

ID	Category	|Session (User)|	|Query|	|Item|	|Attributes|	|Actions|
1	**Clothing**	4642	46091	1100040	2495	58932
2	**Home & Living**	9073	103959	2282542	2455	134416
3	**Paper & Party Supplies**	4419	35132	691919	1666	55037
4	**Craft Supplies & Tools**	10913	123662	2536492	2799	171363
5	**Accessories**	5813	38215	897533	2419	49342
6	**Electronics & Accessories**	1638	10505	216860	1302	14354
7	**Jewelry**	5585	67507	1530285	2266	79874
8	**Overall Category**	26442	474594	9295453	3363	624882

specifically chose to train, evaluate, and analyze the model performance on a search ranking system, as the explicit search queries issued by a user shows higher purchase intent, allowing us to better evaluate *OPAR*'s ranking and interpretation capabilities. Our experimentation seeks to answer the following questions:
Experiment #1: What is the ranking performance of the proposed *OPAR* model? *(Answered in subsection 5.4.1)*
Experiment #2: How does *OPAR* perform as an action-aware model? *(Answered in subsection 5.4.2)*
Experiment #3: How does *OPAR* help to understand users' short-term, in-session shopping preference? *(Answered in subsection 5.4.3)*

5.1 Data Collection

The dataset is collected and sampled from a month of user search logs at Etsy, one of the largest e-commerce platforms for handmade, vintage items, and craft supplies. To avoid bot traffic, filters are added to include sessions with at least 10 search events (i.e., queries, browses, clicks, add-to-carts) and at least one *purchase* to focus on sessions with strong shopping missions. Using an existing query classifier, we predict the most probable category (e.g. jewelry, home and living) associated with the first query of each session, and then bucket the entire session into one of 7 categories. This helps us understand shopping behaviors within each category. Table 7 shows statistics of each data set, representing nearly 500k search queries from 26k sessions and 620k user actions combined on nearly ten million items, with cardinalities computed within each dataset. We do not perform the evaluation on existing public datasets, because (to the best of our best knowledge) there is no existing dataset that includes all meta-data needed for our study (e.g. query, item attribute, user interaction logs).

5.2 Experimental Set-up

We split each of the 8 datasets into 2 parts (with sessions ordered chronologically). The first two-thirds of the data is a **held-out dataset**. Because we are focused on online learning, using only within-session data, the held-out dataset is mainly used for estimating the parameters of the Beta distributions, $\{(\alpha_{atr}, \beta_{atr})\}_{\forall atr}$, and to aggregate attribute counts associated with engaged items to determine attribute popularity, powering the "Atr-POP" algorithm in Section 5.3. The remaining data is the **testing dataset**, on which we report re-ranking performance for *OPAR* and other baseline algorithms on in Table 8.

While *OPAR* can function as a stand-alone ranking algorithm, we evaluate OPAR (as well as other baselines) on top of an existing 2-pass ranking system (as described in Figure 1). More formally, each session in the testing dataset, $S = \{[Q_t, I_t, A_t]\}_{t=1}^{T}$ contains a sequential list of query content Q_t, a candidate set I_t of items to be re-ranked, and logged user actions A_t on I_t (e.g. click, purchase). In our experiments, I_t is a truncated list of the top 48 items returned by an existing 2-pass ranker, indicating that this list comprises of the most

relevant items to the query. As we will see in experimental results, applying *OPAR* adds an effective layer of attribute-based personalization in real-time that was not feasible with the underlying system. In order to simulate an online environment, only within-session user interactions leading up to the current time step are used for ranking predictions.

5.3 Evaluation Metrics and Baselines

Below, we describe the offline metrics we use to evaluate *OPAR* on the testing dataset, as well as the baselines we benchmark.

5.3.1 Evaluation Metrics

Following the general ranking metric Normalized Discounted Cumulative Gain (NDCG) [30], we propose a set of session-level ranking metrics to evaluate our model.

1. *Click-NDCG*: For each query Q_t issued in S that has at least one click in A_t (i.e, clicks as relevances), *click-NDCG*$_t$ measures the re-ranking performance of the item list $\tilde{I}_t$ (after re-ranking I_t) shown to the user at t. For all timestamp with at least a click, we first compute stepwise sequential re-ranking performance *click-NDCG*$_t$ as:
$$click\text{-}NDCG_t = click\text{-}DCG_t / IDCG_t, \forall t = 1, ...T, \tag{23}$$
and *click-NDCG* of a session S is the average of *click-NDCG*$_t$ over events that have at least one click:
$$\textit{click-NDCG} = \text{Average}(\textit{click-NDCG}_t). \tag{24}$$

2. *Purchase-NDCG*: Following the above methodology, we compute the session-level re-ranking performance limit to search events with attributed purchases. A session on a shopping site is defined as a sequence of events ending with a purchase or a significant duration of inactivity. Given that, *Purchase-NDCG* given a session is essentially *purchase-NDCG*$_T$.

For each re-ranking algorithm reported in Table 8, we compute *Click-NDCG* @k and *Purchase-NDCG*@k for $k = \{4, 12, 24, 48\}$ by averaging *click-NDCG*$_s$ @k and *purchase-NDCG*$_s$@k given session s over all sessions in each dataset. Note that k is a multiple of 4 as that this shopping site displays 4 items per row on desktops.

5.3.2 Baselines

We compared *OPAR*'s ranking performance with 4 state-of-the-art baselines:

1. LambdaMART [32] is the boosted tree version of LambdaRank [6], which introduces the use of gradient boosted decision trees for solving a ranking task and won Track 1 of the 2010 Yahoo! Learning To Rank Challenge. A personalized search re-ranker is trained based on long-term user historical data to optimize for the user's purchasability on an item given the query issued and the user's historical preference.

2. Atr-KNN is derived from Item-KNN [11]. Each item is presented by n-hot-encoding of associated attributes with n being the cardinality of all attributes. That is, its i^{th} entry equals to 1 if the referred attribute presents in the item, otherwise 0. Items in the list I_{t+1} are re-ranked based on their euclidean-distance from the last engaged item(s) in I_t. Note that the items $x_i \in I_t$ with no-action has no impact on this re-ranking.

3. Atr-POP reranks the candidate set, I_t, of items based on the attributes' popularity estimated with held-out historical records. This baseline is one of the most common solutions derived from [11] given its simplicity and efficacy.

Table 8: Re-ranking performance comparison on over all data sets (top-left) and 7 category-specific data sets.

		Over All Category						Clothing					
		LambdaMART	Atr-KNN	Atr-POP	GRU4Rec	OPAR	OPAR$_w$	LambdaMART	Atr-KNN	Atr-POP	GRU4Rec	OPAR	OPAR$_w$
Purchase NDCG	@4	0.1795	0.0130	0.0749	0.0618	0.2994	**0.3042**	0.1948	0.0103	0.0516	0.0551	0.2384	**0.2494**
	@12	0.2629	0.0412	0.1323	0.1425	0.3505	**0.3607**	0.2670	0.0348	0.1269	0.0824	0.2685	**0.2744**
	@24	0.3162	0.1260	0.2112	0.2018	0.3718	**0.3900**	0.3019	0.0090	0.2193	0.1434	0.3209	**0.3263**
	@48	0.3724	0.2554	0.2861	0.2518	0.4512	**0.4578**	0.3774	0.2462	0.2784	0.2157	0.3976	**0.4030**
Click NDCG	@4	0.1459	0.0816	0.0705	0.0701	0.3120	**0.3158**	0.1328	0.0067	0.0690	0.0691	0.3058	**0.3197**
	@12	0.2265	0.1456	0.1264	0.1354	0.3213	**0.3229**	0.2137	0.0228	0.1224	0.1414	0.3126	**0.3257**
	@24	0.2955	0.2157	0.2021	0.1922	0.3318	**0.3489**	0.2821	0.0658	0.2045	0.1844	0.3274	**0.3424**
	@48	0.3815	0.3245	0.2813	0.2689	0.4047	**0.4051**	0.3711	0.2309	0.2807	0.2613	0.3988	**0.4061**

		Home & Living						Paper & Party Supplies					
		LambdaMART	Atr-KNN	Atr-POP	GRU4Rec	OPAR	OPAR$_w$	LambdaMART	Atr-KNN	Atr-POP	GRU4Rec	OPAR	OPAR$_w$
Purchase NDCG	@4	0.1755	0.0131	0.0649	0.0571	0.2920	**0.2952**	0.1822	0.0010	0.1255	0.0684	0.2828	**0.2965**
	@12	0.2670	0.0396	0.1226	0.1281	0.3391	**0.3436**	0.2667	0.0406	0.1692	0.0941	0.3367	**0.3497**
	@24	0.3218	0.0936	0.2066	0.1752	0.3838	**0.3879**	0.3276	0.1297	0.2469	0.1542	0.3796	**0.3905**
	@48	0.3874	0.2543	0.2789	0.2164	0.4462	**0.4491**	0.3876	0.2550	0.3216	0.1943	0.4291	**0.4399**
Click NDCG	@4	0.1481	0.0054	0.0601	0.0944	0.3201	**0.3219**	0.1585	0.0052	0.1084	0.0839	0.2825	**0.2874**
	@12	0.2294	0.0213	0.1175	0.1416	0.3244	**0.3256**	0.2394	0.0247	0.1586	0.1367	0.2931	**0.2973**
	@24	0.2978	0.0598	0.1973	0.1843	0.3485	**0.3491**	0.3103	0.0644	0.2300	0.1742	**0.3383**	0.3189
	@48	0.3835	0.2278	0.2746	0.2288	0.4032	**0.4086**	0.3911	0.2306	0.3104	0.2007	0.4017	**0.4072**

		Craft Supplies & Tools						Accessories					
		LambdaMART	Atr-KNN	Atr-POP	GRU4Rec	OPAR	OPAR$_w$	LambdaMART	Atr-KNN	Atr-POP	GRU4Rec	OPAR	OPAR$_w$
Purchase NDCG	@4	0.1912	0.0135	0.0739	0.0741	0.3101	**0.3268**	0.1954	0.0251	0.0683	0.0511	0.2166	**0.2178**
	@12	0.2735	0.0407	0.1296	0.1125	0.3673	**0.3781**	0.2828	0.0741	0.1431	0.0849	**0.2835**	0.2930
	@24	0.3272	0.1208	0.1970	0.1644	0.4084	**0.4188**	0.3324	0.1406	0.2510	0.1222	0.3304	**0.3361**
	@48	0.3844	0.2577	0.2820	0.2214	0.4366	**0.4750**	0.3869	0.2693	0.2917	0.1641	0.3962	**0.4020**
Click NDCG	@4	0.1458	0.0055	0.0749	0.0994	0.3118	**0.3166**	0.1502	0.0105	0.0673	0.0712	0.2495	**0.2605**
	@12	0.2262	0.0513	0.1290	0.1279	0.3241	**0.3293**	0.2324	0.0439	0.1358	0.1331	0.2656	**0.2708**
	@24	0.2955	0.2042	0.1953	0.1935	**0.3525**	0.3521	0.3006	0.1091	0.2398	0.1800	0.3155	**0.3212**
	@48	0.3811	0.2278	0.2815	0.2277	**0.4080**	0.4078	0.3848	0.2391	0.2885	0.2312	0.3548	**0.4029**

		Electronics & Accessories						Jewelry					
		LambdaMART	Atr-KNN	Atr-POP	GRU4Rec	OPAR	OPAR$_w$	LambdaMART	Atr-KNN	Atr-POP	GRU4Rec	OPAR	OPAR$_w$
Purchase NDCG	@4	0.2136	0.0501	0.0715	0.0814	0.2847	**0.2995**	0.1661	0.0060	0.0576	0.0718	0.3051	**0.3285**
	@12	0.3014	0.1109	0.1546	0.1223	0.3386	**0.3782**	0.2534	0.0766	0.1074	0.1142	0.3484	**0.3854**
	@24	0.3519	0.0176	0.2652	0.1674	**0.4257**	0.4152	0.3087	0.1470	0.1668	0.1847	0.3866	**0.3973**
	@48	0.4060	0.2965	0.2981	0.2416	0.4516	**0.4656**	0.3814	0.2460	0.2663	0.2367	0.4425	**0.4598**
Click NDCG	@4	0.1530	0.0267	0.0805	0.0641	**0.2074**	0.2051	0.0701	0.0027	0.0621	0.0614	0.3314	**0.3892**
	@12	0.2324	0.0703	0.1580	0.0939	0.2487	**0.2622**	0.1314	0.0106	0.1141	0.1021	0.3783	**0.3963**
	@24	0.3029	0.1410	0.2657	0.1345	**0.3158**	0.3120	0.1989	0.1276	0.1762	0.1647	0.3956	**0.4162**
	@48	0.3880	0.2560	0.3026	0.1667	0.3978	**0.4078**	0.3119	0.2192	0.2700	0.2144	0.4190	**0.4475**

Table 9: Multiple Purchase Intents within One Session

	Timestamp	Query	Query Taxonomy		Engaged Attributes
1st Purchase Intent	0	'flower girl basket'	paper and party supplies	(NO ACTION)	Browsing
	1-4	'flower girl basket wedding'	paper and party supplies	(CLICK)	'Prime Color: White', 'Occasion: Wedding', 'Holiday: Christmas', 'Wedding theme: Beach & tropical', 'Craft type: Floral arranging'
	5-9	'flower girl basket beach wedding'	paper and party supplies	(CLICK)	'Prime Color: Blue', 'Occasion: Wedding', 'Holiday: Christmas', 'Wedding theme: Beach & tropical', 'Secondary color: White', 'Craft type: Floral arranging'
	10-11	'two flower girl and one pillow'	paper and party supplies		Browsing
Purchase Intent Change					
2nd Purchase Intent	12-15	'hat for beach wedding'	clothing.women_clothing	(CLICK)	'Prime Color: Blue', 'Occasion: Wedding'
	16-22	'turquoise petals'	accesories	(CLICK)	'Prime Color: Blue', 'occasion: Bridal shower', 'Wedding theme: Fairytale & princess'
	23	'bride hair decoration beach theme'	clothing.women_clothing	(NO ACTION)	Browsing
Final Purchase	24	'turquoise petals'	accesories	(PURCHASE)	'Prime Color: Blue', 'Occasion: Bridal shower', 'Wedding theme: Fairytale & princess'

4. GRU4Rec [11] applies recurrent neural networks (RNN) on short session-based data of clicked items to achieve session-based next-item recommendation. Each session is encoded as a 1-of-N vector, in which the i^{th} entry is 1 if the corresponding item is clicked else 0, with N denoting the number of items. While the user's consecutive clicks on items are used in the next item prediction, it is attribute-agnostic.

While it is common for each arm in the bandits to represent a single item or product category, we skip it as a baseline here as this would incur higher exploration cost with potential latency bottleneck when scaling up to an inventory of hundred millions of items and also lose interpretability of product attributes.

5.4 Experimental Results

In this section, we describe experimentation results for evaluating three kinds of performance: (1) ranking performance, (2) impact of differentiating between user action types, and (3) interpretability.

5.4.1 Overall Re-ranking Performance

Table 8 shows experiment results of our model (*OPARs*) against 4 baselines described in Section 5.3. The results can be categorized into two parts: (1) performance on the aggregated datasets over all categories (top-left); and (2) performance on each of the 7 category-specific datasets, representing different shopping missions and behaviors across categories (i.e, "Clothing", "Home & Living"). Across all 8 datasets for the re-ranking task, *OPAR*$_w$ outperform against all 4 baselines, including LambdaMART, Atr-KNN, Atr-POP, and GRU4Rec in both purchase-NDCG and click-NDCG.

For the overall dataset (top-left), *OPAR*$_w$ shows over 6% lift in click-NDCG@48 compared to the best baseline, and over 20% increase in purchase-NDCG@48. Similar results are observed in each category-specific re-ranking. For k, the best improvement for *OPAR*$_w$ is achived at $k = 4$, ordering by $@4 >> @12 >> @24 >> @48$. With attribute-based bandits, interactive feedbacks from the in-session user actions, even just fewer clicks, efficiency propagate rewards to associated attributes and quickly learns preferred attributes that matter the most to the user, thus optimize user purchase intent.

5.4.2 Effectiveness of Action-aware MABs

To explore users' in-session activity with different types of actions (i.e, click, add-to-cart), we run experiments with the action-aware bandit model, with *OPAR*$_w$ hypertuned rewards from clicks vs add-to-carts, to differentiate *types* of user actions. The results in Table 8 are reported from a tuned model that assigns larger weights to *clicks* than *add-to-carts*, with an intuition that there is a high topical drift observed in the user's browsing intent after items are added to carts. As shown in Table 8, collectively *OPAR*$_w$ outperforms *OPAR* by 1.6% and 1.1% in purchase NDCG@4 and click NDCG@4, respectively. When segmenting by categories, *OPAR*$_w$ also outperforms *OPAR* in almost all categories, except *Electronics* & *Accessories* and *Craft Supplies* & *Tools* on purchase NDCG.

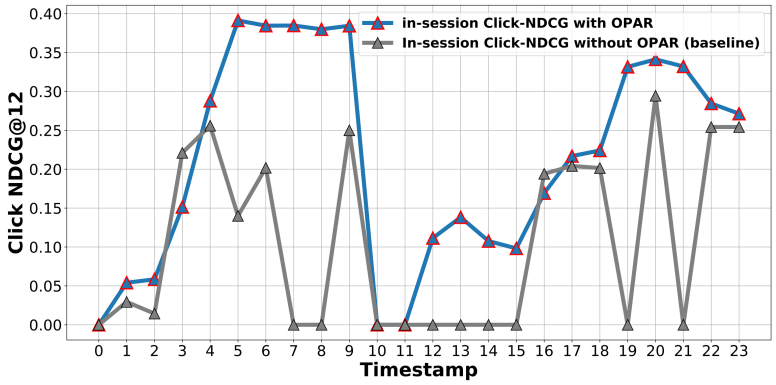
5.4.3 Interpretability of Within-Session Shopping Mission

It is often observed that a user exhibits multiple purchase intents with diverse preferences within a session. Table 9 presents a record of a user's in-session activities. Figure 4 (top) shows the sequential improvement of *OPAR* in session-level click-NDCG over time compared to the baseline, and Figure 4 (bottom) shows how *OPAR* captures user's preference, θ_{atr_h}, on 5 attributes over time. The "Engaged Attributes" column in Table 9 maps out all attributes associated with the clicked items for the corresponding query.

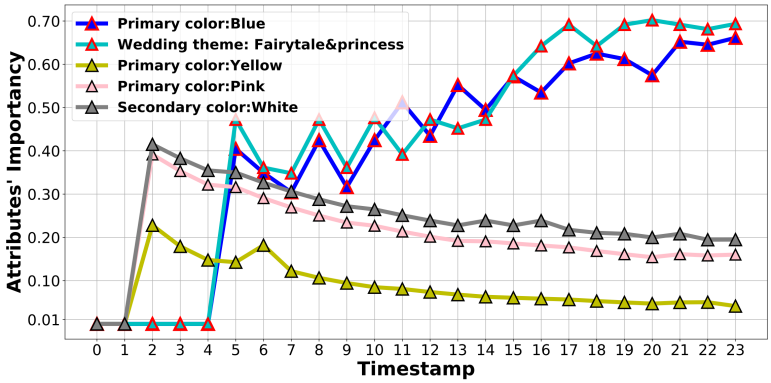
As shown in Table 9, the user is interested in three categories as his/her purchase intents: first in "paper & party supplies", then drift to "women clothing" and "accessories", and lastly converted in "accessories" with a *purchase*. After the browsing period from timestamp $t = 0$ with no user actions, β_{atr} for the attributes associated with the browsing-only items are incremented while no attributes have been updated with positive rewards for the given user. *OPAR* launches from a lower click-NDCG at the beginning, while obtains better re-ranking performance compared with baseline by learning that the user is interested in white prime color and is looking for the wedding occasion theme by the end of $t = 4$. From then *OPAR* outperforms the baseline in click NDCG while activated more attributes related to wedding themes in beach and tropical and expanded to floral crafting type and blue for prime color. The re-ranking performance continues to improve from $t = 5, .., 9$ as more items related to these attribute themes are discovered.

Starting from $t = 12$, the user starts to explore the 2nd categorical purchase intent, pivoting from "paper and party supplies" to "clothing" and "accessories". However, the latest activated attributes based on the engaged

items on the first set of shopping queries still relevant. The user has a consistent preference in attributes, such as “Prime Color: Blue”, “Occasion: Wedding”, and “Wedding theme: Fairytale & princess” as she is searching for a “hat for beach wedding” and/or “bride hair decoration beach theme”. Thus, for the second purchase intent starting at $t = 12$, we observe a high jump start in *OPAR*’s click NDCG at $t = 12$ comparing to the first intent at $t = 1$ and the metric continues to stepwise improve. As demonstrated in Figure 4 (bottom), “wedding theme” and “primary color: blue" are the top two performant attributes that *OPAR* learned and identified over time.



(a)



(b)

Figure 4: In-session *OPAR* re-ranking performance.

6 Conclusion

This paper proposes an interpretable *Online Personalized Attributed-based Re-ranker (OPAR)* as a light-weight third-pass, followed by the normal 2-stage ranking process, to personalize a buyer’s in-session experience based on product attributes. Given the important presence of attributes in the product category with its simplicity in explainability, we propose attribute-based multi-armed bandits to quickly learn the buyer’s fine-grained preferences and re-rank items based on the recent activities within the session to achieve in-session personalization. We then extend the reward function of the attribute-based bandits to weight based on the type of actions the buyer interacts with the item (i.e, click, add-to-cart, purchase). Lastly, we train and evaluate *OPAR* on the real-word e-commerce search ranking system, and show its superior performance against the baselines across multiples datasets. For future works, we could consider bias correction (i.e, position) in parameter updates to reduce self reinforcing, and model interactions between query and attributes to capture user preferences on attributes beyond

engaged items.

References

[1] S. Agrawal, N. Goyal. Analysis of thompson sampling for the multi-armed bandit problem. Conference on learning theory, 2012.

[2] P. Auer, N. Cesa-Bianchi, P. Fischer. Finite-time Analysis of the Multiarmed Bandit Problem. Machine Learning, 2002.

[3] P. Auer, N. Cesa-Bianchi, Y. Freund, R-E. Schapire. The Nonstochastic Multiarmed Bandit Problem. Society for Industrial and Applied Mathematics, 2003.

[4] T. Bai, Ting, J-Y. Nie, W-X. Zhao, Y. Zhu, P. Du, J-R. Wen, Ji-Rong. An Attribute-Aware Neural Attentive Model for Next Basket Recommendation. The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR '18), 2018.

[5] D-M. Blei, A-Y. Ng, M.I. Jordan. Latent Dirichlet Allocation. Journal of Machine Learning Research, 2003.

[6] C-J. Burges, R. Ragno, Q-V. Le. Learning to rank with nonsmooth cost functions. Advances in Neural Information Processing Systems, 2007.

[7] C. Olivier, L. Li. An Empirical Evaluation of Thompson Sampling. Advances in Neural Information Processing Systems 24, 2011.

[8] L. Guo, H. Yin, Q. Wang, T. Chen, A. Zhou, N. Quoc Viet Hung. Streaming Session-Based Recommendation. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2019.

[9] R. Guo, X. Zhao, A. Henderson, L. Hong, H. Liu. Debiasing Grid-based Product Search in E-commerce. Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '20), 2020.

[10] M. Haldar, P. Ramanathan, T. Sax, M. Abdool, L. Zhang, A. Mansawala, S. Yang, B. Turnbull, J. Liao. Improving Deep Learning for Airbnb Search. Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '20), 2020.

[11] B. Hidasi, A. Karatzoglou, L. Baltrunas, D. Tikk. Session-based Recommendations with Recurrent Neural Networks. arXiv: 1511.06939, 2015.

[12] L. Hu, L. Cao, S. Wang, G. Xu, J. Cao, Z. Gu. Diversifying Personalized Recommendation with User-Session Context. Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17), 2017.

[13] Y. Hu, Q. Da, A. Zeng, Y. Yu, Y. Xu. Reinforcement Learning to Rank in E-Commerce Search Engine: Formalization, Analysis, and Application. Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '18), 2018.

[14] J-T. Huang, A. Sharma, S. Sun, L. Xia, D. Zhang, P. Pronin, J. Padmanabhan, G. Ottaviano, L. Yang. Embedding-Based Retrieval in Facebook Search. Proceddings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '20), 2020.

[15] L. Shuai, K. Purushottam. Context-Aware Bandits. arXiv: 1510.03164, 2015.

[16] S. Li, B. Wang, S. Zhang, W. Chen, Wei. Contextual Combinatorial Cascading Bandits. Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48 (ICML'16), 2016.

[17] Z. Li, H. Zhao, Q. Liu, Z. Huang, T. Mei, E. Chen. Learning from History and Present: Next-Item Recommendation via Discriminatively Exploiting User Behaviors. Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2018.

[18] Q. Liu, Y. Zeng, R. Mokhosi, H. Zhang. STAMP: Short-Term Attention/Memory Priority Model for Session-Based Recommendation. Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2018.

[19] X. Liu, J. He, S. Duddy, L. O'Sullivan, Liz. Convolution-consistent collective matrix completion. Proceedings of the 28th ACM international conference on information and knowledge management, p:2209–2212, 2019.

[20] P. Loyola, C. Liu, Y. Hirate. Modeling User Session and Intent with an Attention-Based Encoder-Decoder Architecture. Proceedings of the Eleventh ACM Conference on Recommender Systems, 2017.

[21] P. Nigam, Y. Song, V. Mohan, V. Lakshman, W. Ding, A. Shingavi, H. Teo, H. Gu, B. Yin. Semantic Product Search. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2019.

[22] P. Pobrotyn, T. Bartczak, M. Synowiec, R. Białobrzeski, J. Bojar. Context-Aware Learning to Rank with Self-Attention. arXiv:2005.10084, 2020.

[23] R. Qiu, J. Li, Z. Huang, H. Yin. Rethinking the Item Order in Session-Based Recommendation with Graph Neural Networks. Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM '19), 2019.

[24] M. Quadrana, A. Karatzoglou, B. Hidasi, P. Cremonesi. Personalizing Session-Based Recommendations with Hierarchical Recurrent Neural Networks. Proceedings of the Eleventh ACM Conference on Recommender Systems (RecSys '17), 2017.

[25] J. Sanz-Cruzado, P. Castells, E. López. A Simple Multi-Armed Nearest-Neighbor Bandit for Interactive Recommendation. RecSys, 2019.

[26] R-S. Sutton, A-G. Barto, Andrew G. Reinforcement learning: An introduction. MIT press, 2018.

[27] Y-K. Tan, X. Xu, Y. Liu. Improved Recurrent Neural Networks for Session-Based Recommendations. Proceedings of the 1st Workshop on Deep Learning for Recommender Systems (DLRS'16), 2016.

[28] C-H. Teo, H. Nassif, D. Hill, S. Srinivasan, M. Goodman, V. Mohan, S-V-N. Vishwanathan. Adaptive, Personalized Diversity for Visual Discovery. Proceedings of the 10th ACM Conference on Recommender Systems (RecSys '16), p:35–38, 2016.

[29] Q. Wang, C. Zeng, W. Zhou, T. Li, S-S. Iyengar, L. Shwartz, G-Y. Grabarnik. Online Interactive Collaborative Filtering Using Multi-Armed Bandit with Dependent Arms. IEEE Transactions on Knowledge and Data Engineering, 2019.

[30] Y. Wang, L. Wang, Y. Li, D. He, W. Chen, T-Y. Liu. A theoretical analysis of NDCG ranking measures. COLT: Proceedings of the 26th annual conference on learning theory, volume 8, page 6, 2013.

[31] L. Wu, D. Hu, L. Hong, H. Liu. Turning clicks into purchases: Revenue optimization for product search in e-commerce. The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, 2018.

[32] Q. Wu, C. Burges, S. JC and K-M. Svore, J. Gao. Adapting boosting for information retrieval measures. Information Retrieval Journey, 2010.

[33] Y. Yan, Z. Liu, M. Zhao, W. Guo, W-P. Yan, Y. Bao. A practical deep online ranking system in e-commerce recommendation. Springer: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, p:186–201, 2018.

[34] F. Yu, Q. Liu, S. Wu, L. Wang, T. Tan, Tieniu. A Dynamic Recurrent Model for Next Basket Recommendation. Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '16), 2016.

[35] S. Zhang, L. Yao, A. Sun, Y. Tay. Deep Learning Based Recommender System: A Survey and New Perspectives. ACM Comput. Surv., 2019.

[36] X. Zhao, R. Louca, D. Hu, L. Hong. The Difference Between a Click and a Cart-Add: Learning Interaction-Specific Embeddings. Companion Proceedings of the Web Conference, 2020.

[37] Y. Zhao, Y-H. Zhou, M. Ou, H. Xu, N. Li. Maximizing Cumulative User Engagement in Sequential Recommendation: An Online Optimization Perspective. Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2020.

Improving Hierarchical Product Classification using Domain-specific Language Modelling

Alexander Brinkmann, Christian Bizer
University of Mannheim
{alex.brinkmann, chris}@informatik.uni-mannheim.de

Abstract

In order to deliver a coherent user experience, product aggregators such as market places or price portals integrate product offers from many web shops into a single product categorization hierarchy. Recently, transformer models have shown remarkable performance on various NLP tasks. These models are pre-trained on huge cross-domain text corpora using self-supervised learning and fine-tuned afterwards for specific downstream tasks. Research from other application domains indicates that additional self-supervised pre-training using domain-specific text corpora can further increase downstream performance without requiring additional task-specific training data. In this paper, we first show that transformers outperform a more traditional fastText-based classification technique on the task of assigning product offers from different web shops into a product hierarchy. Afterwards, we investigate whether it is possible to improve the performance of the transformer models by performing additional self-supervised pre-training using different corpora of product offers, which were extracted from the Common Crawl. Our experiments show that by using large numbers of related product offers for masked language modelling, it is possible to increase the performance of the transformer models by 1.22% in wF1 and 1.36% in hF1 reaching a performance of nearly 89% wF1.

1 Introduction

Product aggregators like market places or price portals support customers in finding the right offer for their desired product. To ensure a good customer experience, product aggregators integrate heterogeneous product offers from large numbers of online shops into their own product categorization hierarchy. This hierarchical product classification task is a major challenge for product aggregators as most shops use their own proprietary categorization hierarchy as well as diverse titles and descriptions for the same product. A promising technique to improve hierarchical product classification are pre-trained transformer models [5, 14]. These pre-trained transformer models have recently shown success for many NLP tasks [2, 3, 12, 13, 19, 22]. The training of transformer models involves two steps [2, 3]:

1. Pre-Training: The model is pre-trained on a huge corpus of texts from books, news, online forums and stories using self-supervised Masked Language Modelling (MLM).

2. Fine-Tuning: The pre-trained model is fine-tuned for downstream tasks using task-specific training data.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

During pre-training the model acquires general knowledge on language representation. This knowledge can be applied to solve down-stream tasks. In related work the pre-training step is extended by additionally pre-training the transformer model on domain-specific text corpora [4, 5, 7, 23]. In these works, the extended self-supervised pre-training results in improved performance on downstream tasks.

Motivated by these findings, we investigate whether additional pre-training using heterogeneous product offers from the Web can improve hierarchical product classification. For this purpose, we use product offers, which the Web Data Commons project[1] has extracted from the Common Crawl[2]. For identifying product offers and their attributes, the project relies on schema.org[3] annotations in the HTML pages of the web shops [20]. The annotations enable the reliable extraction of the offer's title, the description of the offered product, as well as the offer's categorization within the proprietary categorization hierarchy of the specific web shop. The heterogeneous category values are of special interest, because the categories contain information about the product classification of the web shop. While being heterogeneous and web shop specific, previous work has show that this knowledge about product categories is beneficial for categorizing products into a single central product hierarchy [9, 17].

We experiment with three different product corpora for pre-training that differ in size and relatedness to the downstream task. Through these different characteristics we measure the influence of size and relatedness of the pre-training corpus on the downstream hierarchical product classification task. Additionally, we experiment with different hierarchical classification methods. The methods combine $\text{RoBERTa}_{\text{base}}$ [3] with various classification heads in order to evaluate different approaches for exploiting the product hierarchy. We evaluate the classification methods using two product classification tasks involving product offers from many different web shops.

The contributions of this paper are as follows:

- We are the first to show that the performance of transformer models can be improved for the task of hierarchical product classification by performing additional pre-training using a corpus of related product offers.
- We show that using related product offers results in a better performance compared to randomly sampled product offers.

This paper is structured as follows: Section 2 introduces the classification models that will later be used for the experiments. Section 3 describes the evaluation tasks. While Section 4 presents the results of baseline experiments without additional language modelling. The effects of domain-specific MLM for hierarchical product classification are investigated in Section 5. Section 6 discusses related work. All data and code needed to replicate the results are available online[4].

2 Classification models

The architecture of all classification models is composed of a pre-trained $\text{RoBERTa}_{\text{base}}$ transformer model and a task-specific classification head. $\text{RoBERTa}_{\text{base}}$ is chosen due to its recent success on related Natural Language Processing (NLP) tasks [3]. Additionally, for one baseline model $\text{RoBERTa}_{\text{base}}$ is replaced by fastText[5], a state of the art neural network architecture for language representations [1]. For the classification the $\text{RoBERTa}_{\text{base}}$ model encodes the input text of the product offer. The first token of the encoded product offer is handed over to the classification head. Based on the first token, also referred to as [CLS] token, the classification head predicts a category for each product hierarchy level. Since it can be assumed that the product hierarchy contains valuable information, the given product classification challenge is tackled with three different classification heads. These

[1] http://webdatacommons.org/largescaleproductcorpus/v2/
[2] https://commoncrawl.org/
[3] https://schema.org/
[4] https://github.com/wbsg-uni-mannheim/productCategorization
[5] https://github.com/facebookresearch/fastText

classification heads try to exploit the upfront known product hierarchy. The three approaches are referred to as flat classification, hierarchical softmax and Recurrent Neural Network (RNN).

2.1 Flat Classification

For the flat classification, a linear layer makes a prediction based on the [CLS] token. As this classification approach only assigns product offers to lowest level of categories, the parent categories are inferred using the product hierarchy. For training a cross-entropy loss is used.

2.2 Hierarchical Softmax

The hierarchical softmax classification head predicts the category path with the highest probability for a product offer. To arrive at a probability for a category path, one local classifier is trained for each category in the product hierarchy. The local classifier predicts the probability of a category to be part of the category path. The product of all local predictions along a category path is the probability of a category path to be predicted for a product offer. Using softmax the most probable category path among all category paths is chosen. The input of the local classifiers is the transformer's [CLS] token. For training the cross-entropy loss is calculated per local classifier and per global category path. The combined loss deals with both the local impact of a single classifier and the global impact of a combination of classifiers along a category path.

2.3 Recurrent Neural Network

For the third classification head a RNN sequentially predicts a category for each level in the product hierarchy. The input for this classification head are the transformer's [CLS] token and a hidden state with the same size as the [CLS] token. Based [CLS] token and hidden state a linear layer predicts the category for the current product hierarchy level. A second linear layer updates the hidden state. The updated hidden state is fed back into the RNN to predict the next level in the product hierarchy. This procedure is repeated until a category is predicted for each level in the product hierarchy. During training the cross-entropy loss is calculated for each predicted category.

3 Evaluation Tasks

This section introduces the hierarchical product classification tasks that are used for the evaluation. The objective of the tasks is to assign product offers from different web shops to the correct categories in a single central product hierarchy.

3.1 MWPD Task

The MWPD task was used at the Mining the Web of Product Data (MWPD) challenge[6] [11] for benchmarking. The MWPD challenge was part of the International Semantic Web Conference (ISWC2020). In the product classification task of the MWPD challenge participants have to sort product offers from different web shops into the GS1 Global Product Classification standard (GPC)[7] [11]. GPC classifies product offers into a product hierarchy based on their essential properties and their relationship to other products. For the gold standard of the MWPD product classification data set the extracted product offers are manually assigned to the first three levels of the GPC.

[6] https://ir-ischool-uos.github.io/mwpd/
[7] https://www.gs1.org/standards/gpc

3.2 Icecat/WDC222 Task

The training set of the Icecat/WDC222 task[8] is built based on the Open Icecat product data catalogue[9]. The Open Icecat product data catalog provides well maintained and normalized product information. For this work the attributes title, description, category and Global Trade Item Number (GTIN) are considered. For the classification the first three levels of the product hierarchy are considered. In order to evaluate whether a classifier is able to correctly classify heterogeneous product offers, the test set of the Icecat/WDC222 task consists of selected product offers from the Web Data Commons (WDC) Product Corpus[10]. This corpus contains offers from 79 thousand different websites, which use schema.org annotations. Using the GTIN the product offers are assigned to one out of 222 leaf categories in the Icecat product hierarchy. All assignments are manually verified. For all product offers the values of the attributes title, descriptions and GTIN are extracted. As the Icecat training set contains normalized product offers and the WDC222 test set contains heterogeneous product offers, the Icecat/WDC222 task measures the transferability of a classifier trained on clean product offers and transferred to a scenario involving heterogeneous product offers.

Table 10 shows that the MWPD training set is small compared to the training set of the Icecat use case, but the product offers of the MWPD task are drawn from a comparably large number of different hosts. The WDC222 test set is again rather small but covers more hosts than the Icecat training set. These high numbers of hosts are an indication for more heterogeneity, because the product offers are differently represented by different hosts. The analysis of the median and maximum number of records per category of both use cases as shown in Table 11 reveals that the distribution of product offers among the categories is skewed towards a small number of categories. This distribution is common for hierarchical classification tasks [8]. The missing description values of the Icecat/WDC222 task are a sign that the description might harm a classifier's performance if the classifier is trained on the Icecat training set and applied to the WDC222 test set.

Evaluation Task	No. Records Train Set	No. Records Test Set	No. Hosts Train Set	No. Hosts Test Set	No. Nodes in Hierarchy	Avg. Depth Hierarchy
MWPD	10,012	3,107	1,547	878	396	3
Icecat/WDC222	765,743	2,984	1	112	410	2.44

Table 10: Evaluation Task Statistics

Evaluation Task	Data Set	Median No. Characters Title	Missing Values Description	Median No. Characters Description	Median No. Records per Category	Maximum No. Records per Category
MWPD	Train	50	0%	304	7	3,228
MWPD	Test	48	0%	365	4	799
Icecat/WDC222	Train	57	29.65%	1,099	215	145,020
Icecat/WDC222	Test	54	22.72%	140.5	3	516

Table 11: Attribute Statistics

[8] http://data.dws.informatik.uni-mannheim.de/largescaleproductcorpus/categorization/
[9] https://icecat.biz/en/menu/channelpartners/index.html
[10] http://webdatacommons.org/largescaleproductcorpus/v2/

4 Baseline Experiments

In order to set baselines, we apply the classification models described in Section 2 to both evaluation tasks that were introduced in Section 3. This section describes the setup as well as the results of the baseline experiments.

4.1 Evaluation Metrics

We use the average weighted F1 (wF1) score and the hierarchical F1 (hF1) score to evaluate the performance of the different models. Both scores are designed for hierarchical classification tasks [10, 11]. The wF1 score is calculated as proposed by the organisers of the MWPD challenge and shown in equation 25 [11].

$$Average\ weighted\ F1\ (wF1) = \sum_{j=1}^{L} \frac{1}{L} \sum_{i=1}^{K_j} \frac{n_i}{N} F_i \tag{25}$$

First, the F1 score of every category i in the hierarchy is calculated. To calculate the weighted F1 score per hierarchy level, the $F1_i$ score for each category i is weighted by number of true instances n_i for each category i divided by the total number of instances N across all categories K on a specific hierarchy level. For the hF1 score all target and prediction categories of the different levels in the product hierarchy are considered to calculate the F1 score. This way the hF1 score is suitable for hierarchical classification tasks, as it directs higher credit to partially correct classifications, considers the distance of errors to the correct category and errors higher up in the hierarchy are punished more severely [10]. Additionally, McNemar's significance test is applied to verify significantly different model performances on the test set [21]. For the test it is determined if a classifier's prediction is correct or incorrect first. Second, the numbers of correctly predicted product offers by the first classifier and incorrectly predicted product offers by the second classifier (correct/ incorrect) and vice versa (incorrect/ correct) are calculated. Using these numbers of correct/ incorrect and incorrect/ correct predictions as well as a significance level of 0.01, McNemar's test determines if the proportion of errors and consequently the performance of the two compared classifiers on the test set is significantly different.

4.2 Experimental Setup

We use the following hyperparameter setting for the experiments: The learning rate is set to 3e-5 for the Icecat/WDC222 task and to 5e-5 for the MWPD task. We use a batch size of 8 and a linear weight decay of 0.01. All fine-tuning experiments are run for 25 epochs on the MWPD data set and 10 epochs on the Icecat/WDC222 data set. The different learning rates and numbers of epochs are a result of multiple experiment runs. In this setting the average results on the test set over three randomly initialized runs are reported for every experiment. For McNemar's test a majority voting among the results of the different runs is performed. As input for the classification models the values of the attributes title and description are lowercased and excessive white-spaces are removed. For the experiments in this section a $\text{RoBERTa}_{\text{base}}$ model is used to obtain a product representation, which is consumed by different classification heads to obtain a classification.

4.3 Results

The naming convention <input attributes>-<transformer model>-<head> is used to refer to the different models. If the value of <input attribute> is "1", only the title is used as input. If the value of <input attribute> is "2", both title and description are used as input. In this section <transformer model> is either "base" for $\text{RoBERTa}_{\text{base}}$ or "fast" for fastText. The value of <head> refers to one of the classification heads introduced in Section 2 "flat", "hier" for hierarchical or "rnn". Experiments with the same capital letter in the column "Same Error Rate" share the same error proportion on the test set according to the significance test. Otherwise the experiment's error proportion is significantly different. The experimental results for the MWPD task are shown in Table 12. Setting

the results of the model 2-fast-flat into context to the other models shows that all transformer-based approaches outperform the fastText baseline model. A comparison of the models 1-base-flat and 2-base-flat reveals that adding the description as input improves the performance of the classification. The performance difference of the models 2-base-flat and 2-base-rnn is not significant and 2-base-hierarchical performs worse than the other two models. Thus, 2-base-flat is chosen as baseline model for the experiments with domain-specific language modelling as described in Section 5. Table 13 shows the results of the different models for the Icecat/WDC222 task. Again, the fastText based model 1-fast-flat is outperformed by the transformer-based models. The comparison of the models 1-base-flat and 2-base-flat shows that adding the description harms the performance of the trained classifier. This finding is expected given the high percentage of missing values and the difference in the median number of characters between training and test set as shown in Table 11. This comparison of the models 1-base-flat and 1-base-rnn shows that the RNN leads to a performance gain. A reason for this improvement might be the huge size of the Icecat training data set compared to the size of the MWPD data set. This size enables the RNN classification head to better learn the encoded hierarchy of the labels, which is beneficial for the classification on the test data set.

Model	Attributes	Classification Head	wF1	Δ wF1	hF1	Δ hF1	Same Error Rate
2-fast-flat	Title, Desc.	Flat	84.26		82.68		
1-base-flat	Title	Flat	87.01	2.75	87.03	4.35	
2-base-flat	Title, Desc.	Flat	87.52	3.26	87.62	4.94	A
2-base-hier	Title, Desc.	Hierarchical	87.00	2.74	87.47	4.79	
2-base-rnn	Title, Desc.	RNN	87.47	3.21	87.67	4.99	A

Table 12: Experimental results without Language Modelling - MWPD Task

Model	Attributes	Classification Head	wF1	Δ wF1	hF1	Δ hF1	Same Error Rate
1-fast-flat	Title	Flat	77.58		83.64		
1-base-flat	Title	Flat	83.36	5.78	84.69	1.05	
2-base-flat	Title, Desc.	Flat	80.91	3.33	81.48	-2.16	
1-base-rnn	Title	RNN	86.56	8.98	85.61	1.97	

Table 13: Experimental results without Language Modelling - Icecat/WDC222 Task

5 Domain-specific Language Modelling

After establishing baseline results in the previous section, we now investigate the effect of domain-specific language modelling on the performance of $\text{RoBERTa}_{\text{base}}$ models for hierarchical product classification. In this section the extraction of the domain-specific product offer corpora and the applied MLM approach are explained. The effects of domain-specific MLM are demonstrated by fine-tuning the newly pre-trained transformer models on the MWPD use case. The results of this fine-tuning are set into relation to the baseline results without domain-specific pre-training.

5.1 Product Corpora

In total three different product corpora are used for domain-specific MLM. All three product corpora contain product offers extracted from the WDC Product Corpus[11]. The WDC Product Corpus contains structured data for 365,577,281 product offers that are extracted from 581,482 different hosts using schema.org annotations. The hosts use schema.org annotations to enrich the search results of product aggregators with their web shop's product offers [20]. Three strategies are applied to retrieve product offers from the WDC Product Corpus. For the first two domain-specific product corpora 1,547 top-level domains of product offers from the MWPD training set are identified. Using these top-level domains, product offers from the same top-level domains are extracted from the WDC Product Corpus. This heuristic assumes that all products offered by a single shop (top-level domain) are related. The first product corpus contains 75,248 product offers and is called Small Related product corpus. The second product corpus contains 1,185,884 product offers and is referred to as Large Related product corpus. Through these two corpora the effect of the number of the product offer on MLM is measured. For the third corpus a large random sample of product offers is extracted from the WDC product corpus. This corpus is referred to as Large Random product corpus. The Large Random product corpus enables us to measure the effect of relatedness of product offers on domain-specific language modelling. Table 14 gives an overview of the product corpora's characteristics. For the two related product corpora the median number of records per hosts is higher compared to the Large Random product corpus. This shows the focus of the related corpora on the top-level domains extracted from the training set. The Large Random product corpus does not have this focus. Consequently, the number of hosts is a lot higher and the median number of records per host is lower.

Product Corpus	No. Records	No. Hosts	Median No. Records per Host	Max No. Records per Host
Small Related	75,248	1,160	100	400
Large Related	1,185,884	1,505	48	5,885
Large Random	1,029,063	98,421	2	2,878

Table 14: Size of Product Corpora

All extracted product offers have a title and at least one of the attributes description or category. The attributes are identified using the schema.org product annotations[12] name for title, description for description as well as category, breadcrumb and breadcrumbList for category. The attribute category is associated with multiple annotations, because different hosts use various annotations to categorise their products. Lastly, all attribute values are lowercased and excessive white spaces are removed. Table 15 shows that the product corpus Large Random has a comparably high percentage of missing description values. The Large Related product corpus has a lot of missing category values. These characteristics might influence the outcome of MLM.

Product Corpus	Median No. Characters Title	Median No. Characters Description	Missing Values Desccription	Median No. Characters Category	Missing Values Category
Small Related	38	310	10.43%	24	29.69%
Large Related	41	275	7.06%	22	68.90%
Large Random	34	39	72.99%	70	44.32%

Table 15: Distribution of Attributes in the Product Corpora

[11] http://webdatacommons.org/structureddata/2017-12/stats/schema_org_subsets.html
[12] https://schema.org/Product

5.2 Attribute Combinations

For MLM the attributes title, category and description are used. The product categories do not follow the categories of the downstream hierarchical product classification, because these categories assigned by the online shops themselves. Still these categories can contain valuable product information. In the basic setup the attribute values are concatenated to a single line text representation of the product. This default attribute combination is referred to as Title-Cat-Desc. To measure the effect of the heterogeneous categories, two additional product text representations are used for MLM. In one scenario only title and description are considered for MLM. The categories are disregarded. This scenario is referred to as Title-Desc and encoded in the model's name as <transformer model>$_{\text{nocat}}$. As alternative setup, the product attributes are split into two lines. One line contains the attribute values of title and category and the other line contains the attribute values of title and description. This scenario is referred to as Title-Cat/Title-Desc and encoded in the model's name as <transformer model>$_{\text{ext}}$. This way the influence of the heterogeneous categories during language modelling can be measured. Due to the smaller size and the low percentage of missing category values of the product corpus Small Related as shown in Table 14, the impact of using the category information on the model performance is evaluated using this product corpus.

5.3 MLM Procedure

The pre-training used to inject knowledge about product offers into the RoBERTa $_{\text{base}}$ model follows the MLM procedure used to pre-train RoBERTa $_{\text{base}}$ initially. During MLM in each epoch a random sample of tokens from the input sequence is selected and replaced by the special token [MASK]. Uniformly 15% of the input tokens are selected for possible replacement. Of these selected tokens, 80% are replaced with [MASK], 10% are left unchanged and 10% are replaced by a randomly selected vocabulary token. For MLM a language modelling head predicts the masked tokens of the input. The MLM objective is a cross-entropy loss on predicting the masked tokens [2, 3]. For the downstream hierarchical product classification the language modelling head is replaced by one of the task-specific classification heads introduced in Section 2.

5.4 Experimental Setup

For pre-training the RoBERTa$_{\text{base}}$ models on the different product corpora, the chosen hyperparameters are a batch size of 4, a learning rate of 5e-5 and a linear weight decay of 0.01. All models are pre-trained for 5 epochs. The downstream hierarchical product classification follows the same settings as the baseline experiments described in Section 4. In this setting the average results on the test set over three randomly initialized runs for each experimental setup are reported. Based on their usefulness for the baseline models both attributes title and description are used as input for hierarchical product classification on the MWPD task. For the Icecat/WDC222 task only the title is used as input. Since the collection of the product corpora focuses on the MWPD task, the conducted experiments with an extended domain-specific MLM focus on the MWPD task, too. The best performing pre-trained model on the MWPD task is transferred to the Icecat/WDC222 task. Table **??** and Table **??** show the experimental results of an extended domain-specific MLM for hierarchical product classification. To reference the different models the same encoding as in Section 4 is used. For <transformer model> the identifiers rel_s for pre-training on the Small Large corpus, rel_l for pre-training on the Related Large corpus and rand_l for pre-training on the Random Large corpus are added. Experiments with the same capital letter in the column "Same Error Rate" share the same error proportion on the test set according to McNemar's significance test. Otherwise the experiment's error proportion is significantly different.

Model	Product Corpus MLM	Attribute Combination MLM	Head	wF1	Δ wF1	hF1	Δ hF1	Same Error Rate
2-base-flat	None	None	Flat	87.52		87.62	4.94	B
2-rel_l-flat	Large Related	Title-Cat-Desc	Flat	87.61	0.09	87.70	0.08	B
2-rel_s-rnn	Small Related	Title-Cat-Desc	RNN	88.31	0.79	88.47	0.85	C
2-rel_l-rnn	Large Related	Title-Cat-Desc	RNN	88.74	1.22	88.80	1.18	
2-rand_l-rnn	Large Random	Title-Cat-Desc	RNN	88.19	0.67	88.34	0.72	
2-rel_l-hierar.	Large Related	Title-Cat-Desc	Hierar.	88.44	0.92	88.60	0.98	
2-rel_s_{nocat}-rnn	Small Related	Title-Desc	RNN	88.27	0.75	88.41	0.79	C
2-rel_s_{ext}-rnn	Small Related	Title-Cat/ Title-Desc	RNN	88.74	1.22	88.98	1.36	

Table 16: Experimental results with Language Modelling - MWPD Task

Model	Product Corpus MLM	Attribute Combination MLM	Head	wF1	Δ wF1	hF1	Δ hF1	Same Error Rate
1-base-rnn	None	None	Flat	86.56		85.58		D
1-rel_l-rnn	Large Related	Title-Cat-Desc	RNN	86.38	-0.18	85.61	+0.03	D

Table 17: Experimental results with Language Modelling - Icecat/WDC222 Task

5.5 Effect of Using Different Product Corpora

Table **??** shows that the baseline model 2-base-flat is outperformed by all other models on the MWPD task. Our best model 2-rel_l-rnn outperforms the baseline model 2-base-flat by 1.22 wF1 and 1.18 hF1 points. According to the significance test this performance difference is significant. This demonstrates the positive impact of domain-specific MLM on hierarchical product classification. The performance increase of the model 2-rel_l-rnn compared to the models 2-rel_s-rnn and 2-rand_l indicates that a large number of related product offers improves the model's performance the most. Among the classification heads, the results of the models 2-rel_l-flat, 2-rel_l-hier and 2-rel_l-rnn show that the RNN profits most from domain-specific pre-training. Table **??** reveals that the models 1-rel_l-rnn and 1-base-rnn have the same performance on the Icecat/WDC222 task. This underlines that the pre-training corpus has to be as similar as possible to the hierarchical product classification task to gain a significant performance boost from pre-training.

5.6 Effect of Using Web Shop Categories

The results in Table **??** indicate a slightly positive effect of using the heterogeneous categorization information from the original web shops during pre-training. A comparison of the models 2-rel_s_{nocat}-rnn and 2-rel_s-rnn shows that disregarding the web shop categories has a negative but not significant impact on the model's performance. In the extended scenario of model 2-rel_s_{ext}-rnn, the model's performance improves up to the performance level of the model 2-rel_l-rnn and significantly outperforms the baseline model 2-base-flat by 1.22 wF1 and 1.36 hF1 points on the MWPD task. A reason for these results might be that doubling the product representations during pre-training has a positive impact, because it almost doubles the amount of available text for pre-training. This effect is comparable to doubling the number of training epochs, which might improve the performance results, too. Another reason might be the length of the different attributes. The values of the attributes title and category are rather short compared to the attribute values of the description as shown by the

median number of characters in Table 11. If the product offer is represented by a single line, the generated masked tokens during MLM are more likely part of the description than part of title or category. If mainly tokens from the description are masked, the model learns to better represent these long descriptions. At the same time, it can be assumed that the title is more informative than the lengthy description. By presenting the product offers twice with different attribute combinations to the model during pre-training the disturbing effect of long descriptions is reduced. This allows the model to better exploit the heterogeneous categories and the title. From our results we can conclude that the heterogeneous categories from the web shops have a slightly positive but not significant impact on the model's performance.

6 Related Work

This section discusses related work on the domain adaptation of transformer models and gives an overview of the state of the art concerning hierarchical product classification using transformer models.

6.1 Domain Adaptation

The technique of pre-training transformer models on large text corpora and fine-tuning them for downstream tasks has proven successful in NLP [2, 3, 12]. BERT is one of these transformer models and was pre-trained on publicly available text corpora consisting such as the text of books and the English Wikipedia [2]. Through pre-training the model obtains the ability to encode natural language [13]. This knowledge about natural language is then transferred to downstream tasks. Pre-trained domain-specific models have shown that pre-training on domain-specific text improves the performance on downstream domain-specific tasks [4, 5, 7, 23]. E-BERT for example uses adaptive masking on a product and a review corpus during pre-training to learn e-Commerce knowledge on phrase-level and on product-level. Pre-training enables E-BERT to outperform a BERT based model on different downstream tasks related to e-commerce [5]. Comparing the effects of adaptive masking and random masking using the product offer corpora that were created for this paper is an interesting direction for future work.

6.2 Hierarchical Product Classification

Related work shows that exploiting the hierarchical structure can improve classification results [6, 8, 14–16]. The participants of the MWPD challenge show that in addition to exploiting the hierarchy, pre-trained transformer models can boost the results of hierarchical product classification tasks [11]. Team Rhinobird, the winners of the MWPD challenge, combine a pre-trained transformer model BERT with a hierarchical classification head [14]. Their Dynamic Masked Softmax classification head sequentially predicts the categories of different levels in the product hierarchy by actively restricting the classes, which can be predicted on the lower levels based on the predicted parent level node. Through different BERT based representations an ensemble of classifiers with the Dynamic Masked Softmax head enables Rhinobird to reach a wF1 score of 88.08 on the MWPD task that is used in this paper. Additionally, Rhinobird [14] applies pseudo labelling on the unlabeled test data to further improve the performance of their model. This procedure might leak information about the test set into the training process. Thus, we compare our models to the Rhinobird results without pseudo labelling. Our best model based on a domain-specifically pre-trained RoBERTa model and a RNN classification head achieves a performance of 88.74 wF1 points. This is an improvement of +0.66 points over Rhinobird's results. Given that Rhinobird achieves this good performance using an ensemble of models, future work could examine how an ensemble consisting of differently pre-trained and differently fine-tuned transformers can further improve the performance of our model. Team ASVinSpace uses a CNN based approach for language modelling with a multi-output classification head that predicts the categories of the different levels in the product hierarchy [18]. Our best model outperforms ASVinSpace's approach by +2.14 wF1 points.

7 Conclusion

Our results show that the performance of transformer models on hierarchical product classification tasks can be improved through domain-specific pre-training on a corpus of related product offers. All domain-specifically pre-trained and fine-tuned models outperform the baseline model, which relies on general pre-training and task-specific fine-tuning. Our experiments with three different domain-specific corpora of product offers demonstrate that a large corpus of related product offers leads to the highest performance gain. If we adjust the product offer representation during pre-training to exploit the special characteristics of the attributes title, description and category, the result on the hierarchical product classification task is further improved even though only a small corpus of related products is used for pre-training. With this approach and a task-specific classification head our best model outperforms the baseline model by 1.22 wF1 points and 1.36 hF1 points.

References

[1] A. Joulin, E. Grave, P. Bojanowski and T. Mikolov. Bag of Tricks for Efficient Text Classification. *15th Conference of the European Chapter of the Association for Computational Linguistics*, 2:427–431, 2017.

[2] J. Devlin, M. Chang, K. Lee and K. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 4171–4186, 2019.

[3] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer and V. Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv:1907.11692 [cs]*, 2019.

[4] J. Lee, W. Yoon, S. Kim, D. Kim, S. Kim, C. So and J. Kang. BioBERT: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36:1234–1240, 2019.

[5] D. Zhang, Z. Yuan, Y. Liu, Z. Fu, F. Zhuang, P. Wang, H. Chen and H. Xiong. E-BERT: A Phrase and Product Knowledge Enhanced Language Model for E-commerce. *arXiv:2009.02835 [cs]*, 2020.

[6] D. Gao, W. Yang, H. Zhou, Y. Wei, Y. Hu and H. Wang. Deep Hierarchical Classification for Category Prediction in E-commerce System. *3rd Workshop on e-Commerce and NLP (ECNLP)*, 64–68, 2020.

[7] I. Beltagy, K. Lo and A. Cohan. SciBERT: A Pretrained Language Model for Scientific Text. *Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 3615–3620, 2019.

[8] C. Silla and A. Freitas. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, 22:31–72, 2011.

[9] R. Meusel, A. Primpeli, C. Meilicke, H. Paulheim and C. Bizer. Exploiting Microdata Annotations to Consistently Categorize Product Offers at Web Scale. *International Conference on Electronic Commerce and Web Technologies*, 83–99, 2015.

[10] S. Kiritchenko, S. Matwin and A. Famili. Functional Annotation of Genes Using Hierarchical Text Categorization. *ACL Workshop on Linking Biological Literature, Ontologies and Databases: Mining Biological Semantics*, 2005.

[11] Z. Zhang, C. Bizer, R. Peeters and A. Primpeli. MWPD2020: Semantic Web challenge on Mining the Web of HTML-embedded product data. *CEUR Workshop Mining the Web of HTML-embedded Product Data*, 2720:2–18, 2020.

[12] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov and Q. Le. XLNet: Generalized Autoregressive Pretraining for Language Understanding. *33rd Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[13] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li and P. Liu. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Machine Learning Research*, 21:1–67, 2020.

[14] L. Yang, E. Shijia, S. Xu and Y. Xiang. Bert with Dynamic Masked Softmax and Pseudo Labeling for Hierarchical Product Classification. *CEUR Workshop Mining the Web of HTML-embedded Product Data*, 2720:2020.

[15] J. Wehrmann, R. Cerri and R. Barros. Hierarchical Multi-Label Classification Networks. *35th International Conference on Machine Learning*, 5075–5084, 2018.

[16] R. You, Z. Zhang, Z. Wang, S. Dai, H. Mamitsuka and S. Zhu. AttentionXML: Label Tree-based Attention-Aware Deep Model for High-Performance Extreme Multi-Label Text Classification. *33rd Conference on Neural Information Processing Systems (NeurIPS)*, 32:11, 2019.

[17] Z. Zhang and M. Paramita. Product Classification Using Microdata Annotations. *International Sematic Web Conference (ISWC)*, 716–732, 2019.

[18] J. Borst, E. Krner and A. Niekler. Language Model CNN-driven similarity matching and classification for HTML-embedded Product Data. *CEUR Workshop Mining the Web of HTML-embedded Product Data*, 2720:11, 2020.

[19] L. Dong, N. Yang, W. Wang, F. Wei, X. Liu, Y. Wang, J. Gao, M. Zhou and H. Hon. Unified Language Model Pre-training for Natural Language Understanding and Generation. *33rd Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[20] A. Primpeli, R. Peeters and C. Bizer. The WDC Training Dataset and Gold Standard for Large-Scale Product Matching. *International World Wide Web Conference (WWW)*, 381–386, 2019.

[21] T. Dietterich Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms. *Neural Computation*, 10:1895–1923, 1998.

[22] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma and R. Soricut. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. *International Conference on Learning Representations (ICLR)*, 2020.

[23] S. Gururangan, A. MarasoviÄ‡, S. Swayamdipta, K. Lo, I. Beltagy, D. Downey and N. Smith. Don't Stop Pretraining: Adapt Language Models to Domains and Tasks. *58th Annual Meeting of the Association for Computational Linguistics*, 8342–8360, 2020.

Deep Hierarchical Product Classification Based on Pre-Trained Multilingual Knowledge

Wen Zhang
wenzhaw@amazon

Yanbin Lu
luyanbin@amazon

Bella Dubrov
belladub@amazon

Zhi Xu
xuzhi@amazon

Shang Shang
shashang@amazon

Emilio Maldonado
emilim@amazon

Abstract

The customer experience of online shopping is largely contingent on the accuracy of product classification. Considering the amount of products and all the possible categories, it is desirable to construct a framework to auto-assign products into correct categories at scale. Machine learning based systems often suffer from poor data quality, such as incomplete item descriptions, adversarial noise in the training data, etc., causing low precision/recall of predictions. To overcome these difficulties, we propose a deep hierarchical product classifier based on BERT pretrained knowledge. Additionally, we propose several learning strategies, e.g., bootstrap learning, negative sampling, soft label and semantic augmentation, to capture consistent knowledge hidden behind noisy data to prevent overfitting. Experiments on a large data set with different data configurations prove the effectiveness of the proposed model.

1 Introduction

Online retailers provide a wide choice of commodities on their websites and a human curated taxonomy that allows customers to browse and discover products of interest. It is a challenging problem to be able to classify product into correct categories at scale. There are two main sources of information involved in the classification process: description of product and definition of categorization. However, considering large amount of products and the number of categories, catalog data are very often to be noisy. For example, product descriptions provided by content providers might be incomplete or misleading, and the multi-functionality of certain products might result in confusion between taxonomy categories. Human labeling is expensive. Hence, it is desirable to have an auto-product classification algorithm that can efficiently handle noisy data and make correct prediction.

The task defined in this study is to effectively extract the semantic representation of a large corpus of data for downstream classification tasks, i.e. large scale product classification. Learning semantic distributed representation has been well explored in recent years from the level of character representation to document representation, with the majority of the work initially focused on word/token embedding and later on sentence representation [3–5]. Recently, substantial work has shown benefits of using pre-trained models (PTMs) to better understand universal language representations and significantly reduce training time of new models. Two representative research projects, OpenAI GPT-1/2/3 (Generative Pre-training) [6–8] and BERT (Bidirectional

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Encoder Representation form Transformer) [9], designed very deep network structures with various pre-training tasks and achieved SOTA performance in many NLP benchmarks. In this study, we leverage a pre-trained version of BERT for multilingual understanding, named multilingual BERT (mBERT)[1]. It is a masked language model [10] trained with shared vocabulary and weights on Wikipedia text from the top 104 languages. We hope that product classification for multiple locales (marketplaces, most of time are referred by countries, where online retailers want to operate) with different languages can benefit from this pre-trained model's ability for multilingual understanding.

Generally, machine learning based systems, e.g. deep learning models, have been found to be vulnerable to various types of noise in the training data, such as corruptions or adversarial samples. Without proper configuration, models with large parameters easily converge to a poor local optimum that overfit to the noise/outliers. The backbone BERT model applied in this research carries millions of parameters, hence proper training strategies to overcome data noise are crucial to the success of the classification tasks. Learning from the task domain and data structures, we explore and design several training strategies to leverage product classification. For example, we create pseudo labels by self-justified learning and smooth model predictions via temperature scaling. These two mechanisms can adaptively prevent model from over-confidence. Accounting for multilabel learning with incomplete data source, negative sampling is also added during training. The performance of these strategies shed lights on model fine-tuning over the noisy data and provide hints for the designation of the next generation of model architectures.

The major contribution of this study can be summarized in 3 folds: 1) We discuss the basic types of data noises existing in practical product data sources; 2) We design a deep hierarchical classifier to efficiently predict product categories in both department and leaf category levels; 3) We propose several training strategies to further improve classification accuracy. To validate the proposed method, we conduct extensive experiments on 2 large independent datasets and also discuss the roles of each training strategy through an ablation study.

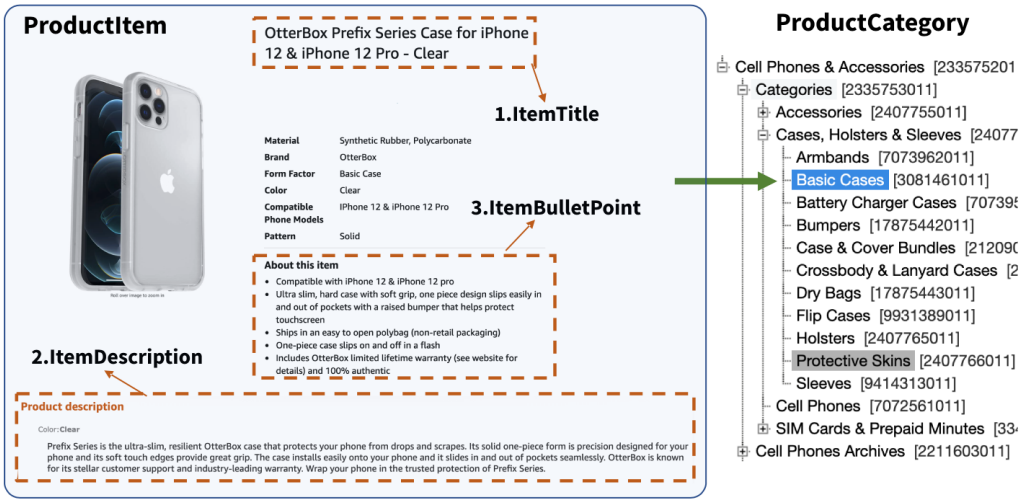


Figure 1: product item and category taxonomy

2 Product Data Quality and Noises

In this section, we will briefly introduce the common types of data noises that affect our classification model. The models of this study are based on natural language processing. Here, we only focus on the noise in text

[1]https://github.com/google-research/bert/blob/master/multilingual.md

information, e.g. descriptions of items and semantic definitions of their categories. These types of noise are also prevalent in multi-modal data but differ in their manifestation and noise distribution.

The text data of a product generally contains descriptions of the item properties. The product title information includes the major properties and functions of the item as a single sentence. Sometimes, the item's category name is already provided in the title. A more detailed description of the product, its various functions and properties is usually found in the item description in the form of several sentences or bullet points. Thus, sentence-level semantic understanding is crucial to extract the full value of this information. It is worth noting that all of the aforementioned data noise comes from the large raw data mostly provided by sellers, hence manual cleaning requires significant cost and time.

Item categories are predefined by online retailers according to catalog selection. The structure of categories, also termed as taxonomy, is the label space that we are targeting during classification. Hence, given different taxonomies, the accuracy of classification models varies due to changes in the complexity and dimension of the label space. In general, a taxonomy is manually curated by taxonomists as a hierarchical tree. Its granularity decides the size of item categories and learning logic from a given root node to its leaf children. Due to the selection are localized, taxonomy trees are not consistent across locales, thus a model capable of handling various label spaces is needed.

2.1 Incomplete and Misleading Item Descriptions

Incompleteness and misleading product description make the classification task challenging. In this study, we focus on three item text features: item title, item description and item bullet points. The information necessary for a correct classification might exist in each of them or across features. For example, in pet supplies, an aquarium mat, that should be classified as an aquarium decoration, might be confused with a dog bed mat, if the keyword "aquarium" is missing in the title. It is a common case, especially in a new locale where low quality data samples are abundant. Sometimes, incomplete item description comes with misleading information, which results in additional confusion for the model. In the previous case, the item bullet points of the aquarium mat focus on how to clean this item, misleading the classifier on the product usage instead of what the product is, and classify it as an aquarium cleaner. This problem might be severe if the other features are short or noisy. Another example is a fish tank which contains several aquarium decorations as gifts. The model needs adaptive ability to focus on important descriptions of the main product in a bundle.

2.2 Adversarial Item Information

This type of data noise is generated by the sellers who might change item descriptions to depict different product aspects, intentionally or not, leading to products potentially depicted for a different category. We can further characterize the noise into soft adversarial noise and hard adversarial noise. The soft adversarial noise emphasizes some peripheral properties of the item, while the hard adversarial noise contains mendacious item descriptions. In the former scenario, to increase the visibility of their products, some sellers tend to describe multiple functions or properties of the item in detail, which obscures its inherent purpose. It is analogous to the case of misleading item description discussed before. This noise increases the difficulties of a stable learning. In the latter scenario, item description is totally irrelevant to the actual selling item. For instance, a TV mount frame can camouflage as a TV with various sizes as long as the seller removes the keyword "mount" from the title. Interestingly, this type of adversarial noise sometimes couples with a correct item image, which suggests opportunities to introduce multimodal learning (such as using product images) during product classification.

2.3 Definition of Noise in Label Space

Another important yet hard to eliminate data noise is label noise. Unlike in the unsupervised setting, where the label space can be inferred during feature learning, we train our classifier with supervised knowledge. This

label space is constructed by taxonomists from a selected group of samples over a period of time, or inherited or migrated from an established stable ontology, e.g. item categories of a new locale is a child tree of a mature locale on an international online retailer. As data accumulates, the amount of items with previously unseen labels increases and thus the current taxonomy fails to capture the true label complexity. On the other hand, defined labels are not necessarily mutually exclusive, resulting in a multi-label learning problem. However, incomplete label space is inevitable in this setting. Confusing labels present additional difficulties for supervised classification.

3 Deep Hierarchical Product Classification Framework (DHPC)

The proposed model for product classification is a two-stage classifier, making the whole pipeline work as a hierarchical classifier. There are two main reasons. First, the label space of target product categories contains thousands of labels, which are sometimes confusing. Some techniques termed as extreme multilabel classification/ranking (XML/XMR) are designed to deal with this type of learning tasks with large label space. However, effective information mining from the large noisy dataset poses numerous challenges to reach high accuracy. The other reason for hierarchical classification is that taxonomy structure defined by internal taxonomists is a hierarchical tree. It is straightforward to incorporate this structure into our model. Besides, from the business perspective, a hierarchical classifier can help to identify departments with low accuracy and then to fine-tune them separately in the order of priority. In this paper, we denote the first level classifier as the department classifier, which assigns products to a department, e.g., electronics, fashion etc. Each locale generally contains dozens of departments. The second level classifier, called a leaf classifier, predicts items further into granular categories of each department. The number of categories ranges from hundreds to tens of thousands.

The whole pipeline is shown in Fig. 2. First, item text information including item title, description and bullet points are preprocessed to remove nonalphabetic characters and resample for balanced category distribution. Since the number of categories in department and leaf classifiers are distinct, we shall wisely select sampling threshold values. For department classifier, we randomly select hundreds of items per leaf node and combine all leaf node samples in a given department. For leaf classifier, we randomly select thousands of items per leaf node without replacement. After data preprocessing, the department and leaf classifiers are trained independently with an identical deep network structure. Details of the model structure will be presented in the next section.

The inference pipeline differs from the training schedule. Rather than independent querying, it works in cascade from department prediction down to leaf node prediction. As indicated in Fig. 2, the department classifier predicts the source item into the electronics department. The leaf classifier of the electronics department further predicts that item as a iPhone case. That is, we rank model prediction scores to find suggested department and then infer a leaf node within that department. During this process, a set of department prediction scores and the corresponding set of leaf prediction scores is generated. The two scores are then multiplied as confidence score for thresholding and model calibration.

3.1 Deep Neural Network Classifier

Considering model ability to handle multiple languages for various locales, we select a state-of-the-art multilingual model as the backbone structure. Specifically, we choose to use the pre-trained Bidirectional Encoder Representations from Transformers (BERT) in a multilingual setting. Our model is then fine-tuned on top of this architecture with an additional feed-forward neural network (FFNN) as the non-linear classifier. The model architecture is shown in Fig. 3 and consists of 4 important components: an embedding component, an transformer encoder, a feature pooled network and a non-linear multi-layer classifier. Parameters of the embedding component and transformer encoder are pre-trained on a large corpus of multilingual text.

In the first step, item text information is tokenized by a trained multilingual dictionary (contains 30522 words). Then, the embedding layer projects these tokenized words into a latent space. WordPiece embedding is

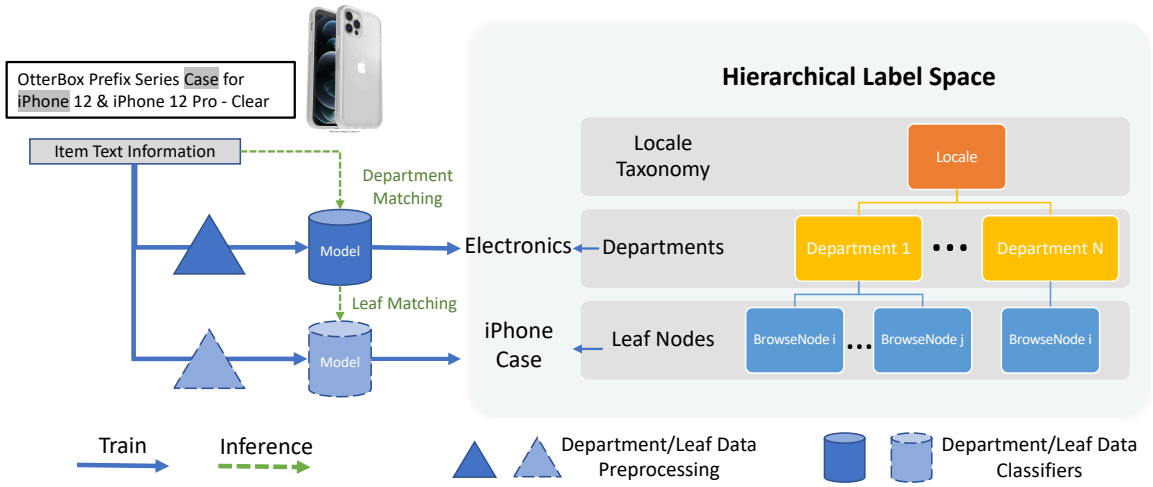


Figure 2: Hierarchical product classification framework (DHPC). The blue arrow and green arrow streams respectively indicate training and inference processes.

used here instead of the usual practice word embedding. During this process, segment embedding and position embedding are also concurrently encoded and added to the token embedding. It allows the model to understand sentence-wise sequential order. In our case, we use segment embedding to indicate whether words belong to item title, description or bullet points. The output of this embedding layer is a sequence of high dimensional symbol representations.

The transformer encoder maps the output of the embedding layer to a sentence/paragraph level semantic embedding vector, $E_{[CLS]}$. The original BERT transformer encoder contains 12 identical bidirectional transformer layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected FFNN. A residual connection is employed around each of the two sub-layers, followed by layer normalization. That is, the output of each sub-layer is LayerNorm(x+Sublayer(x)).

Knowledge Skip. Generally, supervised classification tasks use the first embedding vector $E_{[CLS]}$ of outputs of the transformer layer as the semantic representation. During experiments, we observed that keywords of product categories frequently exist in the title information, which is relatively easier to catch by attention than statement understanding. Therefore, in our model, we consider both deep and shallow encoded knowledge to enhance classification accuracy. $E_{[CLS]}$ from every other intermediate transformer layer are pooled together with the last one and then undergo a feature pooling network. There are several options for feature pooling, e.g., maxpool, meanpool or concatenation.

The last part of our model is a two-layer fully-connected network with BatchNorm and Dropout layers inside. The classification loss is binary cross entropy allowing for multi-label learning.

3.2 Model Accuracy and Coverage

Model accuracy and coverage are two essential evaluation metrics we use to quantitatively assess the model performance in a product classification system. Since the ground truth distribution of product categories is unequal, and the importance of categories is determined by business purposes, the preferred model must combine high accuracy with high coverage.

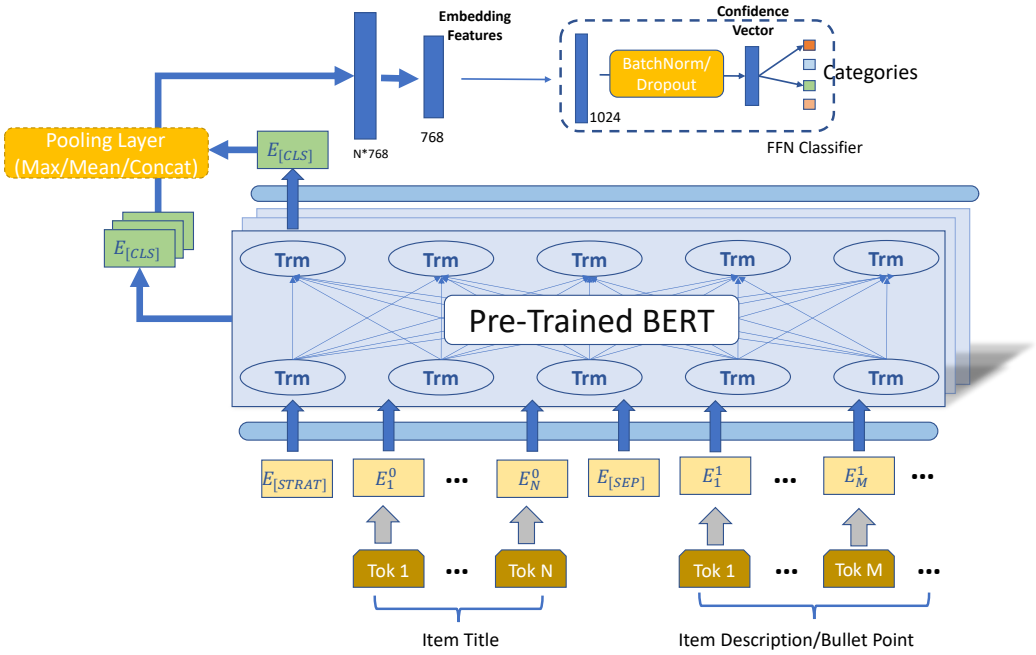


Figure 3: Deep hierarchical product classifier with pre-trained BERT knowledge

4 Training Strategy

Training a deep neural network on large noisy data, e.g. classifying millions of items with thousands of target labels, would easily result in overfitting and get trapped in a false local optimum, i.e. overconfidence. Especially in our cases, the product's text information contains mixed types of noise mentioned in the Sec. 2. To overcome these difficulties during the training process, general training strategies, such as early stopping, dropout, weight regularization/decay etc., are employed. Moreover, to further improve the model as well, we adopt the following training strategies.

4.1 Bootstrap Learning

The first training strategy is a self-justified learning mechanism accounting for knowledge consistency during training [1]. It augments the usual prediction objective with a notion of perceptual consistency, e.g. consistency of model predictions in each batch training. More specifically, it allows the model to disagree with a perceptually-inconsistent training label and effectively relabel the data while training. The assumption behind this idea is that incorrect labels are likely to be inconsistent with other data points predicted to the same label by the model. Therefore, it acts in a manner of self label clean-up and bootstraps itself until convergence to a stable knowledge. Here, we incorporate this idea into the cross-entropy training loss. Recalling the binary cross-entropy (BCE) loss:

$$L_{BCE}(p,q) = -\sum_{k=1}^{N} p_k log(q_k) + (1-p_k) log(1-q_k), \tag{26}$$

where p_k, q_k are ground truth label and model prediction, respectively. N is the size of target labels. To allow for differentiability of the proposed bootstrap loss, the label consistency is defined by predicted category probabilities q_k, hence the above BCE loss is modified as:

$$L_{Bootstrap}(p,q) = -\sum_{k=1}^{N} \beta p_k log(q_k) + \beta(1-p_k)log(1-q_k) + (1-\beta)q_k log(q_k), \tag{27}$$

where parameter $0 \leq \beta \leq 1$ balances bootstrap learning and supervised classification. It is empirically set in the range $[0.8, 0.95]$. Due to the large batch training steps (t_{batch}), we can set a delta activation $\hat{\beta}$ that adaptively turns on/off the bootstrap loss at a given global step T_{gate}:

$$\hat{\beta} = \begin{cases} 1, & \text{if } t_{batch} < T_{gate} \\ \beta, & \text{if } t_{batch} \geq T_{gate} \end{cases} \tag{28}$$

4.2 Negative Sampling

It is common that a product possess multiple functions, but we have limited knowledge of their complete category labels in the training data. The label tags are provided by sellers based on their limited knowledge of category taxonomy. Generally, one or a few positive labels will be allocated for a given item in the training set and the rest of labels are assumed to be negatively associated with it. However, in such a setting, negative samples outnumber positive samples dramatically, leading to a high precision but low recall. To avoid quick convergence to a false local optimum, we adopt negative sampling in the learning objective. Suppose, in Eq. 29, $N = N_{pos} + N_{neg}$, i.e., N_{pos} positive labels and N_{neg} negative labels. Eq. 27 is updated as:

$$L_{negsamp}(p,q) = -\sum_{k=1}^{N_{pos}} \hat{\beta} p_k log(q_k) - \frac{N_{neg}}{\sigma N} \sum_{j \sim P(\sigma)}^{N_{neg}} \hat{\beta}(1-p_k)log(1-q_k) - \sum_{k=1}^{N}(1-\hat{\beta})q_k log(q_k), \tag{29}$$

where $P(\sigma)$ is a probability function to select soft negative samples, and σ controls their contribution in the loss.

4.3 Soft Label with Temperature Scaling

Label smoothing is a way to make our model more robust so that it generalizes well, which is especially useful in big data mining. It was first introduced by Szegedy et al. [2] for multi-class supervised learning tasks in computer vision. It is originally designed for the last classifier layer with softmax function to prevent overconfidence. The mathematical formulation is simple:

$$p_k = (1-\epsilon)p_k + \epsilon u(k), \tag{30}$$

by mixing the ground truth label p_k with a scaling term $u(k)$. $u(k)$ is a distribution function in the label space and the simplest choice is a fixed scaling value, e.g. $\frac{1}{N}$. Hence, it prevents the largest logit (output of last layer) from becoming much larger than all others. We use a similar idea to scale prediction scores instead of the ground truth labels. In Eq. 29, q_k is the sigmoid output of logit, $q_k = sigmoid(logit_k)$. Therefore, following the same logic, we modify q_k as :

$$q_k = sigmoid(\frac{logit_k}{\epsilon(t_{batch})}), \tag{31}$$

where $\epsilon > 0$ can be a fixed value or a function of global batch training step t_{batch} allowing for adaptive scaling as we did in $\hat{\beta}$. It is worth noting that $\epsilon > 1$ and $\epsilon < 1$ carries quite different roles during training, forcing the model towards the opposite converging status. Setting $\epsilon > 1$ in the early training stage can help to explore potential local optima while setting $\epsilon < 1$ makes the model quickly converge to a local optimum.

4.4 Augmentation with Label Semantic Information

Label semantic information plays an essential role in product classification, since this knowledge is predefined to perceptively group items with their inherent properties. We also observe that, if text information of an item contains key words having a similar morphology of label name, it has a greater chance to be correctly predicted. Hence, we believe it is beneficial to incorporate the label information into our training data and force the model to understand these patterns. Specifically, we randomly chose a small portion of training data per leaf node and shuffling their title information by words. Then, we insert their labels' name in any place of the shuffled title. Their item description and bullet points are removed. Because, comparing with item title, other text information tend to follow the logic of natural language in sentences. Shuffling would destroy these properties.

5 Experiments

In this section, we include several experiments to validate the effectiveness of the proposed framework in two large datasets. We also conduct an analysis of product classification with different data settings, model structures, supervised learning goals and training strategies. Since these factors might positively affect our supervised learning tasks, we explore their potential and use cases. All experimental details are given below and results are reported with discussions.

5.1 Product Classification in 2 Large Datasets

This experiment is designed to evaluate the proposed model in real use cases. We try to understand how it performs under different language environments and data qualities.

5.1.1 Experimental Settings

Two independent datasets are sampled from two European locales (two countries, L1 and L2, using distinct languages from an online retailer) which contain millions of product items with text information in a newly build online shopping site. L1 data contains 50% of golden data source, i.e. positive labels of items were human-audited and hence high quality, and 50% general data, i.e., samples of original product data source with noise. L2 data is all general data which means it is noisier than L1 data. The major challenge of learning with these two datasets is that we do not have complete positive/negative label information for each item, e.g. labels are not fully explored. Discussion of data collections and label process can be found in Sec. 2 and Sec. 4.2. The training data contains several hundreds of millions of items with thousands of labels. The test set for each locale is sampled from catalog based on importance. It contains thousands of items distributed in thousands of labels. For these datasets, the text information is mixed with incomplete data source (i.e. missing item description or bullet point) that are deliberately constructed to assess practical use cases.

We chose a deep neural network (DNN) model as the baseline comparison. It encodes multi-view text inputs through an embedding layer and then pools the embeddings before a Multilayer Perceptron Classifier (MPC). The MPC consists of a few fully connected layers. This simple yet effective model successfully classify product item with high accuracy. It worth noting that the DNN model is trained from scratch for each dataset using the same set of hyper-parameters.

Accuracy score is calculated from the human assessment of the model prediction. The final prediction score is the multiplication of department confidence score and leaf confidence score, reflecting how much confidence a model has in the final leaf node predictions. This score is then used to threshold model prediction in order to reach a given accuracy demand. Please note, all of the assessment is weighted by the importance of departments.

Table 18: Comparison with DNN baseline model for department/leaf classification in the L1 data. Improvement(+)/Downgrade(-) of the department/leaf level accuracy is reported.

Department Name	Department Acc Improve	Leaf Acc Improve
Department 1	-0.0377	-0.0427
Department 2	-0.1230	+0.0258
Department 3	-0.0254	+0.0317
Department 4	+0.0065	-0.0433
Department 5	-0.0506	+0.0148
Department 6	-0.0578	+0.0599
Department 7	-0.0262	+0.0520
Department 8	+0.0016	+0.0819
Department 9	-0.0249	+0.0253
Department 10	-0.0658	-0.0067
Department 11	+0.0182	+0.0173
Department 12	+0.0321	+0.0771

5.1.2 Results

L1 Dataset. Hierarchical classification results of product items in L1 are shown in Tab. 18. Compared with the DNN baseline, the overall accuracy predicted by the proposed model is improved by 3.4% for leaf node predictions but slightly worse (drops 1%) for department predictions. Considering the chain effect in the hierarchical classification, within some departments, DHPC possesses a significantly better accuracy than the DNN model. For example, in Department 11 (the largest department) and Department 12 (the 3rd largest), DHPC is better in both department and leaf classifiers. However, we also observe that, in the 2nd largest department (Department 4), although the department classification beats DNN model, DHPC has an obvious worse accuracy in leaf node classification. After investigation, we found the discrepancy is caused from resampling strategies during data prepossessing since category distribution in this department is less balanced than in the other 2 largest departments.

L2 Dataset. L2 data is generally noisier than L1 data, hence the overall classification accuracies in the department and leaf level are lower than that of L1 product classification. In comparison, DHPC improves leaf node accuracy by 2% while the department level accuracy remains the same. Similarly, we further compare them in Department 3, 4, 6 (the top-3 largest departments). Except for the department classification of Department 3, all of the other department/leaf classifications indicate superior performance of the proposed model. Overall, DHPC excels in most department and leaf classifications in the L2 dataset.

Accuracy VS. Coverage. In addition to the accuracy assessed with the whole dataset, we attempt to evaluate the confidence of the proposed model, providing insights for model fine-tuning. To this end, we threshold model predictions based on confidence scores to allow at least 90% accuracy. For L1 dataset, DHPC outperforms DNN model by 13% more coverage after thresholding. For L2 dataset, we observe 13% more coverage at 90% accuracy compare to the DNN model.

5.2 Ablation Study of Proposed Training Strategies

In this section, we conduct several ablation studies to evaluate our proposed training strategies and discover their potentials. Parameter exploration in these experiments provides hints for future model design.

Table 19: Comparison with DNN baseline model for department/leaf classification in L2 data.

Department Name	Department Acc Improve	Leaf Acc Improve
Department 1	-0.1516	+0.0345
Department 2	+0.0324	+0.0231
Department 3	-0.0338	+0.0252
Department 4	+0.0091	+0.0259
Department 5	+0.0511	+0.1128
Department 6	+0.0085	+0.0441
Department 7	-0.0477	-0.0630
Department 8	+0.0909	+0.0489
Department 9	-0.0607	-0.0643
Department 10	+0.0417	+0.0913

5.2.1 Evaluation of Active Bootstrap Learning

Active bootstrap learning helps the model to capture the most consistent knowledge (self-confidence) during training and overcome overfitting issues. We explore various parameter choices for the L1 department classifier. β controls the strength of self-confidence. T_{gate} indicates at which global training step we will turn on this strategy, otherwise the model uses the traditional binary cross-entropy loss. From results in Tab. 20, we find early initialization of bootstrap learning, e.g. $T_{gate} = 0$ with a weak self-confidence setting, e.g. $\beta = 0.9$, offers the best performance. Hence, we argue that, for product classification, consistent knowledge captured at the very early stage of training contributes to high accuracy predictions.

Table 20: Influence of bootstrap learning in L1 department classification. Improvement (+) or Downgrade (-).

β	0.9	0.8	0.7	0.6
T_{gate} = 0	**+0.0148**	-0.0016	+0.0047	+0.0029
T_{gate} = 1000	**+0.0007**	-0.0015	-0.0023	-0.0038
T_{gate} = 3000	**-0.0011**	-0.0020	-0.0024	-0.0038

5.2.2 Model Pruning

We prune the original BERT model structures by using shallow embedding features for product classification. Specifically, we extract $E_{[CLS]}$ of the first 2 layers and then pooling ($PruneBert_2$), first 4 layers ($PruneBert_4$) and first 6 layers ($PruneBert_6$). We follow the same hyper-parameter settings for all pruned models and test their prediction power in two L1 product classification tasks, e.g., department classification and leaf node classification of Department 10. In this experiment, we do not add any noise handling training strategies introduced in Sec. 4. The prediction accuracy is reported in Tab. 21, which indicates a very close performance to the original BERT when using its first 4 layers. Considering that the original BERT has 110 MM parameters, this experiment suggests a simple structure of BERT with shallow encoding layers (half of total parameters) while effectively captures categorical information in our tasks.

5.2.3 Evaluation of Negative Sampling

Defining a suitable $P(\sigma)$ in Eq. 29 is challenging given the complex confusion patters across departments. In this preliminary study, we randomly select a portion of unlabeled categories (σ) as soft negative labels. Here, we explore σ settings from 0 to 60% with step size 10% for department and leaf classifiers respectively. As shown in

Table 21: Changes of department/leaf prediction accuracy after model pruning.

Model	$PruneBERT_2$	$PruneBERT_4$	$PruneBERT_6$
Department Classifier (L1)	-0.012	-0.004	-0.003
Leaf Classifier (Department 10, L1)	-0.015	-0.001	-0.003
Parameters	40MM	54MM	67MM

Table 22: Influence of negative sampling in department/leaf classification. Improvement (+) or Downgrade (-).

σ	0.9	0.8	0.7	0.6
Department Classifier (L1)	-0.013	-0.019	-0.011	-0.019
Leaf Classifier (Department 10, L1)	+0.006	+0.001	**+0.010**	+0.005

Tab. 22, the department classifier exerts the best performance when negative sampling is turned off, but the leaf node classifier embraces a large σ value, e.g. $\sigma = 0.7$ for leaf classifier of Department 10 in L1 data.

5.2.4 Evaluation of Soft Labels

We search smoothing factor ϵ in a wide range [1.5, 0.7] to provide a comprehensive understanding of soft labels for noisy data. We turn on this strategy at the very beginning of the training process since we do not found any opportunities if we delay it. The result in Tab. 23 shows that making smoothed soft labels, e.g., $\epsilon < 1$, helps our model towards high accuracy but it has to be carefully chosen to prevent from over-smoothing, e.g. downgrade if $\epsilon < 0.7$. We also find the necessity of more training epochs to allow a complete convergence (20% more training epochs).

Table 23: Influence of soft label in department/leaf classification. Improvement (+) or Downgrade (-).

ϵ	1.5	1.2	0.9	0.7
Department Classifier (L1)	-0.002	-0.001	**+0.015**	+0.005
Leaf Classifier (Department 10, L1)	-0.010	+0.001	**+0.010**	-0.002

6 Conclusions and Future Work

In this study, we propose a deep hierarchical classification framework for text-based product classification based on the pre-trained multilingual BERT model. We address typical types of data noises in the product data source and design several learning strategies to handle them. Experiments on two large datasets demonstrate the effectiveness of the proposed model. The additional ablation study further provides domain knowledge of optimal choices of training strategies under different data and task settings.

There are several interesting directions for further investigation. For example, we can improve classification accuracy by using a pre-trained model designed for languages of the targeting locale. Besides, multimodal information such as product images can be complementary sources as a feature. In addition, we can further upgrade the proposed training strategies by adding/replacing them with self-adaptive selection functions, such as label-semantic-based negative sampling.

References

[1] Reed, Scott, Honglak Lee, Dragomir Anguelov, Christian Szegedy, Dumitru Erhan, and Andrew Rabinovich. "Training deep neural networks on noisy labels with bootstrapping." arXiv preprint arXiv:1412.6596 (2014).

[2] Szegedy, Christian, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. "Rethinking the inception architecture for computer vision." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2818-2826. 2016.

[3] Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. "Glove: Global vectors for word representation." In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pp. 1532-1543. 2014.

[4] Bojanowski, Piotr, Edouard Grave, Armand Joulin, and Tomas Mikolov. "Enriching word vectors with subword information." Transactions of the Association for Computational Linguistics 5 (2017): 135-146.

[5] Peters, Matthew E., Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. "Deep contextualized word representations." arXiv preprint arXiv:1802.05365 (2018).

[6] Radford, Alec, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. "Improving language understanding by generative pre-training." (2018).

[7] Radford, Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. "Language models are unsupervised multitask learners." OpenAI blog 1, no. 8 (2019): 9.

[8] Brown, Tom B., Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan et al. "Language models are few-shot learners." arXiv preprint arXiv:2005.14165 (2020).

[9] Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).

[10] Taylor, Wilson L. "“Cloze procedure”: A new tool for measuring readability." Journalism quarterly 30, no. 4 (1953): 415-433.

[11] Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean. "Distilling the knowledge in a neural network." arXiv preprint arXiv:1503.02531 (2015).

[12] Liu, Jingzhou, Wei-Cheng Chang, Yuexin Wu, and Yiming Yang. "Deep learning for extreme multi-label text classification." In Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 115-124. 2017.

[13] Jain, Himanshu, Yashoteja Prabhu, and Manik Varma. "Extreme multi-label loss functions for recommendation, tagging, ranking & other missing label applications." In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 935-944. 2016.

[14] Goldberg, Yoav, and Omer Levy. "word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method." arXiv preprint arXiv:1402.3722 (2014).

../2021-06/tcde.pdf

IEEE Computer Society
10662 Los Vaqueros Circle
Los Alamitos, CA 90720-1314

Non-profit Org.
U.S. Postage
PAID
Los Alamitos, CA
Permit 1398