Practical Web-scale Recommender Systems

Yukihiro Tagami

August 2018

Abstract

Recommender systems have become ubiquitous on today's World Wide Web and are utilized by social networking sites, e-commerce sites, video-sharing sites, and Web portal sites to help users discover personalized content. These real-world systems need to deal with scalability issues caused by a huge number of users and candidate items while there is much user and context information that can be used for recommendation. In this thesis, we tackle two main challenges for constructing practical Web-scale recommender systems: 1) how to select items suitable for each user from a lot of candidate items in a very limited time, and 2) how to compose users' representations from users' activity histories.

For the former problem, we develop a ranking model suited for an inverted index-based fast retrieval system. We applied this approach to a real ad serving system and conducted A/B testing for evaluating its online performance. The system using our approach achieved significant improvements over the existing production system. Furthermore, we propose two multi-label classifiers that enable faster and more accurate predictions to be made by utilizing the tree and graph structure. Experimental results on several large-scale public datasets, which have hundreds of thousands of labels, show that our classifiers improve the trade-off between prediction and accuracy. At the same level of accuracy, the prediction time of our classifiers was up to 58 times shorter than that of a recent state-of-the-art method.

For the latter problem, we present representation learning for users' Web browsing sequences on the basis of analysis of our real-world Web visits data. The users' low-dimensional vector representations learned in an unsupervised manner are used among the user-related prediction tasks in common. For each prediction task, an individual classifier or regressor is trained by using these common vectors as features and task-specific users' properties or actions as targets. Our proposed method achieved better results than those of existing methods on two kinds of ad-related prediction problems based on logs from large-scale Web services. In addition, the prediction accuracies in some tasks were successfully improved by simply increasing the data size of users' Web browsing sequences as the training data sizes of prediction tasks themselves were not changed.

Published Work

This thesis contains content from the following publications:

- Yukihiro Tagami, Toru Hotta, Yusuke Tanaka, Shingo Ono, Koji Tsukamoto, Akira Tajima. Translation Method of Contextual Information into Textual Space of Advertisements. In *Proceedings of the 23rd International Conference on World Wide Web Companion (WWW2014 Posters)*, 2014. (Chapter 3)
- Yukihiro Tagami, Toru Hotta, Yusuke Tanaka, Shingo Ono, Koji Tsukamoto, Akira Tajima. Filling Context-Ad Vocabulary Gaps with Click Logs. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD2014)*, 2014. (Chapter 3)
- Yukihiro Tagami, Hayato Kobayashi, Shingo Ono, Akira Tajima. Modeling User Activities on the Web using Paragraph Vector. In *Proceedings of the 24th International Conference on World Wide Web Companion (WWW2015 Posters)*, 2015. (Chapter 5)
- Yukihiro Tagami, Hayato Kobayashi, Shingo Ono, Akira Tajima. Distributed Representations of Web Browsing Sequences for Ad Targeting. In *Proceedings of the 2nd International Workshop on Ad Targeting at Scale (TargetAd2016)*, 2016. (Chapter 5)
- Yukihiro Tagami. Learning Extreme Multi-label Tree-classifier via Nearest Neighbor Graph Partitioning. In *Proceedings of the 26th International Conference on World Wide Web Companion (WWW2017 Posters)*, 2017. (Chapter 4)
- Yukihiro Tagami. AnnexML: Approximate Nearest Neighbor Search for Extreme Multi-label Classification. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD2017)*, 2017. (Chapter 4)
- Yukihiro Tagami, Toru Hotta, Yusuke Tanaka, Shingo Ono, Koji Tsukamoto, Akira Tajima. Efficient Retrieval for Contextual Advertising Utilizing Past Click Logs. In *Transactions of the Japanese Society for Artificial Intelligence*, Volume 32, Number 6, pages A-H52_1-10, 2017 (in Japanese). (Chapter 3)
- Yukihiro Tagami, Hayato Kobayashi, Shingo Ono, Akira Tajima. Representation Learning for Users' Web Browsing Sequences. In *IEICE Transactions on Information and Systems*, Volume E101-D, Number 7, July 2018. (Chapter 5)
- Yukihiro Tagami. Speeding up Extreme Multi-label Classifier by Approximate Nearest Neighbor Search. In *IEICE Transactions on Information and Systems*, Volume E101-D, Number 11, November 2018. (Chapter 4)

Acknowledgements

First, I would like to thank my advisor Hisashi Kashima and members of his laboratory at Kyoto University. Whenever I went to Kyoto University from Tokyo, they warmly welcomed me and gave me helpful comments.

At Yahoo Japan Corporation, I have been working with many talented engineers and researchers. I would especially like to thank my great coauthors for many insightful discussions and comments. Hayato Kobayashi gave me a lot of research ideas and explained how to write a good paper. Shingo Ono and Akira Tajima, who were both my immediate bosses, encouraged me to take the doctoral course and made much effort to introduce an in-company Ph.D. supporting system.

Last but most importantly, this thesis would not have been completed without the generous support of my wife, Mina.

Contents

Chapter 1

Introduction

On the today's World Wide Web, users cannot go a single day without seeing items provided by recommender systems. For example, Facebook, Instagram, and Twitter show personalized news feeds or timelines for each user [31, 60]. YouTube recommendations help users to discover personalized content from a large number of videos [32, 11]. Contextual advertising, which is also a kind of recommendation task, also plays a crucial role in today's Internet ecosystem. These recommender systems are key components for Web services to differentiate themselves.

With the success of the Netflix Prize[1], it is widely known that matrix factorization-based collaborative filtering techniques are effective for rating prediction problems, which is one of recommendation task. However, as Netflix researchers themselves stated [46], Netflix's current recommender system consists of various algorithms to help people find TV programs and movies to watch, not just focusing on high predicted star ratings.

We consider there to be a gap between recommendation methods proposed in state-of-the-art studies and approaches used in real-world Web-scale recommender systems. Thus, in this thesis, we tackle some challenges for constructing Web-scale recommender systems.

In real-world recommendation settings, the candidate items typically change as time progresses. Some items are added while others are removed. In the case of news article recommendation, articles are uploaded over time, and old ones are deleted from the candidates. In the case of online advertising, advertisers start and stop advertising campaigns continually on the basis of their business plans. In short, candidate items frequently change. Therefore, recommender systems need to select items from candidates at the time the user requests. This selection should be done quickly. If it takes too long, the user's experience is spoiled since some or all of a Web page is not displayed. Real-world recommender systems respond to each request in a very short time by using various techniques such as pre-calculation and information retrieval methods.

[1]`https://www.netflixprize.com/`

Typical benchmark datasets used for comparing recommendation methods only include user ID and a few pieces of demographic information. For example, the MovieLens 100K and 1M datasets [49] contain users' simple demographic information (gender, age, occupation, and zip code)[2]. Similarly, the Last.fm[3] dataset includes user gender, age, and country information. In addition to the basic user information described above, various pieces of information can be obtained from users' actions on a Web site in real-world settings. In the YouTube recommendation task [32], users are represented as low-dimensional vectors on the basis of various user information, such as watched videos, search queries, and user language. For cross-domain user modeling, Elkahky et al. [40] proposed a multi-view deep learning approach that uses Web search queries obtained from Bing as user features. They applied this approach to real-world Microsoft product recommendation tasks: Windows Apps recommendation, news recommendation, and movie/TV recommendation. Thus, in real-world settings, useful user representations must also be made from these kinds of information.

To summarize, the requirements of "practical Web-scale recommender systems" that we consider in this thesis are two-fold:

1. Selecting suitable items for each user in a very short time
2. Making valuable users' representations from users' activity histories

For the first requirement, we develop an inverted index-based retrieval system for contextual advertising (in Chapter 3). On the top of the inverted index and efficient retrieval algorithm, our recommendation model enables each user's request to be responded to within tens of milliseconds. We applied this approach to a real ad serving system and conducted A/B testing for evaluating the online performance. The system using our approach achieved significant improvements over the existing production system. Furthermore, we tackle multi-label classification tasks with an extremely large number of labels (in Chapter 4). These tasks are called "extreme multi-label classification." Recommendation and ranking problems can be formulated as extreme multi-label classification tasks by treating each candidate as a separate label [53]. For these tasks, we propose two classifiers that can make a faster predictions by utilizing tree and graph structure. Experimental results on several large-scale public datasets show that our methods significantly improve the trade-off between prediction and accuracy, especially on data sets that have larger label space. At the same level of accuracy, the prediction time of our classifiers was up to 58 times shorter than that of a recent state-of-the-art method.

For the second requirement, we present a representation learning method for users' Web browsing sequences, which is easily obtained user information (in Chapter 5). Since users' representations obtained by this method are intended to be used among various

[2]On the other hand, bigger MovieLens datasets, such as MovieLens 10M and 20M datasets, just contain user IDs and do not have users' side information.

[3]`https://www.last.fm/`

recommendation tasks in common, we learn the model and representations in an unsupervised manner. For each prediction task, an individual classifier or regressor is trained by using these common vectors as features and task-specific users ' properties or actions as targets. Our method achieved better results than existing methods on two ad-related prediction problems based on logs from large-scale Web services. In addition, the prediction accuracies of some tasks were successfully improved by simply increasing the data size of users' Web browsing sequences as the data sizes of prediction tasks themselves were not changed.

To summarize, the main contributions of this thesis are as follows:

- We successfully construct an effective recommendation system that responds to each user's request in tens of milliseconds on top of an inverted index and efficient retrieval algorithm (Chapter 3)
- For multi-label classification tasks with hundreds of thousands of labels, we propose two classifiers that can make a faster and more accurate predictions by utilizing a tree and graph structure (Chapter 4)
- On the basis of an analysis of users' Web browsing sequences, we propose a representation learning method that obtains common and useful low-dimensional feature vectors among various user-related prediction tasks (Chapter 5)

The remainder of this thesis is organized as follows: Chapter 2 provides some background on machine learning and information retrieval techniques for constructing real-world recommender systems. Chapters 3, 4, and 5 present our proposed methods, as described above. Finally, Chapter 6 concludes this thesis and presents some directions for future work.

Chapter 2

Background

This chapter describes machine learning and information retrieval techniques for constructing real-world recommender systems.

2.1 Rating Prediction

Rating prediction tasks are well-known recommendation tasks that aim to predict user ratings for items on the basis of previous ratings. Thus, these tasks are typically regarded as regression problems. For example, in the Netflix Prize dataset [1], ratings are from 1 to 5 stars.

The rating of the q-th user for the i-th item is represented as $y_i^{(q)}$. The objective of these tasks is to predict some unknown rating $y_i^{(q)}$'s on the basis of already-known ratings.

These tasks are also called "matrix completion" because they are regarded as prediction problems of some elements in a rating matrix $\boldsymbol{Y} = [y_i^{(q)}]_{|Q|\times|I|} \in \mathbb{R}^{|Q|\times|I|}$. Here, Q and I are sets of user and item indices, respectively. Matrix factorization is commonly used for these tasks. Matrix factorization methods assume that a rating matrix has a low rank structure and factorize the matrix into two matrices $\boldsymbol{U} \in \mathbb{R}^{|Q|\times K}$ and $\boldsymbol{V} \in \mathbb{R}^{|I|\times K}$ such that $\boldsymbol{u}_q^{\mathrm{T}}\boldsymbol{v}_i \approx y_i^{(q)}$. Here, K is a pre-specified dimension size of latent factors, $\boldsymbol{u}_q \in \mathbb{R}^K$ is the q-th row vector of $\boldsymbol{U}$, and $\boldsymbol{v}_i \in \mathbb{R}^K$ is the i-th row vector of $\boldsymbol{V}$. In other words, $\boldsymbol{u}_q$ and $\boldsymbol{v}_i$ are latent factor vectors corresponding to the q-th user and i-th items, respectively.

A typical objective function to be minimized is defined as follows:

$$\sum_{(q,i)\in R} (\boldsymbol{u}_q^{\mathrm{T}}\boldsymbol{v}_i - y_i^{(q)})^2 + \lambda_u\|\boldsymbol{u}_q\|^2 + \lambda_v\|\boldsymbol{v}_i\|^2, \tag{2.1}$$

where R is a set of index pairs of user and item for already-known ratings. The second and last terms are L_2-regularization terms for avoiding over-fitting. λ_u and λ_v are pre-specified regularization parameters for user and item latent vectors, respectively.

Since Equation 2.1 is non-convex for $\boldsymbol{U}$ and $\boldsymbol{V}$, stochastic gradient methods are commonly utilized for optimization [28]. Instead of calculating the full gradient for all index pairs, these methods randomly choose a pair from R and iteratively optimize the objective function by calculating the gradient corresponding to the sampled pair. At each step, a certain index pair (q, i) is selected, then the objective function 2.1 is reduced to the following sub problem:

$$(\boldsymbol{u}_q^{\mathrm{T}} \boldsymbol{v}_i - y_i^{(q)})^2 + \lambda_u \boldsymbol{u}_q^{\mathrm{T}} \boldsymbol{u}_q + \lambda_v \boldsymbol{v}_i^{\mathrm{T}} \boldsymbol{v}_i.$$

Therefore, $\boldsymbol{u}_q$ and $\boldsymbol{v}_i$ are updated on the basis of the sampled gradient.

$$\begin{aligned}
\boldsymbol{u}_q &\leftarrow \boldsymbol{u}_q - \eta \left[(\boldsymbol{u}_q^{\mathrm{T}} \boldsymbol{v}_i - y_i^{(q)}) \boldsymbol{v}_i + \lambda_u \boldsymbol{u}_q \right], \\
\boldsymbol{v}_i &\leftarrow \boldsymbol{v}_i - \eta \left[(\boldsymbol{u}_q^{\mathrm{T}} \boldsymbol{v}_i - y_i^{(q)}) \boldsymbol{u}_q + \lambda_v \boldsymbol{v}_i \right],
\end{aligned}$$

where η is a learning rate. This learning rate is typically a constant value, but can be dynamically adjusted for faster convergence [29].

After $\boldsymbol{U}$ and $\boldsymbol{V}$ have been iteratively updated, the already-known ratings are well approximated by using these matrices. Then, unknown ratings are also predicted by $\hat{y}_i^{(q)} = \boldsymbol{u}_q^{\mathrm{T}} \boldsymbol{v}_i$ for index pairs $(q, i) \notin R$. The items that have higher predicted rating values $\hat{y}_i^{(q)}$ are recommended to the q-th user.

2.2 Ranking

In the above rating prediction settings, recommender systems first predict items' ratings, and then present top-rated items for each user. However, the objective of recommender systems is to select of items that match a user's preference. Therefore, recommender systems do not need to predict rating scores directly but only have to distinguish the preference order of some items. Thus, recommendation tasks are essentially regarded as ranking problems. In this section, we present basic approaches for ranking problems.

We use "user" and "query" interchangeably since we consider user information as query for a recommender (or ranking) system.

2.2.1 Evaluation metrics

In this subsection, we present widely used evaluation metrics for ranking tasks. For more details, refer to Chapter 8 in the book written by Manning et al. [70]. In the following, $\pi(q, k) = j$ means that the j-th item ranks in the k-th position by the predicted score for q-th query.

Mean average precision (MAP) is defined as follows:

$$\begin{aligned}
\mathrm{MAP} &= \frac{1}{|Q|}\sum_{q\in Q}\mathrm{AP}_q,\\
\mathrm{AP}_q &= \frac{1}{\sum_{i=1}^{N_q} z_j^{(q)}}\sum_{k=1}^{N_q} z_{\pi(q,k)}^{(q)}\mathrm{P}_q@k,\\
\mathrm{P}_q@k &= \frac{1}{k}\sum_{l=1}^{k} z_{\pi(q,l)}^{(q)}.
\end{aligned}$$

where AP_q is the average precision for q-th query and $\mathrm{P}_q@k$ is precision of the predicted top-k items for q-th query. $z_i^{(q)} = 1$ if the i-th item matches the q-th query, and $z_i^{(q)} = 0$ otherwise. In other words, by using threshold parameter t, $z_i^{(q)} = 1$ when $y_i^{(q)} \geq t$, and $z_i^{(q)} = 0$ when $y_i^{(q)} < t$.

Normalized discounted cumulative gain (nDCG) is another widely used metric for ranking tasks. However, unlike MAP, nDCG is a metric that considers relevance and predicted rank and becomes larger when more relevant items are located at higher ranks. In other words, nDCG uses $y_i^{(q)} \in \mathbb{Z}$ instead of $z_i^{(q)} \in \{0, 1\}$, which MAP uses.

$$\begin{aligned}
\mathrm{nDCG} &= \frac{1}{|Q|}\sum_{q\in Q}\frac{\mathrm{DCG}_q@N_q}{\max_\pi \mathrm{DCG}_q@N_q},\\
\mathrm{DCG}_q@k &= \sum_{l=1}^{k}\frac{y_{\pi(q,l)}^{(q)}}{\log_2(l+1)}.
\end{aligned}$$

The denominator of $\mathrm{DCG}_q@k$ ($\log_2(l + 1)$) represents that nDCG becomes smaller as relevant items are allocated at lower positions.

MAP and nDCG are defined by using a whole item set for a query. In the typical ranking setting, P@k and nDCG@k, which are defined by using the predicted top-k items, are also used:

$$\begin{aligned}
\mathrm{P}@k &= \frac{1}{|Q|}\sum_{q\in Q}\mathrm{P}_q@k,\\
\mathrm{nDCG}@k &= \frac{1}{|Q|}\sum_{q\in Q}\frac{\mathrm{DCG}_q@k}{\max_\pi \mathrm{DCG}_q@k}.
\end{aligned}$$

2.2.2 Pairwise approaches

In the ranking problem setting, a list of recommended items ordered by predicted scores is more important than score values themselves.

A feature vector extracted from pairs of q-th query and i-th item is represented as $\boldsymbol{x}_i^{(q)}$. Our objective is to obtain a scoring function $f(\cdot)$ such that the order of scores $f(\boldsymbol{x}_i^{(q)}) > \cdots > f(\boldsymbol{x}_j^{(q)}) > \cdots > f(\boldsymbol{x}_j^{(q)})$ strongly correlates with the desired order. Of course, as described above, we ultimately focus on the ordered list rather than scores. However, because of simplicity of implementation, this approach is widely used.

Pairwise approaches focus on magnitude of relationship between item pairs in the candidates. If the q-th user prefer the i-th item to the j-th one, ranking scores are expected to maintain the relationship $f(\boldsymbol{x}_i^{(q)}) > f(\boldsymbol{x}_j^{(q)})$. Thus, objective function of pairwise approaches is defined as follows:

$$\min_f \sum_{q\in Q} \sum_{(i,j)\in P_q} L\left(f(\boldsymbol{x}_i^{(q)}), f(\boldsymbol{x}_j^{(q)})\right) + \Omega(f), \tag{2.2}$$

where $P_q = \{(i,j) \mid y_i^{(q)} > y_j^{(q)}\}$ is a set of index pairs for the q-th query, $L(\cdot,\cdot)$ is a loss function, and $\Omega(f)$ is a regularization function (or complexity metric) for scoring function $f(\cdot)$.

Ranking support vector machine (Rank SVM) [56] is a pairwise method. By using hinge loss, the loss function is represented as follows:

$$\begin{aligned} L\left(f(\boldsymbol{x}_i^{(q)}), f(\boldsymbol{x}_j^{(q)})\right) &= \max\left(0, 1 - \left(f(\boldsymbol{x}_i^{(q)}) - f(\boldsymbol{x}_j^{(q)})\right)\right) \\ &= \max\left(0, 1 - \Delta_{ij}^{(q)}\right), \end{aligned}$$

where $\Delta_{ij}^{(q)} = f(\boldsymbol{x}_i^{(q)}) - f(\boldsymbol{x}_j^{(q)})$ is the difference among scores. RankSVM aims to learn function $f(\cdot)$ where $\Delta_{ij}^{(q)} \geq 1$ holds for as many pairs $(i,j) \in P_q$ as possible.

If we simply use a linear scoring function $f(\boldsymbol{x}_i^{(q)}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i^{(q)}$ with a weight vector $\boldsymbol{w}$, the difference among scores becomes: $\Delta_{ij}^{(q)} = \boldsymbol{w}^{\mathrm{T}}\left(\boldsymbol{x}_i^{(q)} - \boldsymbol{x}_j^{(q)}\right)$. Therefore, by using the L_2 regularization term $\Omega(f) = \frac{1}{2}\|\boldsymbol{w}\|^2$, the objective function 2.2 is represented as follows:

$$\min_f \sum_{q\in Q} \sum_{(i,j)\in P_q} \max(0, 1 - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\delta}_{ij}^{(q)}) + \frac{1}{2}\|\boldsymbol{w}\|^2.$$

Here, $\boldsymbol{\delta}_{ij}^{(q)} := \boldsymbol{x}_i^{(q)} - \boldsymbol{x}_j^{(q)}$ is the difference between two feature vectors. Since this objective function is the same as that of the standard SVM, fast linear SVM implementation such as LIBLINEAR [41] can be utilized to optimize it.

RankNet [20] is another pairwise method. The cross entropy loss is used as a loss function as follows:

$$\begin{aligned} L\left(f(\boldsymbol{x}_i^{(q)}), f(\boldsymbol{x}_j^{(q)})\right) &= -\log \frac{\exp(f(\boldsymbol{x}_i^{(q)}))}{\exp(f(\boldsymbol{x}_i^{(q)})) + \exp(f(\boldsymbol{x}_j^{(q)}))} \\ &= -\log \frac{1}{1 + \exp\left(f(\boldsymbol{x}_j^{(q)}) - f(\boldsymbol{x}_i^{(q)})\right)} \\ &= \log\left(1 + \exp\left(f(\boldsymbol{x}_j^{(q)}) - f(\boldsymbol{x}_i^{(q)})\right)\right) \\ &= \log\left(1 + \exp\left(-\Delta_{ij}^{(q)}\right)\right). \end{aligned}$$

Burges et al. [20] used a neural network as the scoring function $f(\cdot)$ of RankNet. In this case, the neural network is learned via back-propagation [63]. Of course, if we just use a linear scoring function $f(\boldsymbol{x}_i^{(q)}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i^{(q)}$ and L_2 regularization term $\Omega(f) = \frac{1}{2}\|\boldsymbol{w}\|^2$, the objective function is simplified to that of the L_2 regularized logistic regression. Again, a fast solver such as LIBLINEAR [41] can efficiently optimize the objective function.

2.2.3 Listwise approaches

Cao et al. [24] showed that the pairwise loss of RankNet does not inversely correlate with nDCG in some cases. That is, nDCG starts to drop after some iterations during the learning phase even if the pairwise loss continues to decrease. Thus, for tailoring a loss function that inversely correlates with MAP or nDCG, listwise methods have been studied. Note that MAP and nDCG are not easy to directly optimize since these metrics are non-smooth with respect to the scores [22] and we can not simply apply gradient-based methods to optimize them.

Listwise methods focus on the order relationship in the item list itself.

$$\min_f \sum_{q\in Q} L\left(\{f(\boldsymbol{x}_i^{(q)}), y_i^{(q)}\}_{i=1}^{N_q}\right) + \Omega(f) \tag{2.3}$$

The loss function of ListNet [24] is defined by using cross entropy loss as follows:

$$L\left(\{f(\boldsymbol{x}_i^{(q)}), y_i^{(q)}\}_{i=1}^{N_q}\right) = -\sum_{i=1}^{N_q} \phi_i^{(q)} \log \psi_i^{(q)},$$

$$\text{where } \phi_i^{(q)} = \frac{\exp(y_i^{(q)})}{\sum_{k=1}^{N_q}\exp(y_k^{(q)})},\ \psi_i^{(q)} = \frac{\exp(f(\boldsymbol{x}_i^{(q)}))}{\sum_{k=1}^{N_q}\exp(f(\boldsymbol{x}_k^{(q)}))}.$$

ListNet can be viewed as an extended version of RankNet. Whereas RankNet uses item pairs generated from item lists as instances, ListNet uses item lists themselves.

The loss function of ListNet is originally defined on the basis of top-k probability. Here, we represent the loss function of top-1 ListNet, which only considers top-1 probability. The model training in the case of $k \geq 2$ has high computational complexity because a large number of item permutations should be considered. Thus, top-1 ListNet was used in the experiments in the original paper [24] and other studies [110]. Luo et al. [69] proposed a stochastic ListNet approach that reduces the complexity by computing the gradient within a limited permutation subset.

ListMLE [110] is another listwise method. The loss function is defined as follows:

$$L\left(\{f(\boldsymbol{x}_i^{(q)}), y_i^{(q)}\}_{i=1}^{N_q}\right) = -\sum_{k=1}^{N_q} \log \varphi_k^{(q)},$$

$$\text{where } \varphi_k^{(q)} = \frac{\exp(f(\boldsymbol{x}_{\tau(q,k)}^{(q)}))}{\sum_{l=k}^{N_q}\exp(f(\boldsymbol{x}_{\tau(q,l)}^{(q)})}.$$

Here, $\tau(q,k) = j$ means the j-th items ranks in the k-th position by the relevance score $y_j^{(q)}$ for query q.

Xia et al. investigated the desired properties of loss function for ranking problems [110]. They pointed out that cross entropy loss, which ListNet uses, is not very "sound." Soundness of a loss function guarantees that the loss can well represent the ranking problem. For example, an incorrect prediction should receive a larger penalty than a correct prediction, and the penalty should reflect the confidence of prediction [110]. If we apply a monotonically increasing function (such as $\exp, \log$, or sqrt) to target score $y_i^{(q)}$, the preference order of items does not change but cross entropy loss does. On the other hand, likelihood loss, which ListMLE utilizes, is not affected by this mapping function since this loss focuses on the order relationship in the item list.

LambdaRank [22, 38] and LambdaMART [21, 23] are regarded as a combination approach of pairwise and listwise methods. LamdaRank focuses on item pairs in each list, like RankNet does. However, while RankNet treats all pairs equally in the gradient calculation for updates, LambdaRank weights gradients of pairs on the basis of their contributions for target metrics, such as MAP and nDCG. In other words, LambdRank utilizes the weighted sum of the gradients for updates, by considering their positions of items in the list sorted by current predicted scores. LambdaMART uses gradient boosting with trees instead of the above gradient-based model updates. For more details of these methods, refer to the Burges's paper [23].

2.3 Top-k Retrieval

As described in the previous section, recommender systems only have to obtain top-k scored items from a large number of candidates for each user. Therefore, the item selection tasks are regarded as information retrieval tasks.

In this section, we describe two kinds of top-k retrieval approaches. One is for high dimensional sparse vector space, and the other is for dense vector space.

2.3.1 High dimensional sparse vector space

Inverted index-based approaches are efficient top-k retrieval methods for dot product on a high dimensional sparse vector space[1]. These inverted index-based methods have been proposed in the information retrieval field for retrieving the documents that include query terms.

In this task, documents are represented by using a "bag-of-words" model. That is, we consider each term in the documents as a feature. For example, let the following two

[1]Here, we consider Euclidean space that has millions or tens of millions of dimensions as "high dimensional space."

sentences be documents.

- $d^{(1)}$: To be, or not to be, that is the question.
- $d^{(2)}$: The worst is not, So long as we can say, ‘This is the worst.’

These two documents are represented as follows:

	as	be	can	is	long	not	or	question	so	that	the	this	to	we	worst
$d^{(1)}$	0	2	0	1	0	1	1	1	0	1	1	0	2	0	0
$d^{(2)}$	1	0	1	2	1	1	0	0	1	0	1	1	0	1	2

The feature “worst” has the value 2 in $d^{(2)}$ since this term appears twice in the second document. Here, punctuation such as “,” and “.” are omitted, and by ignoring the case, “To” and “to” are regarded as the same term. In the above examples, bag-of-words representations of documents are typically sparse vectors. This is because the number of terms in a document is at most thousands whereas the number of words in multiple documents can be millions.

We represent documents-terms relationships by generalizing the above table as follows.

	t_1	t_2	t_3	t_4	t_5	$\cdots$	t_M
$d^{(1)}$	0	1.35	0.65	0	0	$\cdots$	0
$d^{(2)}$	2.07	0	1.12	0.56	0.71	$\cdots$	0.33
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$d^{(N)}$	0	1.89	0	0.31	0	$\cdots$	0

Here, N is the number of documents and M is the number of terms in the vocabulary. In this table, a term t_4 has the value 0.56 in document $d^{(2)}$. The values are typically derived by using standard IR techniques, such as term frequency and inverse document frequency (tf-idf) [70].

Furthermore, we transpose this table, omit the 0 values, and then make an inverted index.

Term	Posting list
t_1	$2 : 2.07,\ \cdots$
t_2	$1 : 1.35,\ \cdots,\ N : 1.89$
t_3	$1 : 0.65,\ 2 : 1.12,\ \cdots$
t_4	$2 : 0.56,\ \cdots,\ N : 0.31$
t_5	$2 : 0.71,\ \cdots$
$\vdots$	$\vdots$
t_M	$2 : 0.33,\ \cdots$

The fourth row of the table represents that the term t_3 has the value 0.65 in document $d^{(1)}$ and the value 1.12 in $d^{(2)}$.

By using this inverted index, we can efficiently perform a basic Boolean retrieval [70]. For example, if a query is (t_2 AND t_5), which means all documents that includes both t_2 and t_5 are required, we only consider the documents that appear in both posting lists of t_2 and t_5. Similarly, if a query is (t_2 OR t_4), the documents that appear in either posting lists of t_2 or t_5 are retrieved.

Next, we consider a ranked retrieval. Thus, documents are scored by their relevance to a given query. A widely-used scoring function is the dot product between the query and document vectors by using the vector space model:

$$\mathrm{score}(\boldsymbol{q}, \boldsymbol{d}) = \boldsymbol{q}^{\mathrm{T}}\boldsymbol{d} = \sum_{i=1}^{M} q_i d_i,$$

where $\boldsymbol{q}$ and $\boldsymbol{d} \in \mathbb{R}^M$ are vectors of the query and document, respectively. q_i is a value of the i-th element of query vector $\boldsymbol{q}$, which indicates a weight for term t_i in the query. d_i is also the value corresponding to term t_i in the document.

Here, we consider the case where both vectors $\boldsymbol{q}$ and $\boldsymbol{d}$ are sparse because the number of terms is typically at most dozens. The scoring function is represented for simplicity as follows:

$$\mathrm{score}(\boldsymbol{q}, \boldsymbol{d}) = \sum_{1 \leq i \leq M, q_i \neq 0} q_i d_i.$$

Thus, similarly to the OR operator in the Boolean retrieval, we only calculate the scores of documents that appear in either posting lists corresponding to terms where $q_i \neq 0$.

The above approach only evaluates a part of an entire document set. However, as the number of documents or terms in a query becomes large, the computational cost of using this simple approach no longer satisfies our needs. In the real-world settings, retrieval systems need to respond within tens or hundreds of milliseconds to each query. Therefore, more sophisticated methods are required, especially for Web-scale retrieval systems.

Since our objective is retrieval of top-k ranked documents, we further skip the scoring by using some pruning techniques without missing any correct documents. That is, if we are convinced that scores of documents are less than the final k-th largest score during the evaluation, we safely skip the scoring of those documents. The WAND algorithm [19] and its variants [43, 35] are successful methods that use dynamic pruning techniques.

2.3.2 Dense vector space

As described in the previous subsection, in high-dimensional sparse vector space, top-k documents can be efficiently retrieved without missing any by using an inverted index and pruning techniques. On the other hand, in dense vector space, where almost no element values of vectors are zero, almost all documents need to be evaluated. This is because there are not efficient pruning techniques due to the "curse of dimensionality" [45]. Therefore, approximate search techniques have been studied for this case. Approximate search methods focus on improving the tradeoff between retrieval speed and approximation quality[2].

Locality-sensitive hashing (LSH) [45] is an approximate nearest neighbor search method for high-dimensional space[3]. LSH reduces the dimensionality of vectors corresponding to data points for ease of retrieval. This dimensionality reduction aims to map similar vectors to the same "bucket" by using a hash function. Some methods use the hash function independent of the data distribution, whereas others learn the hash function by considering the target data distribution.

Asymmetric LSH (ALSH) [91] is an extended version of LSH for approximately retrieving items that have k-largest dot product values for a given query. LSH variants typically use the same hash function for both queries and items. For example, Euclidean distance between a query and an item is minimized when the two vectors are the same. Cosine distance is also maximized when the two vectors are the same. On the other hand, when the norm of one item vector becomes larger, the value of the dot product becomes larger than when the vectors are the same. Therefore, as long as the same hash function is applied to both queries and items, these conditions are not satisfied. ALSH first applies the different transformations to queries and items and then uses the hash function of LSH.

For ALSH transformation functions, Shrivastava and Li [91] proposed $Q : \mathbb{R}^M \rightarrow \mathbb{R}^{M+m}$ and $P : \mathbb{R}^M \rightarrow \mathbb{R}^{M+m}$ for the query vector $\boldsymbol{q}$ and the item vector $\boldsymbol{d}$, respectively.

$$\begin{aligned} Q(\boldsymbol{q}) &= [\boldsymbol{q}; 1/2; 1/2; \ldots; 1/2]\,, \\ P(\boldsymbol{d}) &= \left[\boldsymbol{d}; \|\boldsymbol{d}\|^2; \|\boldsymbol{d}\|^4; \ldots; \|\boldsymbol{d}\|^{2^m};\right]\,, \end{aligned}$$

where $[;]$ is the concatenation. By using squared values and dot product of transformed vectors:

$$\begin{aligned} \|Q(\boldsymbol{d})\|^2 &= \|\boldsymbol{q}\|^2 + m/4, \\ Q(\boldsymbol{q})^{\mathrm{T}} P(\boldsymbol{d}) &= \boldsymbol{q}^{\mathrm{T}}\boldsymbol{d} + \frac{1}{2}\left(\|\boldsymbol{d}\|^2 + \|\boldsymbol{d}\|^4 + \cdots + \|\boldsymbol{d}\|^{2^m}\right), \\ \|P(\boldsymbol{d})\|^2 &= \|\boldsymbol{d}\|^2 + \|\boldsymbol{d}\|^4 + \|\boldsymbol{d}\|^8 + \cdots + \|\boldsymbol{d}\|^{2^{m+1}}, \end{aligned}$$

Euclidean distance between two transformed vectors is represented as follows:

$$\begin{aligned} \|Q(\boldsymbol{q}) - P(\boldsymbol{d})\|^2 &= \|Q(\boldsymbol{q})\|^2 - 2Q(\boldsymbol{q})^{\mathrm{T}} P(\boldsymbol{d}) + \|P(\boldsymbol{d})\|^2 \\ &= \|\boldsymbol{q}\|^2 + m/4 - 2\boldsymbol{q}^{\mathrm{T}}\boldsymbol{d} + \|\boldsymbol{d}\|^{2^{m+1}}. \end{aligned}$$

If $\|\boldsymbol{d}\|^{2^{m+1}}$ is sufficiently small, that is $\|\boldsymbol{d}\| < 1$ and m is large, $\|Q(\boldsymbol{q}) - P(\boldsymbol{d})\|^2$ inversely correlates to $\boldsymbol{q}^{\mathrm{T}}\boldsymbol{d}$.

[2]Of course, index size is also important, but we here focus on retrieval speed and approximation quality.

[3]Here, we consider Euclidean space that has hundreds or thousands of dimensions as "high dimensional space".

Note that the above transformation functions can be combined with approximate nearest neighbor search methods other than LSH even though they were originally proposed for LSH. There are some methods that transform maximum dot product search problems into maximum cosine or minimum Euclidean distance search problems [9, 92].

Bernhardsson et al. [10] compare the performances of implementations of approximate nearest neighbor search methods on various datasets and distance metrics.

Chapter 3

Inverted Index-based Retrieval System for Contextual Advertising

3.1 Introduction

Online advertising is a key component supporting today's Internet ecosystem and has grown into a multi-billion dollar industry. Many different types of advertising are used: sponsored search advertising, contextual advertising, display advertising, real-time bidding auctions, and more [119]. In this study, we focus on contextual advertising, which consists of short text messages that are usually displayed on third-party Web pages such as news sites or blogs. The advertiser is primarily interested in targeting relevant users, and the publisher is concerned with keeping the user experience pleasant. To satisfy these two objectives, an ad-networking service selects ads that are relevant to the page content and/or the user information. In this study, we focus on increasing the click-through rate (CTR), as this metric directly relates to the user experience, publisher revenue, and advertising effectiveness objectives.

The relevance of an ad to page content is typically determined using a tf-idf score that measures the word overlap between the page content and the ad content. Ads that have high relevance scores to the Web page are selected. This task is therefore regarded as a similarity search in high-dimensional sparse vector space. Inverted index-based approaches can retrieve the ads in very short response time by utilizing the sparsity of the vector space. This is an effective technique when the expected word overlap rate is high, but it falters when the vocabulary used on the page is different from the vocabulary used in the ad. For example, an ad for "SIM-free smartphones" would be relevent to a Web page comparing Mobile Virtual Network Operator (MVNO) services, but the word overlap might not be very high. Another example could be an ad for "HTC One" and a page about "New Nexus 7."

To remedy this problem, some previous studies have used a semantic taxonomy in the matching function [18] or introduced a page-ad probability model with hidden classes [86]. However, in these approaches, it is necessary to expand the above inverted index-based ad retrieval system or build a new index to handle the categories or classes in order to respond to each ad request in a very restricted time. In addition, a review of the number and hierarchical structure of categories or a re-creation of clusters is periodically required in the operation of the ad serving system, and these tasks are not always easy to perform.

To overcome the above problems, we have developed an approach that calculates a matching score between two term vectors using an inverted index and does not require modification of an ordinary ad retrieval system. In other words, this approach translates ad request information into the textual space of ads. With this translation table, the feature vector of ad requests is transformed into the input term vector of the ad retrieval system. The process is illustrated in Figure 3.1.

This translation table will become very large because the two spaces are typically quite large. We can efficiently learn the translation table from past click data with low-rank approximation of the matrix. However, even if the learning can be done efficiently, the transformed term vector must be made sparse because the performance of the inverted index-based ad retrieval system progressively worsens in accordance with the number of nonzero values in the input vector [35, 43, 96]. Therefore, we first select ad features related to each query feature in accordance with our metric based on the past CTR, and learn the translation matrix. In other words, we choose a subset of the matrix elements and learn only the weights with past click logs. By using these procedures, the ads that have high CTR are efficiently retrieved without the performance of the ad retrieval system degrading. Our main contributions are as follows.

- We propose a method of translating contextual information into the textual features of ads by using past click data.
- Our approach is easy to implement and maintain because there is no need to modify the existing ad retrieval system.
- We evaluated our approach using a real-world dataset from an ad network and obtained better results than existing methods.
- We applied our approach to a real ad serving system and achieved an improvement over the existing production system.

The rest of this study is organized as follows. Section 3.2 provides a general overview of the contextual advertising system. Section 3.3 presents our method of translating query information into ad term vector space. Section 3.4 details the experimental setup and results. We conclude our paper in Section 3.5 by summarizing our findings.

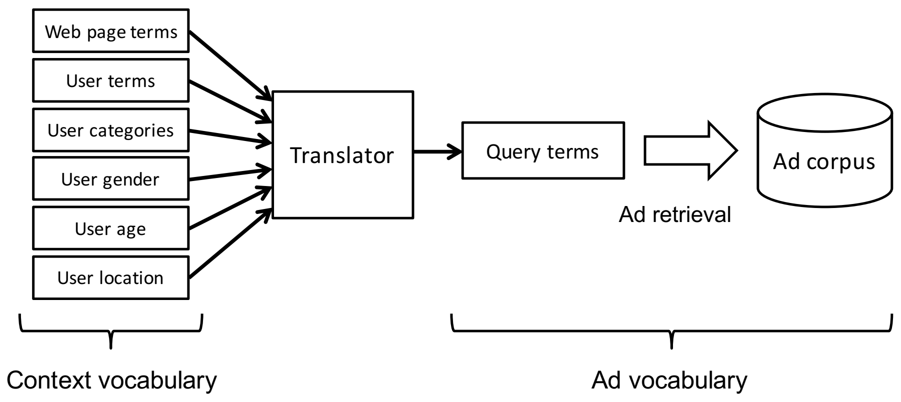


Figure 3.1: Overview of our proposed approach. Our method is a translation of request features into the input term vector of an ad retrieval system.

3.2 Overview and Related Works

This section provides a general overview of contextual advertising and related studies.

3.2.1 Overview of Contextual Advertising

There are four players in contextual advertising: the publisher, the advertiser, the ad network, and the user. The publisher owns Web pages and reserves some space on these pages for ads. An advertisers place ads in an ad network. Each time a user visits the Web page, the publisher requests a set of ads to be displayed from the ad network. For the request, the ad network chooses appropriate ads from candidates and supplies them to the publisher. Then, the publisher displays the returned ads on the reserved space and the user sees them. If it takes too long for the ad network to retrieve ads, the user's experience is spoiled since some or all of the Web page is not displayed. Thus, the ad network responds to each ad request from the publisher in a really short time, typically tens or hundreds of milliseconds. In the common pay-per-click model, which is the prevalent pricing model for contextual advertising, the advertiser pays the Web publisher and the ad network a fee only if a user clicks on their advertisement and visits their Web site. Thus, the expected revenue from displaying each ad is a function of both the bid price and the CTR. The bid price is the cost that the advertiser agrees to pay per click, so the advertising system already knows this. In contrast, the CTR for each ad can vary significantly depending on a variety of factors ranging from the Web page to the user. Consequently, one of the main problems in contextual advertising is determining how to accurately predict CTR and efficiently retrieve ads with a high CTR from an ad corpus in the restricted response time.

3.2.2 Related Works

Some previous works have focused on developing methods to match ads to pages, since ads that are related to the page content are more likely than generic ads to provide a better user experience. That in turn increases the probability of users clicking on the ads. In these studies, the problem of matching ads with pages is transformed into a similarity search in a vector space. The relevance of an ad to page content is indicated by a tf-idf score that measures the word overlap between the page content and the ad content. Chakrabarti et al. [25] and Karimzadehgan et al. [59] introduced methods to learn the weights of each word in a page and an ad using HTML tags and ad sections. They use an inverted index to efficiently retrieve top-K items from ad corpora.

Althogh both pages and ads are mapped to the same space, there is a discrepancy (impedance mismatch) between the vocabulary used in the ads and on the pages. Various approaches have been proposed to overcome this problem. Broder et al. [18] used a 6000-node semantic taxonomy in the matching function between pages and ads. In addition, Ratnaparkhi [86] introduced a page-ad probability model in which semantic relationships between page terms and ad terms are modeled with hidden classes. Yih and Jiang [116] proposed an approach to map the original term vectors to a "concept space" so that semantically close words would be captured by the same concept. Wang et al. [104] formulated and tackled the problem of relevance learning for online targeting in heterogeneous social networks. They inferred user interests and ad concepts from heterogeneous sources and links, and developed a user-ad relevance feature on the basis of weighted matching between any pair of concept classes. Murdock et al. [77] applied machine translation techniques to improve the matching between pages and ads.

Joshi et al. [57] presented a method to leverage user information including a user's demographic information (e.g., age, gender, and location) and behavioral information (e.g., the user's recent search history, page visits, and ad clicks) in a content match advertising setting. They mapped the non-textual user features to the textual space of ads.

In the cases where ad relevance cannot easily be gleaned from the page text alone, the "clickable terms" approach has been proposed [50]. This approach involves matching a Web site directly with a set of ad side terms, independent of the page content.

Another line of research attempts to predict the CTR of ads. These studies are related to not only contextual advertising but also to sponsored search advertising because both typically use the pay-per-click model. Predictions of CTR for ads are generally based on a statistical model trained by using past click data. Examples of such models include logistic regression [26, 27, 73], probit regression [47], boosted trees [34, 103], and factorization machines [58, 80]. The accuracy of the model depends greatly on the design of the features. Cheng and Cantú-Paz [26] presented a framework for the personalization of click models. They developed user-specific and demographic-based features that reflect the click behavior of individuals and groups. These features are based on observations of

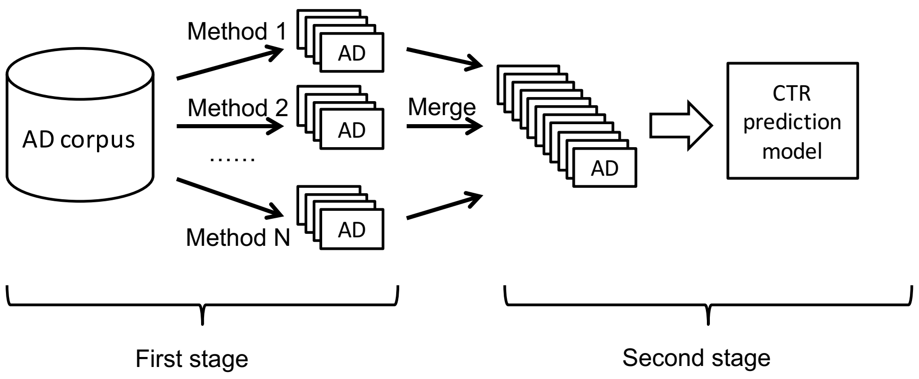


Figure 3.2: Two-stage approach in the ad serving system. Ads are retrieved by multiple methods in the first stage. The ads are merged and passed on to the second stage for CTR prediction.

the search and click behaviors of a large number of users of a commercial search engine. Other recent works [2, 64, 88] have proposed models to estimate conversion rates (CVR).

3.2.3 Our Ad Serving System

Although machine-learned CTR prediction models are more accurate, as described in Section 3.2.2, evaluating all the items in an ad corpus is too time-consuming if we want to ensure that the Web page loads quickly [3]. Our ad serving system therefore adopts a two-stage approach similar to some other studies [3, 25]. The first stage retrieves top-K items from an ad corpus using an inverted index. The second stage selects the desired top-k using brute force CTR prediction on the K retrieved ads ($k \ll K$). In the first stage, ads are independently retrieved by multiple methods in parallel. The retrieved ads in the first stage are merged and passed on to the second stage for CTR prediction. Ads are ranked and displayed on the basis of the predicted CTR and the bid price. Thus, the objective of the first stage is to retrieve the ads that have high estimated CTR (and/or high bid price) and pass on them to the second stage. This process is illustrated in Figure 3.2. We refer to the "ad retrieval system" as a part of the first stage and focus on this part. We used a heavily tuned search engine that uses the WAND algorithm [19, 43, 35].

In this study, we propose a method to efficiently retrieve ads with high CTR using the search engine without any changes, even building a new index. To simply measure relevance between contextual information and ads, distributed representations obtained from trained neural networks may be applicable [51, 32]. However, since such distributed representations are typically low-dimensional dense vectors, an ad retrieval system that combines inverted index with the WAND algorithm can not retrieve ads efficiently. Off course, some methods [67] have been proposed to efficiently retrieve ads for such kinds of vectors, but these methods are not easy to implement in a large-scale real-world system at a satisfactory level of quality. Therefore, in this study, we focus on a method that generates high-dimensional sparse vectors and utilizes the existing inverted index-based retrieval system.

3.3 Methods

In this section, we first define the matching function between a query and an ad and then describe our approach to mapping the query information to the textual space of ads.

3.3.1 Matching Function

We define a query feature vector as $\boldsymbol{q} = (q_1, \ldots, q_{D_q})^{\mathrm{T}}$ and an ad feature vector as $\boldsymbol{a} = (a_1, \ldots, a_{D_a})^{\mathrm{T}}$. $\boldsymbol{q}$ is a query feature vector of an ad request that includes Web page and user information. We also define a general form of the matching function for $\boldsymbol{q}$ and $\boldsymbol{a}$ using translation matrix $\boldsymbol{W} = [w_{ij}]_{D_q \times D_a}$, as:

$$\mathrm{mscore}(\boldsymbol{q}, \boldsymbol{a}) = \boldsymbol{q}^{\mathrm{T}} \boldsymbol{W} \boldsymbol{a} = \sum_{i=1}^{D_q} \sum_{j=1}^{D_a} w_{ij} q_i a_j, \tag{3.1}$$

where D_q is a dimension of query feature space and D_a is a dimension of ad feature space.

Equation (3.1) is a general form of the matching function. If D_a is the same as D_q and $\boldsymbol{W}$ is the identity matrix $\boldsymbol{I}$, score$(\boldsymbol{q}, \boldsymbol{a})$ turns out to be a simple dot product:

$$\mathrm{mscore}(\boldsymbol{q}, \boldsymbol{a}) = \boldsymbol{q}^{\mathrm{T}} \boldsymbol{I} \boldsymbol{a} = \sum_{i=1}^{D_q} q_i a_i.$$

Of course, the cosine similarity can be calculated if $\boldsymbol{q}$ and $\boldsymbol{a}$ are normalized. Also, in general cases, by multiplying the query feature vector $\boldsymbol{q}$ by the translation matrix $\boldsymbol{W}$ and using $\boldsymbol{q}' = \boldsymbol{W}^{\mathrm{T}} \boldsymbol{q}$ as an input vector, we calculate the score $\mathrm{mscore}(\boldsymbol{q}, \boldsymbol{a}) = \boldsymbol{q}^{\mathrm{T}} \boldsymbol{W} \boldsymbol{a} = \boldsymbol{q}'^{\mathrm{T}} \boldsymbol{a}$ with an ordinary inverted-index-based ad retrieval system.

When categories or classes are used, translation matrix $\boldsymbol{W}$ is decomposed as:

$$\boldsymbol{W} = \boldsymbol{\Gamma}_q^{\mathrm{T}} \boldsymbol{W}_c \boldsymbol{\Gamma}_a,$$

where $\boldsymbol{\Gamma}_q \in \mathbb{R}^K \times \mathbb{R}^{D_q}$, $\boldsymbol{\Gamma}_a \in \mathbb{R}^K \times \mathbb{R}^{D_a}$, $\boldsymbol{W}_c \in \mathbb{R}^K \times \mathbb{R}^K$, and K is the number of categories or classes. Here, $\boldsymbol{\Gamma}_q$ and $\boldsymbol{\Gamma}_a$ are the respective matrices that convert query feature vector $\boldsymbol{q}$ and ad feature vector $\boldsymbol{a}$ into categories or classes vector $\boldsymbol{q}_c$ and $\boldsymbol{a}_c$. Thus, the matching function in this case is expressed as:

$$\begin{aligned} \mathrm{mscore}(\boldsymbol{q}, \boldsymbol{a}) &= (\boldsymbol{\Gamma}_q \boldsymbol{q})^{\mathrm{T}} \boldsymbol{W}_c (\boldsymbol{\Gamma}_a \boldsymbol{a}) \\ &= \boldsymbol{q}_c^{\mathrm{T}} \boldsymbol{W}_c \boldsymbol{a}_c, \end{aligned}$$

where $\boldsymbol{q}_c = \boldsymbol{\Gamma}_q \boldsymbol{q}$ and $\boldsymbol{a}_c = \boldsymbol{\Gamma}_a \boldsymbol{a}$. This decomposition can also be viewed as a matrix factorization or low-rank approximation.

3.3.2 Learning a Translation Matrix

As described in Section 3.1, we need to learn the translation matrix $\boldsymbol{W}$ efficiently and make the transformed term vector sparse for efficient ad retrieval. For this purpose, adding the L_1 regularization term of $\boldsymbol{W}^{\mathrm{T}}\boldsymbol{q}$ to the objective function, we may directly optimize it by using the alternating direction method of multipliers (ADMM) [95]. However, the objective function including this regularization term is relatively complex to optimize. In this study, we propose another approach, in which we can directly control the sparseness of the matrix by considering the performance of the ad retrieval system. Instead of applying L_1 regularization, we first select a subset of the matrix elements and then learn the corresponding w_{ij}.

We calculate the following score m_{ij} for each pair of q_i and a_j presented in the training data:

$$m_{ij} = \frac{\mathrm{CTR}(q_i, a_j)}{\max(\mathrm{CTR}(q_i), \mathrm{CTR}(a_j))},$$

where $\mathrm{CTR}(q_i)$ denotes the CTR when the query feature vector includes q_i and $\mathrm{CTR}(a_j)$ denotes the CTR of ads that include feature a_j. Similarly, $\mathrm{CTR}(q_i, a_j)$ represents the CTR of ads that include feature a_j when the query feature vector includes q_i. A large m_{ij} value means that ads that include feature a_j are more likely to be clicked when the query feature vector includes q_i. We conducted preliminary experiments by replacing the denominator part of m_{ij} with some variants, such as the product of the two CTRs and square root of the product. We observed that this max form empirically achieved better results. A set of the pairs that has a larger m_{ij} is selected:

$$P = \{(i, j) \mid m_{ij} > T\},$$

where T is a thresholding hyper-parameter. The number of non-zero elements in $\boldsymbol{W}$ decreases as a function of T. We use P and replace the matching function (3.1) as follows:

$$\mathrm{mscore}(\boldsymbol{q}, \boldsymbol{a}) = \sum_{(i,j) \in P} w_{ij} q_i a_j.$$

If we perform an ordinary feature selection using mutual information or L_1 regularization, we obtain features that lead to negative w_{ij}'s as well as positive w_{ij}'s. However, we retrieve top-K ads that have larger $\mathrm{mscore}(\boldsymbol{q}, \boldsymbol{a})$ from an ad corpus as described in Section 3.2.3. We cannot retrieve ads that have higher $\mathrm{mscore}(\boldsymbol{q}, \boldsymbol{a})$ even if we use the negative w_{ij}'s. Thus, we use the above approach in the hope of selecting the features to lead to the positive w_{ij}'s.

We learn the above w_{ij} by using past click data. A score proportional to the click-through rate (CTR) for $\boldsymbol{q}$ and $\boldsymbol{a}$ is therefore defined as having the following linear form:

$$\begin{aligned}
\text{score}(\boldsymbol{q}, \boldsymbol{a}) &= \text{mscore}(\boldsymbol{q}, \boldsymbol{a}) + \text{bscore}(\boldsymbol{q}, \boldsymbol{a}) \\
&= \sum_{(i,j) \in P} w_{ij} q_i a_j + \boldsymbol{w}_{\text{basic}}^{\mathrm{T}} \boldsymbol{x}_{\text{basic}} \\
&= \boldsymbol{w}_{\text{match}}^{\mathrm{T}} \boldsymbol{x}_{\text{match}} + \boldsymbol{w}_{\text{basic}}^{\mathrm{T}} \boldsymbol{x}_{\text{basic}} \\
&= \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}, \qquad (3.2)
\end{aligned}$$

where $\text{bscore}(\boldsymbol{q}, \boldsymbol{a})$ is a basic score as $\text{bscore}(\boldsymbol{q}, \boldsymbol{a}) = \boldsymbol{w}_{\text{basic}}^{\mathrm{T}} \boldsymbol{x}_{\text{basic}}$. $\boldsymbol{x}_{\text{basic}}$ is a feature vector that includes features such as the ad's own clickability. $\boldsymbol{w}_{\text{basic}}$ is a weight vector corresponding to $\boldsymbol{x}_{\text{basic}}$. For $(i, j) \in P$, $\boldsymbol{w}_{\text{match}}$ and $\boldsymbol{x}_{\text{match}}$ are defined as $\boldsymbol{w}_{\text{match}} = [\cdots, w_{ij}, \cdots]^{\mathrm{T}}$ and $\boldsymbol{x}_{\text{match}} = [\cdots, q_i a_j, \cdots]^{\mathrm{T}}$. $\boldsymbol{w} = \left[\boldsymbol{w}_{\text{match}}^{\mathrm{T}}, \boldsymbol{w}_{\text{basic}}^{\mathrm{T}}\right]^{\mathrm{T}}$ and $\boldsymbol{x} = \left[\boldsymbol{x}_{\text{match}}^{\mathrm{T}}, \boldsymbol{x}_{\text{basic}}^{\mathrm{T}}\right]^{\mathrm{T}}$ are concatenated vectors. The reason for adding $\text{bscore}(\boldsymbol{q}, \boldsymbol{a})$ to $\text{mscore}(\boldsymbol{q}, \boldsymbol{a})$ is that the score proportional to CTR consists of not only a matching score between the query and ad but also other factors such as the ad's own clickability and display position on the Web page.

As described in Section 3.2.3, our focus is improving the first stage of the two-staged ad serving system. We learn w_{ij} with click logs instead of simply using m_{ij}, as we expect that first retrieving top-K ads on the basis of a score approximating the second stage score leads to better top-k results at the second stage.

We then describe how to utilize click logs to learn weights. In the contextual advertising setting, a Web publisher typically requests some ads simultaneously because more than one ad is displayed on a page at the same time. In other words, one ad request r in the logs includes $N^{(r)}$ impressions of ads:

$$(\boldsymbol{q}^{(r)}, \boldsymbol{a}_1^{(r)}, y_1^{(r)}), (\boldsymbol{q}^{(r)}, \boldsymbol{a}_2^{(r)}, y_2^{(r)}), \ldots, (\boldsymbol{q}^{(r)}, \boldsymbol{a}_{N^{(r)}}^{(r)}, y_{N^{(r)}}^{(r)}).$$

Each impression of an ad consists of a tuple $(\boldsymbol{q}^{(r)}, \boldsymbol{a}_i^{(r)}, y_i^{(r)})$. Here, $\boldsymbol{q}^{(r)}$ represents the query feature vector of the request r and $\boldsymbol{a}_i^{(r)}$ represents the ad feature vector of the i-th impression in request r. The output variable $y_i^{(r)} = 1$ if a user clickes the ad and $y_i^{(r)} = 0$ if not.

As described in our previous work [101], we focus on a kind of ad request and apply the learning-to-rank approach to learn the weights.

There are two kinds of ad requests in data R:

$$\begin{aligned}
R^+ &= \{r \mid \exists i (y_i^{(r)} = 1)\}, \\
R^- &= \{r \mid \forall i (y_i^{(r)} = 0)\}.
\end{aligned}$$

R^+ denotes a set of ad requests that includes at least one clicked impression; hence we refer to R^+ as **clicked requests**. R^- is called **non-clicked requests** since no ad requests in R^- include clicked impressions. Of course, $R = R^+ \cup R^-$ and $R^+ \cap R^- = \emptyset$. Examples of these two kinds of ad request are shown in Figure 3.3.

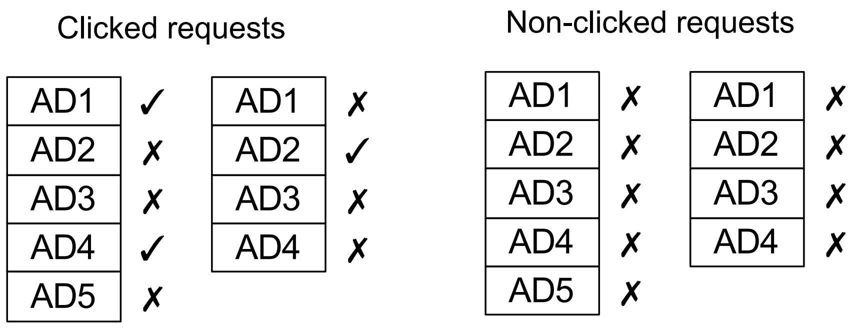


Figure 3.3: Examples of clicked requests and non-clicked requests. Ticks denote clicked impressions, and crosses represent non-clicked impressions.

We regard ad impressions in an ad request as documents related to a query in an information retrieval context and apply the learning-to-rank approach to learn the weights. We make pairwise preferences from each clicked request $r \in R^+$.

$$\{(\boldsymbol{a}_i^{(r)}, \boldsymbol{a}_j^{(r)}) \mid \forall i, j (y_i^{(r)} = 1 \land y_j^{(r)} = 0)\}.$$

This process is illustrated in Figure 3.4.

The preference $(\boldsymbol{a}_i^{(r)}, \boldsymbol{a}_j^{(r)})$ indicates that a score proportional to the CTR of $\boldsymbol{a}_i^{(r)}$ for $\boldsymbol{q}^{(r)}$ is expected to be higher than that of $\boldsymbol{a}_j^{(r)}$. We represent the above preference using $\text{score}(\boldsymbol{q}, \boldsymbol{a})$ and transform using (3.2) as follows:

$$\begin{aligned}
&\text{score}(\boldsymbol{q}^{(r)}, \boldsymbol{a}_i^{(r)}) > \text{score}(\boldsymbol{q}^{(r)}, \boldsymbol{a}_j^{(r)}) \\
&\Leftrightarrow \text{score}(\boldsymbol{q}^{(r)}, \boldsymbol{a}_i^{(r)}) - \text{score}(\boldsymbol{q}^{(r)}, \boldsymbol{a}_j^{(r)}) > 0 \\
&\Leftrightarrow \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i^{(r)} - \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_j^{(r)} > 0 \\
&\Leftrightarrow \boldsymbol{w}^{\mathrm{T}} (\boldsymbol{x}_i^{(r)} - \boldsymbol{x}_j^{(r)}) > 0.
\end{aligned}$$

Using the squared hinge loss, we define a pairwise loss function $L(\boldsymbol{w})$ like RankSVM [56] as follows:

$$L(\boldsymbol{w}) = \sum_{r \in R^+} \sum_{i : y_i^{(r)} = 1} \sum_{j : y_j^{(r)} = 0} \max(0, 1 - \boldsymbol{w}^{\mathrm{T}} (\boldsymbol{x}_i^{(r)} - \boldsymbol{x}_j^{(r)}))^2. \tag{3.3}$$

We add a regularization term and seek the weight vector $\hat{\boldsymbol{w}}$ that minimizes the following optimization problem:

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} \frac{1}{2} \|\boldsymbol{w}\|_2^2 + C \cdot L(\boldsymbol{w}),$$

where $C \geq 0$ is a penalty parameter. The translation matrix $\boldsymbol{W}$ is restored from the weight vector $\hat{\boldsymbol{w}}_{\text{match}}$, where $\hat{\boldsymbol{w}} = \left[\hat{\boldsymbol{w}}_{\text{match}}^{\mathrm{T}}, \hat{\boldsymbol{w}}_{\text{basic}}^{\mathrm{T}}\right]^{\mathrm{T}}$. Note that we set $w_{ij} = 0$ for $(i, j) \notin P$.

We conducted preliminary experiments using hinge loss and logistic loss in addition to the above squared hinge loss. We decided to use the squared hinge loss as it was found to have a favorable balance between accuracy and training time.

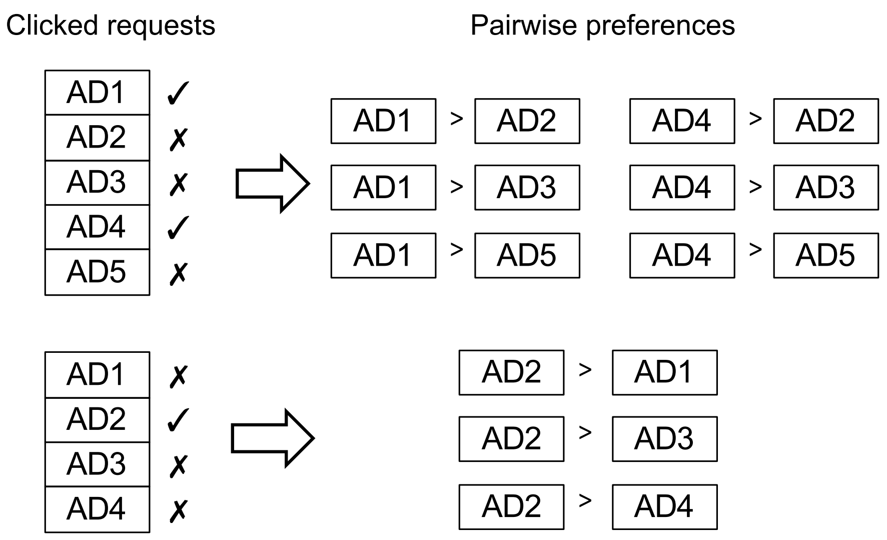


Figure 3.4: Making pairwise preferences from clicked requests.

3.3.3 Retrieval from AD Corpus and Implementation

By using the learned matrix $\mathbf{W}$, the query feature vector $\boldsymbol{q}$ is transformed into the input term vector of the ad retrieval system for each ad request.

$$\boldsymbol{q}_{\text{input}} = \boldsymbol{W}^{\mathrm{T}}\boldsymbol{q}.$$

Our proposed method only require this transformation. Thus it is easy to implement and maintain because there is no need to modify the existing inverted index or add new index.

This input term vector includes some non-zero values, which are proportional to the number of non-zero values in the query feature vector. However, the performance of the ad retrieval system declines in accordance with the number of non-zero values in the input vector [43]. Therefore, we need to limit the number of these values with hyper-parameter M:

$$||\boldsymbol{q}_{\text{input}}||_0 \leq M.$$

$||\boldsymbol{q}_{\text{input}}||_0$ is the L_0-norm of $\boldsymbol{q}_{\text{input}}$, which is the number of its non-zero elements. We simply choose top-M elements, which have larger values.

The computational cost of the above vector transformation using the matrix $\boldsymbol{W}$ is much cheaper than that of retrieval using the transformed vector as input. Therefore, the response performance of the ad retrieval system mainly depends on M, which is the number of non-zero elements in the input vector. By changing M, the response time becomes faster than linear (superlinear) [96, 43, 35]. The sparsity-adjustment threshold T, introduced in Section 3.3.2, does not affect the response performance very much. However, when we use a small value of M, we must increase T. Because a large value of T induces the matrix $\boldsymbol{W}$ to be sparse, the number of non-zero elements in $\boldsymbol{W}^{\mathrm{T}}\boldsymbol{q}$ and the number of elements to be limited by M becomes small. Thus, when implementing a real ad serving system, we first determine the value of M in order to satisfy the requirement of response time. Then, we choose the optimal T value on the basis of the M value.

3.4 Experiments

This section describes offline and online evaluations. In the offline evaluation, we compared the proposed method with existing ones by using the past click logs. In the online evaluation, we applied our approach to a real ad serving system and observed changes in users' responses. Due to business confidentiality, we report only relative performance when showing experimental results.

3.4.1 Offline Evaluation

Our objective in this study is to develop a method that successfully improves ads' CTRs without big changes to the existing ad retrieval system. To evaluate the improvement, we apply our approach to a real ad serving system and compare its online performance with those of other systems by conducting A/B testing. However, the implementation cost is not negligible, and there are risks of decreasing revenue and user satisfaction by conducting online A/B testing. Therefore, we first conduct an offline evaluation by using past click logs, which is relatively low cost, for estimating the improvement over the existing methods and the effect of changing values of hyper-parameters. Note that we do not evaluate response time of the proposed method in this offline setting since the performance is guaranteed to fulfill the required condition in the later online evaluation described in Section 3.4.2.

In this section, we first describe experimental settings such as datasets, features, models, and evaluation metrics. Next, we compare our approach with existing methods and present model performances when changing the hyper-parameters T and M.

Datasets and Features

We compare the models using data sampled from an ad network for a period of eight weeks. Data from the first six weeks are used as a training set, data from the fifth week are used as a validation set, and data from the last week are treated as a testing set. This ad network is for the Japanese market[1], so all ads and pages are written in Japanese, with a few exceptions.

As described in Section 3.3.2, each sample of the datasets is an impression of an ad and consists of a tuple $(\boldsymbol{q}^{(r)}, \boldsymbol{a}_i^{(r)}, y_i^{(r)})$. The output variable $y_i^{(r)} = 1$ if a user clicks the ad and 0 if not. The query features $\boldsymbol{q}^{(r)}$ include Web page and user information. The Web page

[1] `http://promotionalads.yahoo.co.jp/service/ydn/index.html`

Table 3.1: Summary of features

Feature type	Source	Details
Query features $\boldsymbol{q}$	Web page	Terms extracted from Web page
	User	Terms extracted from behavioral events, categories based on behavioral events, gender, age, location
Ad features $\boldsymbol{a}$	Ad	Tf-idf weighted terms
Basic features $\boldsymbol{x}_{basic}$	Past click log	Historical CTR of ad and advertiser, display position on the Web page

features are extracted terms. These terms are scored on the basis of their position on the page and HTML tags. Some terms are chosen by the score. The user features are terms and categories in which the user is interested, as well as gender, age, and location. The user's gender falls into three classes: male, female, and unknown. Similarly, the user's age is categorized into 13 groups. As in the study by Aly et al. [5], these terms and categories are extracted from user behavior events such as page visits, search queries, and ad clicks. These categories are similar to the hierarchical taxonomy in the work of Broder et al. [18]. There are about 900 categories. We simply use textual features as the ad features $\boldsymbol{a}_i^{(r)}$, which are tf-idf weighted terms based on the title and description in this study. These features are summarized in Table 3.1.

The basic features $\boldsymbol{x}_{\text{basic}}$ described in Section 3.3.2 include the display position on the Web page and the historical CTR of the ad and advertiser. As described above, since CTR can be changed by user-related features, it is natural to include these features into $\boldsymbol{x}_{\text{basic}}$. However, ad-independent features do not affect the order relation of scores in an ad request because these features change scores by the same amount for all ads in the request. Further, the loss value is not changed because we use the pairwise loss function 3.3 and the ad-independent features are disappeared when taking into account the difference between two feature vectors $(\boldsymbol{x}_i^{(r)} - \boldsymbol{x}_j^{(r)})$. Thus, ad-related features are just included in $\boldsymbol{x}_{\text{basic}}$. These features are also summarized in Table 3.1.

We chose eight diverse Web sites, including news, blogs, question-and-answer, finance, sports, weather, and travel sites. The models we evaluate are constructed with respect to each Web site, since the Web pages and the users that visit them are different. The data statistics for each Web site are summarized in Table 3.2. The number of clicked requests $|R^+|$ and the average number of impressions per clicked request $\overline{N^{(r)}}$ changed over the six weeks, due to seasonal trends, changes in the budget of the advertisers, and actions carried out by publishers to achieve sales targets. $\overline{\#clicks}$, which is the average number of clicked impressions per clicked request, is approximately 1.

Table 3.2: Data statistics for Web sites used in evaluation.

Web site	Type	$\lvert R^+ \rvert$	$\overline{N^{(r)}}$	$\overline{\#\text{clicks}}$
A	training	711,539	6.97	1.03
	validation	142,649	7.03	1.03
	testing	119,464	7.11	1.02
B	training	2,676,577	4.99	1.03
	validation	429,760	4.99	1.03
	testing	356,578	4.99	1.01
C	training	1,648,118	4.64	1.02
	validation	137,646	3.09	1.02
	testing	134,168	3.15	1.01
D	training	919,870	4.11	1.01
	validation	92,547	4.05	1.01
	testing	81,077	4.07	1.00
E	training	905,842	4.07	1.01
	validation	217,165	4.94	1.01
	testing	169,627	4.94	1.01
F	training	153,849	5.00	1.04
	validation	23,836	5.00	1.04
	testing	17,879	5.00	1.02
G	training	297,814	8.13	1.03
	validation	53,306	8.10	1.02
	testing	42,098	8.05	1.02
H	training	4,644,350	4.48	1.02
	validation	780,037	4.54	1.02
	testing	498,407	4.15	1.01

Existing Methods and Evaluation Metric

We compared the proposed method with three existing production methods: *existing 1*, *2*, and *3*. These methods also use the features described in Section 3.4.1 and Table 3.1. *Existing 1* utilizes only terms extracted from a Web page and weighted via an optimization procedure similar to the method of Karimzadehgan et al. [59] for ad retrieval. *Existing 2* uses terms extracted from user's behavioral events, such as Web page visits, search keywords, and ad clicks. *Existing 3* uses category information obtained from user's behavioral events for ad retrieval. For *existing 2* and *3*, we extracted terms and categories similarly to the method of Aly et al. [5].

As noted in 3.2.3, in terms of application to a real ad serving system, we focus on the method that generates high-dimensional sparse vectors and utilizes the existing retrieval system with the inverted index and WAND algorithm. The above three existing methods satisfy this condition by representing both the query and ad as high-dimensional sparse vectors. In contrast, vectors obtained via training neural networks are typically low-dimensional and dense. Therefore, we excluded these methods in the comparison because they do not satisfy the condition.

In the comparison, we use the following $\mathrm{score}_{\mathrm{existing}}(\boldsymbol{q}, \boldsymbol{a})$ instead of $\mathrm{score}(\boldsymbol{q}, \boldsymbol{a})$ in Equation (3.2):

$$\mathrm{score}_{\mathrm{existing}}(\boldsymbol{q}, \boldsymbol{a}) = w \cdot \mathrm{escore}(\boldsymbol{q}, \boldsymbol{a}) + \mathrm{bscore}(\boldsymbol{q}, \boldsymbol{a}),$$

where $\mathrm{escore}(\boldsymbol{q}, \boldsymbol{a})$ is a matching score calculated by an existing method. Each existing method has a different $\mathrm{escore}(\boldsymbol{q}, \boldsymbol{a})$.

As described in Section 3.3.3, we need to limit the number of query terms because of the performance of the ad retrieval system. In the experiment, we carried out our evaluation by changing the value of M. Thus, we rewrite the scoring function for the prediction as:

$$t\text{-score}(\boldsymbol{q}, \boldsymbol{a}) = t\text{-mscore}(\boldsymbol{q}, \boldsymbol{a}) + \mathrm{bscore}(\boldsymbol{q}, \boldsymbol{a}), \tag{3.4}$$

where t-mscore$(\boldsymbol{q}, \boldsymbol{a})$ is a matching function using truncated $\boldsymbol{q}_{\mathrm{input}} = \boldsymbol{W}^{\mathrm{T}}\boldsymbol{q}$. We change M and truncate the query term vector $\boldsymbol{q}_{\mathrm{input}}$ during the evaluation, not during training. This means the same translation matrix $\boldsymbol{W}$ is used. Note that this evaluation does not reflect the actual online setting very well when the value of M is limited. We use t-score$(\boldsymbol{q}, \boldsymbol{a})$ in the offline evaluation, although ads are retrieved by t-mscore$(\boldsymbol{q}, \boldsymbol{a})$ from an ad corpus in the actual online setting. Because of $\mathrm{bscore}(\boldsymbol{q}, \boldsymbol{a})$, the order by t-mscore$(\boldsymbol{q}, \boldsymbol{a})$ and t-score$(\boldsymbol{q}, \boldsymbol{a})$ is not the same for some ad requests in the testing set.

We evaluated the performance of the model by using mean average precision (MAP) [70]:

$$\begin{aligned}
\mathrm{MAP} &= \frac{1}{|R^+|} \sum_{r \in R^+} \mathrm{AP}^{(r)}, \\
\mathrm{AP}^{(r)} &= \frac{\sum_{k=1}^{N^{(r)}} \mathrm{P}_k^{(r)} y_{\pi^{(r)}(k)}^{(r)}}{\sum_{k=1}^{N^{(r)}} y_{\pi^{(r)}(k)}^{(r)}}, \\
\mathrm{P}_k^{(r)} &= \frac{\sum_{l=1}^{k} y_{\pi^{(r)}(l)}^{(r)}}{k},
\end{aligned}$$

where $\mathrm{AP}^{(r)}$ is the averaged precision over all relevant documents for request r, and $P_k^{(r)}$ is the precision up to rank position k. Here, $\pi^{(r)}(k) = i$ means that the i-th impression ranks in the k-th position by the predicted score $\mathrm{score}(\boldsymbol{q}^{(r)}, \boldsymbol{a}_j^{(r)})$. We normalize the scores of the method by a basic model that uses only $\mathrm{bscore}(\boldsymbol{q}, \boldsymbol{a})$ during both the training and evaluation. All values of metrics in this study are transformed by

$$\Delta\mathrm{MAP} = \left(\frac{\mathrm{MAP}}{\mathrm{MAP}_{\mathrm{basic}}} - 1 \right) \times 100.$$

Note that this $\mathrm{MAP}_{\mathrm{basic}}$ is reasonably high because the display position included in $\boldsymbol{x}_{\mathrm{basic}}$ is a very beneficial feature. In this evaluation setting, we just make predictions for tens of

Table 3.3: Experimental results. Values are ΔMAP. The **bold** elements indicate the best performance of the methods.

Method	Web site							
	A	B	C	D	E	F	G	H
Existing 1	+1.90%	+0.01%	–0.04%	+0.01%	+0.87%	+1.22%	+0.31%	+0.00%
Existing 2	+0.85%	+2.71%	+0.53%	+0.24%	+0.85%	+0.59%	+0.98%	+1.00%
Existing 3	+0.04%	+0.37%	+0.04%	+0.03%	+0.02%	+0.38%	+0.01%	+0.11%
$T = 0.20$	+0.60%	+6.18%	+1.42%	+0.50%	+1.70%	+1.66%	+1.73%	+2.60%
$T = 0.15$	+1.16%	+6.60%	+1.50%	+0.60%	+1.79%	+2.14%	+1.86%	+2.90%
$T = 0.10$	+2.97%	+6.83%	**+1.55%**	+0.64%	**+1.84%**	+2.32%	+2.00%	+3.13%
$T = 0.05$	**+3.56%**	**+6.93%**	+1.54%	**+0.68%**	+1.81%	**+2.56%**	**+2.12%**	**+3.24%**

ads that are selected by the existing retrieval methods, although we need to retrieve ads from hundreds of thousands to millions candidates in the actual ad serving setting. This is also a limitation of offline evaluation with past click logs.

Results

We first evaluated our approach when changing the thresholding hyper-parameter T. As described in Section 3.3.2, the number of non-zero elements in $\boldsymbol{W}$ decreases as a function of T. This means that model performance is expected to improve with decreasing T. In this setting, M is not limited during the evaluation. The experimental results are summarized in Table 3.3. The **bold** elements indicate the best performance of the methods. Our proposed method achieved an improvement over the existing methods. As expected, MAPs improve with decreasing T. For Web sites B and H, which have a lot of training data, ΔMAP is larger than other Web sites and the impact of changes in T is relatively small. For Web site A, the MAPs of *existing 1* are higher than the scores when $T = 0.20$ and 0.15. The impact of changes in T is large. This result indicates that the model trained with more data achieves larger and more robust improvement. In comparing the existing methods, there are strong and weak points for each Web site. *Existing 1* achieved the best results for Web sites A, E, and F. Conversely, *existing 2* achieved the best results for Web sites B, C, D, G, and H. This means is that the importance of the features used to retrieve ads differs significantly depending on the Web site. In contrast with these existing methods, our proposed method uses both Web page and user information for ad retrieval, which is why it had better results.

Next, we investigated the model performance of each T when changing M. As described in Section 3.4.1, we change M and truncate the query term vector $\boldsymbol{q}_{input}$ during the evaluation, not during the training. The experimental results are shown in Figure 3.5. As expected, the MAPs decay in response to a decrease of M. One might think that our proposed method is rather worse than the basic model in situations where the ΔMAP is

Table 3.4: Example of Web site B's mapping table for user terms.

User term	Translated term	Weight
iPhone	iPhone	0.2114
	ケース (case)	0.1534
	iPad	0.0868
プリウス (Toyota Prius)	プリウス (Toyota Purius)	0.2600
	燃費 (mileage)	0.0732
	HV (Hybrid Vehicle)	0.0607
歯科 (dentistry)	歯科 (dentistry)	0.3297
	歯科医師 (dentist)	0.1892
	インプラント (implant)	0.1035
毛穴 (pores)	毛穴 (pores)	0.2319
	洗顔 (face washing)	0.1001
	化粧品 (cosmetics)	0.0663
温泉 (hot spring)	温泉 (hot spring)	0.1730
	旅館 (Japanese inn)	0.1272
	露天風呂 (outdoor hot spring)	0.0809
カーナビ (car navigation system)	カーナビ (car navigation system)	0.1229
	トヨタ (Toyota)	0.0906
	ホンダ (Honda)	0.0720

a negative value, such as $T = 0.05$ and $M = 50$ on Web site H. However, such offline evaluation results do not reflect the actual online performance very well because of the difference between t-score$(\boldsymbol{q}, \boldsymbol{a})$ and t-mscore$(\boldsymbol{q}, \boldsymbol{a})$, as described in Section 3.4.1. In the next section, the comparison of the existing methods and our proposed method when M is limited is carried over to the online evaluation. Here, we claim that M is first determined by the performance of the ad retrieval system and that T then needs to be tuned for each Web site in a real ad serving setting.

Tables 3.4 and 3.5 are examples of the mapping tables used on Web site B for user terms and categories, respectively. As expected, user terms are translated into the same term and related terms. In addition, the weight of the same term is larger than that of related terms in almost all cases. Similarly, user categories are translated into related terms.

3.4.2 Online Evaluation

In the previous subsection, we confirmed the potential improvement of the proposed method by conducting the offline evaluation with past click data. In this subsection, we evaluate the actual performance of our method in an online setting.

To measure the online performance, we applied our approach to a real ad serving

Table 3.5: Example of Web site B's mapping table user interest category.

User category	Translated term	Weight
Automotive/Domestic/Toyota	クラウン (Toyota Crown)	0.2605
	トヨタプリウス (Toyota Prius)	0.2171
	ランクル (Toyota Land Cruiser)	0.2053
Health Pharma/Adult Disease/Hypertensive Disease	血圧 (blood pressure)	0.1784
	高血圧 (high blood pressure)	0.1196
	食事法 (diet)	0.0531
Travel and Transportation/Overseas/Europe	海外 (overseas)	0.1181
	ヨーロッパ (Europe)	0.1168
	海外旅行 (foreign travel)	0.0868
Miscellaneous/Sex and Romance/Personals	婚活 (marriage hunting)	0.1100
	出会い (matchmaking)	0.0742
	カップル (couple)	0.0546
Life Stage/Wedding	ウェディング (wedding)	0.1398
	婚約 (engagement)	0.1391
	ドレス (dress)	0.1024

system. This ad serving system adopts a two-stage approach, as described in Section 3.2.3 and shown in Figure 3.2. Note that the objective of the first stage is to retrieve the ads that have high estimated CTR and pass on them to the second stage. We added the proposed method to the first stage and compared the online performance by conducting A/B testing. Hyper-parameters are set as ($T = 0.20, M = 20$). As noted in Section 3.3.3, we first determine the value of M in order to satisfy the requirement of response time and then choose the T value by considering the offline evaluation results and model training time. The response time when changing the value of M was estimated by preliminary system tests. For a fair comparison, we chose the value of M so as to make the response time of the two methods compared in A/B testing almost the same (tens milliseconds for each request). Furthermore, the total number of ads retrieved in the first stage is set the same because CTR can be higher even if the number of ads just increases. The CTR prediction model used for each version in the second stage was also the same. This prediction model was a statistical model trained by using the past click data [101]. We ran the online test over a 1-week period in November 2013 for each Web site.

We use three metrics for the online test: CTR, cost per click (CPC), and revenue per request (RPR). These metrics are defined as follows:

$$\begin{aligned}\text{CTR} &= \frac{\#\text{clicks}}{|R|},\\ \text{CPC} &= \frac{\text{revenue}}{\#\text{clicks}},\\ \text{RPR} &= \frac{\text{revenue}}{|R|},\end{aligned}$$

Table 3.6: Online A/B testing results. Metrics are click-through rate (CTR), cost per click (CPC), and revenue per request (RPR). Values represent the relative gains. We performed a chi-squared test on the CTR results.
* : p-value < 0.05, ** : p-value < 0.01, *** : p-value < 0.001

Metric	Web site			
	A	B	C	D
CTR	−3.67% **	+4.60% ***	+0.48%	+2.82% *
CPC	+3.63%	−2.00%	+1.62%	+1.31%
RPR	−0.18%	+2.51%	+2.10%	+4.17%

Metric	Web site			
	E	F	G	H
CTR	+2.47% **	+1.42%	+3.27%	+4.02% ***
CPC	−1.01%	+7.51%	−2.42%	−2.94%
RPR	+1.44%	+9.04%	+0.77%	+0.97%

where $|R|$ denotes the number of ad requests. revenue is the total amount of the fee that advertisers paid.

The experimental results are summarized in Table 3.6. These percentages also represent the relative gain. The CTR was improved for all Web sites expect site A. We simply set the hyper-parameters as $(T = 0.20, M = 20)$ for all Web sites although the result of $T = 0.20$ is worse than those of *existing 1* and *2* on Web site A in Table 3.3. Thus, this result is reasonable and indicates that hyper-parameters need to be tuned for each Web site. Web sites B and H, which have a lot of training data, have larger improvements in CTR than other Web sites as well as the offline test. We performed a chi-squared test on the CTR results. The results for the Web sites A, B, D, E, and H are statistically significant at the 5% level (p-value < 0.05). The RPR also improved for all Web sites expect A, whereas the CPC decreased for Web sites B, E, G, and H. This drop in CPC is usually favored by the advertisers. In this online testing, ads were ranked and displayed by considering revenues. Ads retrieved by our proposed method had a high CTR and relatively low bid price, which is why the Web sites had the result they did. As described in Section 3.1, we focus on increasing CTR in this study. Consequently, our proposed method improves revenue.

3.5 Conclusion

Contextual advertising is a form of textual advertising usually displayed on third-party Web pages. Because of the need to achieve both effective advertising and a positive user experience, one of the main problems with contextual advertising is determining how to select ads that are relevant to the page content and/or the user information. In this study, we introduced a translation method that learns a mapping of contextual information to the textual features of ads. The contextual information includes the user's demographic and behavioral information as well as Web page content information. Our proposed method only requires the transformation of the context feature vector with a learned matrix into the input vector of the ad retrieval system. Thus, it is easy to implement and there is no need to modify the existing inverted index or add new index. We evaluated this approach offline on a real-world dataset from an ad network and obtained better results than existing methods. We also applied our approach with a real ad serving system and achieved significant improvements over the existing production system.

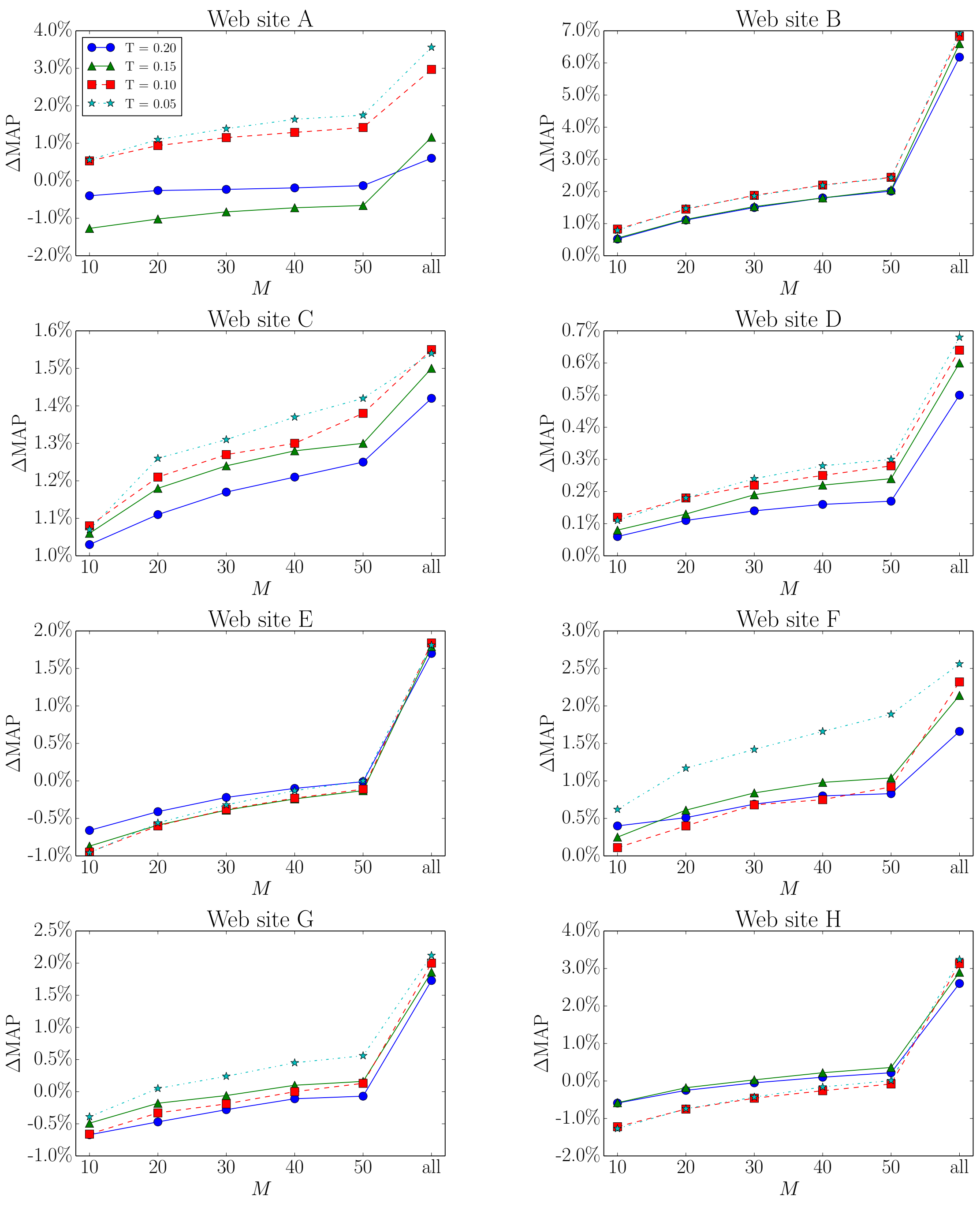


Figure 3.5: Experimental results when changing M.

Chapter 4

Multi-label Classification with an Extremely Large Number of Candidates

4.1 Introduction

Extreme multi-label classification has recently been receiving much attention. Its objective is to learn a classifier that can automatically annotate a data point with the most relevant subset from an extremely large label set (10^4 to 10^6) [83, 12]. In one example, an extreme multi-label classifier is learned to tag a new Wikipedia article by using a subset of the most relevant Wikipedia categories [81]. In another example, a classifier is built to recommend advertisers bid on some search keywords, given their ad landing pages [4]. Therefore, extreme multi-label classification tasks are closely related to recommendation and ranking tasks [83, 53].

When a label space is extremely large, a traditional baseline approach that builds a one-versus-rest classifier for each label is computationally expensive [114, 8]. More specifically, this naive approach needs to train an extremely large number of binary classifiers. Thus, some methods of dealing with this problem attempt to reduce the effective number of labels.

Some "embedding-based" approaches rely on the low-rank label matrix assumption [118, 75, 111]. This assumption means that there are correlations between labels, so these approaches learn a small number of "latent" factors of labels. Regressors are learned to perform prediction on these "latent" factors of labels with features and project them back onto the original high-dimensional label space. However, this assumption is violated in many real-world datasets because of the number of "tail" labels that only a few data points have [12, 111].

To address this problem, Sparse Local Embeddings for Extreme Classification (SLEEC) [12] was developed. SLEEC is also an embedding-based method but is free of any low-rank label matrix assumptions. This method first partitions data points by using k-means clustering and then learns a projection matrix (or regressor) for each partition by preserving distances from a relatively small number of nearest neighbors in the label space. In other words, SLEEC reduces the effective number of labels by converting a multi-label classification problem into a set of regression problems by using nearest neighbors in the label space. The number of these regression problems is independent of the number of labels, whereas the above naive one-versus-rest approach needs to train the same number of classifiers as labels. Prediction is performed by using the labels of training points close to the test point in the embedding space. In other words, SLEEC uses a k-nearest neighbor classifier in the embedding space.

However, SLEEC also has three problems. The first is learning to partition data. SLEEC partitions training points with k-means clustering before learning embeddings. This means SLEEC only uses feature vectors and does not access label information in this procedure. Therefore, data points that have similar label vectors are not guaranteed to be assigned to the same partition. This partitioning could affect the quality of embeddings learned in subsequent steps. The second is learning embeddings. In the prediction step, SLEEC predicts labels of the test point by using the nearest training points in the embedding space, as described above. Hence, the order of distance plays a crucial role, whereas the values themselves are not very important. Thus, the objective function of SLEEC is somewhat indirect for this purpose. In addition, SLEEC's optimization process for learning regressors is slightly complicated because of sparsity-induced and rank-constraint regularization terms. The third problem is prediction speed. Bhatia et al. [12] reported that SLEEC made predictions within 8 milliseconds per test point compared with 0.5 milliseconds for the tree-based FastXML [83] on a WikiLSHTC-325K dataset, but SLEEC made predictions much more accurately than FastXML. Although this prediction time would be acceptable for most real-world applications, much faster prediction is preferable for scaling up to solve Web-scale classification problems.

In this study, we present a novel graph embedding method named "AnnexML[1]," which copes with the all three problems in a comprehensive and more direct way on the basis of the k-nearest neighbor graph (KNNG). The key idea of AnnexML is reproducing the KNNG of label vectors in the embedding space to improve both the prediction accuracy and speed of the k-nearest neighbor classifier. The KNNG consists of training points as vertices. A directed edge connects from the i-th vertex to the j-th one if the j-th point is included in the set of the nearest neighbors of the i-th point in a certain metric space.

More specifically, AnnexML tackles the above three problems as follows. For the first problem, AnnexML learns a multiclass classifier, which partitions the approximate KNNG of the label vectors in order to preserve the graph structure as much as possible.

[1]**A**pproximate **N**earest **N**eighbor Search for **EX**treme **M**ulti-**L**abel Classification

Then, for the second problem, AnnexML projects each divided subgraph into an individual embedding space by formulating this problem as a ranking problem instead of regression one. This objective function is easily optimized with simple stochastic gradient descent. For the third problem, AnnexML efficiently retrieves approximate nearest neighbors of a test point by exploring the learned KNNG in the embedding space. This technique improves the trade-off between prediction time and accuracy.

In parallel with embedding-based approaches, tree-based approaches [4, 106, 83, 53, 55] are also common for extreme multi-label classification. These approaches achieve logarithmic time prediction because of their tree structure. This property of fast prediction is preferable for real-world applications. Thus, in this study, we develop a tree-based method as well as the embedding-based method.

FastXML [83] is a tree-based extreme multi-label classifier. This method learns an ensemble of multiple trees using random initialization, like random forest does [17]. During the training phase of each tree, FastXML recursively partitions the feature space corresponding to an internal node using a linear classifier optimized for normalized Discounted Cumulative Gain (nDCG) based ranking loss. A test point traverses the tree from the root node to a leaf node, and FastXML then predicts labels using the empirical label distribution of training points in the leaf node that a test point has reached.

Although FastXML can make fast predictions, it has lower prediction accuracy than some other approaches [12, 111, 8]. Some authors pointed out that this is due to the cascading effect of a tree structure [8]. Thus, this study aims to develop a new tree-based method that predicts more accurately while retaining the fast prediction property.

As described above, the FastXML prediction is made with training points in the leaf node that a test point has reached. This is considered to be a nearest neighbor classifier using all training points in the feature subspace of the leaf node. From this point of view, the key idea of our method is allocating the "nearest neighbors" on the label space to the same feature subspace. In other words, our method attempts to split each internal node of a tree while keeping as many "nearest neighbors" on the label space as possible. The prediction accuracy of the nearest neighbor classifier corresponding to a leaf node is expected to be improved by this splitting approach. We translate this idea into a graph partitioning problem and call our method a "graph partitioning tree" (GPT).

To summarize, our main contributions are as follows.

- We propose an extreme multi-label classifier based on graph embedding, AnnexML (Section 4.3)
    - We present a novel method of learning to partition data points by using an approximate KNNG as weak supervision, instead of unsupervised k-means clustering (Section 4.3.1)
    - For learning embeddings, we formulate this problem as a ranking one and op-

timize the objective function by using simple stochastic gradient descent (Section 4.3.2)

  - For faster prediction, we use an approximate k-nearest neighbor search technique by efficiently traversing the learned KNNG in the embedding space (Section 4.3.3)

- We propose the GPT — a tree-based extreme multi-label classifier (Section 4.4)
  - We introduce a novel method in which classifiers are learned by finding the minimum cut of the approximate KNNG (Section 4.4.2)
- We conducted experiments on several large-scale real-world datasets and compared our methods with recent state-of-the-art methods in terms of prediction accuracy and time (Section 4.5)

4.2 Problem Formulation

In this study, we consider a dataset $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$, which consists of N training points, where $\boldsymbol{x}_i \in \mathcal{X} \subseteq \mathbb{R}^M$ is an M-dimensional feature vector and $\boldsymbol{y}_i \in \mathcal{Y} \subseteq \{0,1\}^L$ is the corresponding L-dimensional label vector. $y_{ij} = 1$ if the i-th sample has the j-th label, and $y_{ij} = 0$ otherwise. Multi-label learning aims to build a classifier, $f : \mathbb{R}^M \to \{0,1\}^L$, that accurately predicts the label vector for a given sample.

In the context of extreme multi-label learning, the number of labels L is large and in the same order as the numbers of training points N and features M (see Table 4.2 for the statistics of the datasets used in the experiments). For example, a naive approach, which uses the one-versus-rest technique where an independent classifier is learned for each label, needs to train a massive number of binary classifiers. In addition, at the prediction time of this approach, all binary classifiers are applied to each test point. Thus, this naive approach might be computationally expensive in terms of both training and prediction time.

To overcome the above problem, some methods developed for extreme multi-label classification attempt to reduce the effective number of labels. We briefly describe these methods in Section 4.6.

4.3 AnnexML

In this section, we introduce our proposed method, AnnexML.

Hereafter, we represent an index set of entire training points as $\mathcal{I} = \{1, 2, \ldots, N\}$. Let $\boldsymbol{X} = [\boldsymbol{x}_1; \ldots; \boldsymbol{x}_N] \in \mathbb{R}^{M \times N}$ be the data matrix and $\boldsymbol{Y} = [\boldsymbol{y}_1; \ldots; \boldsymbol{y}_N] \in \mathbb{R}^{L \times N}$ be the label matrix.

Algorithm 1 Overview of AnnexML training (single learner)

```
Require: Training data: $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$
1: Partition $\mathcal{D}$ into $K$ subsets $\mathcal{D}_1, \dots, \mathcal{D}_K$    ▷ Section 4.3.1
2: for each parition $c$ do
3:     Learn projection matrix $\boldsymbol{V}_c$ using $\mathcal{D}_c$    ▷ Section 4.3.2
4:     $\boldsymbol{Z}_c \leftarrow \boldsymbol{V}_c \boldsymbol{X}_c$
5: end for
6: return $\{(\mathcal{D}_1, \boldsymbol{V}_1, \boldsymbol{Z}_1), \dots, (\mathcal{D}_K, \boldsymbol{V}_K, \boldsymbol{Z}_K)\}$
```

Algorithm 2 Overview of AnnexML prediction (single learner)

```
Require: Test point: $\boldsymbol{x}_t$, number of nearest neighbors: $n$
1: $c_t \leftarrow$ partition closest to $\boldsymbol{x}_t$
2: $\boldsymbol{z}_t \leftarrow \boldsymbol{V}_{c_t} \boldsymbol{x}_t$
3: $\tilde{\mathcal{N}}^{(t)}_{\boldsymbol{Z}_{c_t}} \leftarrow$ approximate $n$-nearest neighbors of $\boldsymbol{z}_t$ in $\boldsymbol{Z}_{c_t}$    ▷ Section 4.3.3
4: $\hat{\boldsymbol{y}}_t \leftarrow$ empirical label distribution for points in $\tilde{\mathcal{N}}^{(t)}_{\boldsymbol{Z}_{c_t}}$
5: return $\hat{\boldsymbol{y}}_t$
```

First, we describe training and prediction overviews. These high-level overviews are similar to those of SLEEC [12]. However, we stress that AnnexML is a graph embedding method that copes with the three problems of SLEEC in a comprehensively and straightforwardly on the basis of KNNG, which consists of training points as vertices.

Algorithm 1 shows an overview of the training procedure. First, AnnexML splits the data points into K partitions and then learns a linear map $\boldsymbol{V}_c \in \mathbb{R}^{d \times M}$ that projects the data points to a low-dimensional subspace for each partition $c \in \{1, \dots K\}$, instead of a global projection map. The embedding vector $\boldsymbol{z}_i \in \mathbb{R}^d$ is mapped from the feature vector $\boldsymbol{x}_i$ by using $\boldsymbol{V}_c$ for each $i \in \mathcal{I}_c$. Here, $\mathcal{I}_c$ is the index set of data points in the partition c. The data matrix of partition c is denoted as $\boldsymbol{X}_c$. Similarly, $\boldsymbol{Y}_c$ and $\boldsymbol{Z}_c$ represent the label and embedding matrix, respectively. AnnexML is able to improve its prediction accuracy by learning an ensemble of multiple learners with different random initializations, as described later (Section 4.3.1).

Algorithm 2 shows an overview of the prediction procedure. AnnexML first determines the partition to which a test point $\boldsymbol{x}_t$ belongs, finds approximate k-nearest neighbors (training points) in the low-dimensional subspace corresponding to the partition c_t, and finally predicts labels on the basis of the labels of neighbors. To make predictions with an ensemble of multiple learners, first, all sets of nearest neighbors obtained by learners are aggregated; then, AnnexML outputs an empirical label distribution of points in the aggregated nearest neighbors.

In the following subsections, we give details on the training and prediction procedures. The key point of AnnexML is reproducing the structure of the KNNG of the label vectors as much as possible in the embedding space. In Section 4.3.1, we represent a novel method

Table 4.1: Label frequency of training data used in experiments. For example, more than 174,000 labels occur in at most five training points in WikiLSTHTC-325K dataset.

Frequency	WikiLSHTC-325K		Amazon-670K	
1	79,732	(24.82%)	71,817	(10.76%)
≤ 2	112,788	(35.11%)	309,976	(46.45%)
≤ 3	137,596	(42.84%)	435,442	(65.25%)
≤ 4	157,541	(49.04%)	509,203	(76.31%)
≤ 5	174,341	(54.27%)	555,905	(83.30%)
≤ 10	226,956	(70.65%)	637,379	(95.51%)
All	321,222	(100.00%)	667,317	(100.00%)

of learning to partition data. AnnexML learns a multiclass classifier to divide the KNNG of the label vectors into subgraphs. Then, in Section 4.3.2, we learn a projection matrix in order to preserve the edge connections of each subgraph in an individual embedding space. Regarding this problem as a ranking one, we apply a learning-to-rank technique that uses cosine similarity. Finally, in Section 4.3.3, we present an approximate nearest neighbor search method that efficiently explores the learned KNNG in the embedding space. By replacing the naive brute-force calculation with this method, AnnexML successfully speeds up the prediction time without a noticeable drop in accuracy.

4.3.1 Learning to Partition Data Points

AnnexML partitions data points before learning embeddings for efficient training and prediction, like SLEEC does. Whereas SLEEC simply uses k-means clustering for this purpose, AnnexML aims to allocate the data points that have similar label vectors to the same partition. This means that AnnexML utilizes label information $\boldsymbol{y}$ but SLEEC only accesses feature vectors $\boldsymbol{x}$. The label frequency of an extreme multi-label classification dataset follows a “heavy tailed” distribution [12, 111]. Table 4.1 shows the label frequency in the training data of WikiLSHTC-325K and Amazon-670K datasets. About 54% and 83% of the labels occur in at most five training points on each dataset. Without label information, the data points that have the same tail label might be allocated to different partitions as the number of partitions increases. Thus, this difference between AnnexML and SLEEC could affect the quality of the embeddings learned in the subsequent steps and the final prediction results.

From the perspective of reproducing the KNNG of the label vectors, AnnexML divides the graph into K subgraphs while maintaining the structure as much as possible. Hence, this problem can be regarded as finding the minimum K-way graph cut [7]. However, in contrast with the common minimum K-cut problem, we need to learn a multi-class classifier to predict the partition of an unknown test point. Thus, we use a novel sequential maximization procedure for learning the classifier.

To construct the KNNG of the label vectors, we find the nearest neighbors $\mathcal{N}_{\boldsymbol{Y}}^{(i)}$ for each i-th data point from the indices of all training points $\mathcal{I}$. The set of nearest neighbors $\mathcal{N}_{\boldsymbol{Y}}^{(i)}$ is defined by using the inner product between normalized label vectors $\boldsymbol{y}/|\boldsymbol{y}|$ as:

$$\mathcal{N}_{\boldsymbol{Y}}^{(i)} := \underset{S:S\subseteq\mathcal{I},|S|=n,i\notin S}{\arg\max} \sum_{j\in S} \frac{\boldsymbol{y}_i^{\mathrm{T}}\boldsymbol{y}_j}{|\boldsymbol{y}_i||\boldsymbol{y}_j|}, \tag{4.1}$$

where S is the index set in which the number of elements equals n and $|\boldsymbol{y}_i| = \sum_j y_{ij}$ is the number of labels that a data point has.

The computational cost of naively finding $\mathcal{N}_{\boldsymbol{Y}}^{(i)}$ for all data points is $\mathcal{O}(N^2)$. Thus, this naive approach is infeasible for a large $N \sim 10^6$. If the label vectors $\boldsymbol{y}$ are sufficiently sparse, we can efficiently find the nearest neighbors $\mathcal{N}_{\boldsymbol{Y}}^{(i)}$ by using an inverted index. In this case, we first construct a list of references of data points that have each label, and then we just have to evaluate pairs of data points in each list. This procedure is similar to calculating the dot product between two sparse vectors, which only considers dimensions where both vectors have nonzero elements and sums up the products of their values. The estimated computational cost is $\sum_{j=1}^{L} n_j(n_j-1)/2$, where n_j is the number of data points that have the j-th label or the size of the j-th list. However, if a few n_j corresponding to "head" labels are near N, which means almost all data points have the same label, the above cost reaches $\mathcal{O}(N^2)$. Efficient algorithms for top-k retrieval on an inverted index [43] or finding approximate k-nearest neighbors [37] can be applied to this situation. However, we focus on tail labels and ignore head labels in some cases. In other words, we expect that the number of data points corresponding to head labels in each partition is not enough to affect the subsequent step, even if head labels are ignored in this step. Thus, we only consider the l tail labels and their corresponding lists under the conditions $\sum_{j=1}^{l} n_j(n_j-1)/2 \leq \alpha N,\ n_1 \leq n_2 \leq \ldots \leq n_L$ to find approximate nearest neighbors $\tilde{\mathcal{N}}_{\boldsymbol{Y}}^{(i)}$ in order to keep the computational cost within $\mathcal{O}(N)$ by using the adjusting parameter $\alpha \ll N$. Note that it is easy to determine l that satisfies the above condition by sorting n_js and accumulating $n_j(n_j-1)/2$ in ascending order.

After approximate nearest neighbors $\tilde{\mathcal{N}}_{\boldsymbol{Y}}^{(i)}$ are obtained for all data points, we learn the weight vector $\boldsymbol{w}_c$ for each partition c in order that the data points having the same tail labels are allocated in the same partition while the data points are divided almost equally among partitions. Using a sequential maximization procedure like stochastic k-means clustering [16], we sequentially maximize the following objective function for each i-th sample.

$$\max_{\boldsymbol{w}_{c_i}} \sum_{j\in\tilde{\mathcal{N}}_{\boldsymbol{Y}}^{(i)}} \log\sigma(\boldsymbol{w}_{c_i}^{\mathrm{T}}\boldsymbol{x}_j) + \sum_{k\in S^-} \log\sigma(-\boldsymbol{w}_{c_i}^{\mathrm{T}}\boldsymbol{x}_k) - \lambda|\boldsymbol{w}_c|_1, \tag{4.2}$$

where $c_i = \underset{c}{\arg\max}\ \boldsymbol{w}_c^{\mathrm{T}}\boldsymbol{x}_i$ is the partition to which the i-th point belongs at this time step, S^- is the set of indices randomly selected from $\mathcal{I}$, $\sigma(z) = 1/(1+\exp(-z))$ is a sigmoid function, and λ is a regularization parameter. We learn the linear classifier $\boldsymbol{w}_c$'s by using the FTRL-Proximal algorithm [72] with AdaGrad [39] learning rate adjustment.

The first term of the above objective function is intended to assign the approximate nearest neighbors $\tilde{\mathcal{N}}_{\boldsymbol{Y}}^{(i)}$ to the same partition c_i to which the i-th point belongs. The second term aims for the randomly selected points S^- to not be included in the partition c_i. Since some partitions that have a lot of data points cause the training and prediction time to become long, this term prevents a lot of data points from being allocated in a single partition. The last term is the L_1-regularization term to make $\boldsymbol{w}_c$'s sparse. Sparse $\boldsymbol{w}_c$'s are also preferable for faster prediction.

The above L_1-regularization term reduces the final model size as well as the prediction time. At training time, $\boldsymbol{w}_c$'s should be stored in dense vectors for faster training. These vectors use $\mathcal{O}(KM)$ space. Note that the number of partitions K depends on the number of all training points N (see Section 4.5). Thus, this space complexity is infeasible for large N and M. To remedy this problem, we use a hashing trick [105] for storing $\boldsymbol{w}_c$'s in 24-bit space (approximately 16.7 million bins) instead of $\mathcal{O}(KM)$ space. In practice, we did not notice a drop in accuracy when using this technique.

Since the above objective function is non-convex (when c_i is not fixed), AnnexML is also able to improve its prediction accuracy by learning an ensemble of multiple learners with different random initializations of $\boldsymbol{w}_c$, like the k-means clustering of SLEEC.

At the prediction step (line 1 in Algorithm 2), the partition c_t to which a test point $\boldsymbol{x}_t$ belongs is determined by using the learned $\boldsymbol{w}_c$'s as:

$$c_t = \arg\max_c \boldsymbol{w}_c^{\mathrm{T}} \boldsymbol{x}_t.$$

4.3.2 Learning Embeddings

For learning embedding vector $\boldsymbol{z}_i$ and projection matrix $\boldsymbol{V}_c$, AnnexML uses a pairwise learning-to-rank approach. This is because the learning objective of AnnexML is to reconstruct the KNNG of label vectors in the embedding space. In other words, arranging the k-nearest neighbors of the i-th sample is regarded as a ranking problem if we consider an i-th point and other points as a query and items, respectively. Thus, we use the pairwise learning-to-rank approach similar to S2Net [117] and deep structured sementic models (DSSM) [51] in order to optimize k-nearest neighbors in the embedding space more directly.

To represent the objective function of AnnexML, we first define the relevance score between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ as cosine similarity between embedding vectors $\boldsymbol{z}_i = \boldsymbol{V}_c\boldsymbol{x}_i$ and $\boldsymbol{z}_j = \boldsymbol{V}_c\boldsymbol{x}_j$:

$$R(\boldsymbol{x}_i, \boldsymbol{x}_j) := \cos(\boldsymbol{z}_i, \boldsymbol{z}_j) = \frac{\boldsymbol{z}_i^{\mathrm{T}}\boldsymbol{z}_j}{\|\boldsymbol{z}_i\|\|\boldsymbol{z}_j\|} = \frac{\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{V}_c^{\mathrm{T}}\boldsymbol{V}_c\boldsymbol{x}_j}{\|\boldsymbol{V}_c\boldsymbol{x}_i\|\|\boldsymbol{V}_c\boldsymbol{x}_j\|}.$$

We also represent the conditional probability by using the above relevance score and softmax function as follows:

$$\begin{aligned}P(\boldsymbol{x}_j \mid \boldsymbol{x}_i) &:= \frac{\exp(\gamma R(\boldsymbol{x}_i, \boldsymbol{x}_j))}{\exp(\gamma R(\boldsymbol{x}_i, \boldsymbol{x}_j)) + \sum_{k \in S_c^-} \exp(\gamma R(\boldsymbol{x}_i, \boldsymbol{x}_k))} \\ &= \frac{\exp(\gamma \cos(\boldsymbol{z}_i, \boldsymbol{z}_j))}{\exp(\gamma \cos(\boldsymbol{z}_i, \boldsymbol{z}_j)) + \sum_{k \in S_c^-} \exp(\gamma \cos(\boldsymbol{z}_i, \boldsymbol{z}_k))},\end{aligned}$$

where γ is a scaling parameter that magnifies $\cos(\boldsymbol{z}_i, \boldsymbol{z}_j)$ from $[-1, +1]$ to the range of larger values and $S_c^- \subseteq \mathcal{I}_c$ is the set of indices randomly selected from data points in the corresponding partition c. Then, we minimize the negative log likelihood:

$$\min_{\boldsymbol{V}_c} \sum_{i \in \mathcal{I}_c} \sum_{j \in \mathcal{N}_{\boldsymbol{Y}_c}^{(i)}} -\log P(\boldsymbol{x}_j \mid \boldsymbol{x}_i).$$

Here, the set of nearest neighbors $\mathcal{N}_{\boldsymbol{Y}_c}^{(i)}$ is almost the same as $\mathcal{N}_{\boldsymbol{Y}}^{(i)}$ (see Equation 4.1) but is selected from $\mathcal{I}_c$ rather than $\mathcal{I}$. Note that the computational cost of finding exact $\mathcal{N}_{\boldsymbol{Y}_c}^{(i)}$ for all i is not very large since the number of data points in a cluster $|\mathcal{I}_c|$ is much smaller than that of all training points $N = |\mathcal{I}|$ (the number of clusters to which K should be set to fulfill this condition). Thus, we can use the exact $\mathcal{N}_{\boldsymbol{Y}_c}^{(i)}$ for learning embeddings. The above negative log likelihood is transformed as:

$$\begin{aligned}&-\log P(\boldsymbol{x}_j \mid \boldsymbol{x}_i) \\ &= -\log\left(\frac{\exp(\gamma \cos(\boldsymbol{z}_i, \boldsymbol{z}_j))}{\exp(\gamma \cos(\boldsymbol{z}_i, \boldsymbol{z}_j)) + \sum_{k \in S_c^-} \exp(\gamma \cos(\boldsymbol{z}_i, \boldsymbol{z}_k))}\right) \\ &= -\log\left(\frac{1}{1 + \sum_{k \in S^-} \exp(\gamma(\cos(\boldsymbol{z}_i, \boldsymbol{z}_k) - \cos(\boldsymbol{z}_i, \boldsymbol{z}_j)))}\right) \\ &= \log\left(1 + \sum_{k \in S_c^-} \exp(-\gamma \Delta_{ijk}))\right),\end{aligned}$$

where $\Delta_{ijk} = \cos(\boldsymbol{z}_i, \boldsymbol{z}_j) - \cos(\boldsymbol{z}_i, \boldsymbol{z}_k)$ is the difference between two cosine values. Hence, this objective function aims to increase the difference between these cosine values.

We learn the projection matrix $\boldsymbol{V}_c$ by using stochastic gradient descent with AdaGrad [39]. Let a, b, and c be $\boldsymbol{z}_i^{\mathrm{T}} \boldsymbol{z}_j$, $1/\|\boldsymbol{z}_i\|$, and $1/\|\boldsymbol{z}_j\|$, respectively. The gradient of the $\cos(\boldsymbol{z}_i, \boldsymbol{z}_j)$ is derived as:

$$\begin{aligned}\frac{\partial \cos(\boldsymbol{z}_i, \boldsymbol{z}_j)}{\partial \boldsymbol{V}_c} &= \frac{\partial}{\partial \boldsymbol{V}_c}\left(\frac{\boldsymbol{z}_i^{\mathrm{T}} \boldsymbol{z}_j}{\|\boldsymbol{z}_i\| \|\boldsymbol{z}_j\|}\right) \\ &= -abc^3 \boldsymbol{z}_i \boldsymbol{x}_i^{\mathrm{T}} - acb^3 \boldsymbol{z}_j \boldsymbol{x}_j^{\mathrm{T}} + bc(\boldsymbol{z}_i \boldsymbol{x}_j^{\mathrm{T}} + \boldsymbol{z}_j \boldsymbol{x}_i^{\mathrm{T}}).\end{aligned}$$

Using cosine similarity instead of the inner product has two advantages. First, the objective function is regularized by the normalizing factor of a cosine. Hence, the explicit regularization term of $\boldsymbol{V}_c$ is not needed, so this learning procedure is simple. Second, we can easily apply an approximate nearest neighbor search technique that uses the learned KNNG for speeding up the prediction. We give the details of this technique in Section 4.3.3.

Algorithm 3 ANNSearch(Query $\boldsymbol{q}$, Ball tree T, KNNG G)

```
1: $H \leftarrow$ empty heap
2: TreeSearch($\boldsymbol{q}$, $T$.root, $H$)
3: GraphSearch($\boldsymbol{q}$, $G$, $H$)
4: return $H$
```

4.3.3 Faster Prediction using Approximate Nearest Neighbor Search on KNNG

The prediction of AnnexML mostly relies on the k-nearest neighbor search in the embedding space (see Algorithm 2). Thus, for faster prediction, we need to speed up this nearest neighbor search task. Instead of SLEEC's naive brute-force search, we apply an approximate k-nearest neighbor search method to this task. This method efficiently explores the learned KNNG in the embedding space by using an additional tree structure and a pruning technique via the triangle inequality. Note that we construct the KNNG precisely in the training phase and perform an approximate and fast search on the graph in the prediction phase.

Representing the nearest neighbor search task more concretely, we find the following index set $\mathcal{N}^{(t)}_{\boldsymbol{Z}_c}$ from training points $\mathcal{I}_c$ in a certain partition c with cosine similarity:

$$\mathcal{N}^{(t)}_{\boldsymbol{Z}_c} = \mathop{\arg\max}_{S: S\subseteq \mathcal{I}_c, |S|=n} \sum_{j\in S} \cos(\boldsymbol{z}_t, \boldsymbol{z}_j) = \mathop{\arg\max}_{S: S\subseteq \mathcal{I}_c, |S|=n} \sum_{j\in S} \frac{\boldsymbol{z}_t^{\mathrm{T}} \boldsymbol{z}_j}{\|\boldsymbol{z}_j\|},$$

where $\boldsymbol{z}_t = \boldsymbol{V}_c \boldsymbol{x}_t$ is the embedding vector that corresponds to a test point $\boldsymbol{x}_t$.

For efficient indexing in a metric space, the triangle inequality is a key element, as suggested by Sugawara et al. [97]. We also want to utilize the triangle inequality for efficient retrieval, but the cosine distance $1 - \cos(\boldsymbol{z}_t, \boldsymbol{z}_j)$ cannot satisfy this inequality. Fortunately, if the indexed vectors $\boldsymbol{z}_j$ are normalized in advance, that is, $\|\boldsymbol{z}_j\| = 1$ for all j, the above $\mathcal{N}^{(t)}_{\boldsymbol{Z}_c}$ becomes the same as the following index set of nearest neighbors by using the Euclidean distance:

$$\mathop{\arg\min}_{S: S\subseteq \mathcal{I}_c, |S|=n} \sum_{j\in S} \|\boldsymbol{z}_t - \boldsymbol{z}_j\|^2 = \mathop{\arg\min}_{S: S\subseteq \mathcal{I}_c, |S|=n} \sum_{j\in S} \left(\|\boldsymbol{z}_j\|^2 - 2\boldsymbol{z}_t^{\mathrm{T}} \boldsymbol{z}_j\right).$$

Thus, we use the Euclidean distance between normalized vectors as metrics for searching. Note that this transformation does not change the structure (or edge connections) of the learned KNNG.

Algorithms 3, 4, 5, and 6 are the pseudo codes that represent the approximate k-nearest neighbor search procedure using the KNNG. In these pseudo codes, the embedding of a test point $\boldsymbol{z}_t$ is represented as query vector $\boldsymbol{q}$. To efficiently find reasonable starting points from the graph, our method combines the KNNG with a ball tree. For the indexing method of the ball tree, refer to the papers [79, 85].

Algorithm 4 TreeSearch(Query $\boldsymbol{q}$, Tree node N, Heap H)

```
1: if $N$ is a leaf node then
2:     LineSearch($\boldsymbol{q}$, $N.S$, $H$)
3: else
4:     ldist ← $d(\boldsymbol{q}$, $N$.left.center)
5:     rdist ← $d(\boldsymbol{q}$, $N$.right.center)
6:     if ldist < rdist then
7:         TreeSeach($\boldsymbol{q}$, $N$.left, $H$)            ▷ follow the left child node
8:     else
9:         TreeSeach($\boldsymbol{q}$, $N$.right, $H$)           ▷ follow the right child node
10:    end if
11: end if
```

Algorithm 5 LineSearch(Query $\boldsymbol{q}$, Index set S, Heap H)

```
1: for $i \in S$ do
2:     dist ← $d(\boldsymbol{q}, \boldsymbol{z}_i)$
3:     if dist < LargestDistance($H$) then
4:         PopAndPushHeap($H$, $i$, dist)
5:     end if
6: end for
```

A query $\boldsymbol{q}$ first traverses from the root to the leaf nodes of the ball tree (Algorithm 4). At each internal node, the test point determines which child node (left or right) is to be followed by using the distances from the centers of the balls. After the test point reaches a leaf node, the distance from each indexed point corresponding to the node is calculated and pushed into the heap (Algorithm 5). Using these data points as seeds, the KNNG is explored (Algorithm 6). In the exploration step, if a data point is satisfied by the condition and pushed into the heap, the nearest neighbors are also pushed into the queue of candidates for the subsequent evaluation.

To summarize, we first obtain a reasonable set of approximate nearest neighbors by using a ball tree and then improve the approximation quality by exploring the KNNG on the basis of local search. Since we only need to calculate the distances to a small number of balls' centers and subset of training points, we can speed up this search task.

4.3.4 Comparison with SLEEC

In this subsection, we clarify the difference between AnnexML and SLEEC.

As noted in the beginning part of this section, the training and prediction procedures of AnnexML are similar to those of SLEEC. In other words, Algorithms 1 and 2 also show overviews of the training and prediction procedures of SLEEC, respectively. However, AnnexML has three improvements over SLEEC, which are described in the above

Algorithm 6 GraphSearch(Query $\boldsymbol{q}$, KNNG G, Heap H)

```
1: $C \leftarrow$ index of $H$                              ▷ a queue of candidates
2: $D \leftarrow$ empty set                                 ▷ a set of already evaluated
3: while $C$ is not empty do
4:     $i \leftarrow$ pop from $C$
5:     if $i$ in $D$ then
6:         continue
7:     end if
8:     $D \leftarrow D \cup \{i\}$
9:     dist $\leftarrow d(\boldsymbol{q}, \boldsymbol{z}_i)$
10:    if dist $<$ LargestDistance($H$) then
11:        PopAndPushHeap($H$, $i$, dist)
12:        for $j$ in $\mathcal{N}_{\boldsymbol{Z}_c}^{(i)}$ do            ▷ nearest neighbors in $G$
13:            push $j$ to $C$
14:        end for
15:    end if
16: end while
```

subsections.

For partitioning data points (line 1 in Algorithm 1), SLEEC simply applies usual k-means clustering to feature vectors of data points whereas AnnexML learns the classifiers by considering both feature and label vectors. Therefore, AnnexML is more likely to allocate the data points that have similar label vectors to the same partition and improve the quality of embeddings learned in subsequent steps.

For learning embeddings (line 3 in Algorithm 1), SLEEC utilizes the following objective function:

$$\min_{\boldsymbol{V}_c} \sum_{i \in \mathcal{I}_c} \sum_{j \in \mathcal{N}_{\boldsymbol{Y}_c}^{(i)}} \left\| \boldsymbol{y}_i^{\mathrm{T}} \boldsymbol{y}_j - \boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{V}_c^{\mathrm{T}} \boldsymbol{V}_c \boldsymbol{x}_j \right\|^2 + \lambda \sum_{i \in \mathcal{I}_c} |\boldsymbol{V}_c \boldsymbol{x}_i|_1 + \mu \left\| \boldsymbol{V}_c \right\|_F^2 .$$

The first term indicates the sum of squared errors between the inner product of label vectors and that of embedding vectors. The second term is an L_1-regularization term for embedding vectors $\boldsymbol{z}_i = \boldsymbol{V}_c \boldsymbol{x}_i$, which leads to sparse embeddings for reducing the size of models and the prediction time as well as avoiding overfitting. The last is the L_2-regularization term of $\boldsymbol{V}_c$. λ and μ are regularization parameters corresponding to the second and last terms, respectively. This objective function is non-convex and non-differentiable. Thus, the optimization process is divided into two phases and done using singular value projection [54] and ADMM [95].

Obviously, the above objective function of SLEEC is a regression problem of minimizing the sum of squared errors. This means SLEEC learns the projection matrix $\boldsymbol{V}_c$ to reproduce the inner product values of label vectors by using embedding vectors. However, at the prediction step, these learned values are used only for retrieving k-nearest neighbors. In contrast, the objective function of AnnexML focuses on whether a data point is included in the set of k-nearest neighbors or not. Thus, the learning procedure of AnnexML is more intuitive and consistent with the prediction procedure.

Instead of the approximate nearest neighbors search of AnnexML (line 3 in Algorithm 2), SLEEC just performs an exact and brute-force search for prediction. Note that the brute-force calculation of SLEEC cannot be directly replaced with our approximate search procedure because it does not use cosine similarity (or Euclidean distance) but rather the inner product as the metric in the embedding space. Some Maximum Inner-Product Search (MIPS) methods [85, 9, 91] might be applicable to SLEEC for faster prediction. However, such a study is beyond the scope of this paper. We leave this direction as future work.

4.4 GPT

In this section, we present our tree-based method, the GPT.

Algorithm 7 shows an overview of the training procedure. This overview is almost the same as that for conventional decision tree learning [17] and FastXML, but the procedure for splitting the internal node (SplitNode) is tailored for multi-label classification with an extremely large label space. This splitting procedure learns a binary linear classifier (or hyperplane), which partitions the feature space and training points. If the number of training points corresponding to the node n becomes less than the predefined threshold (Line 9), splitting is terminated, and the node is treated as a leaf node. The leaf node stores the empirical label distribution of the data points as follows:

$$\bar{\boldsymbol{y}}_n = \frac{1}{|\mathcal{I}_n|} \sum_{i \in \mathcal{I}_n} \boldsymbol{y}_i.$$

Here, we represent the index set corresponding to node n as $\mathcal{I}_n \subseteq \mathcal{I} = \{1, 2, \ldots, N\}$. This averaged label distribution is used for the subsequent prediction step.

Algorithm 8 shows an overview of the prediction procedure. This overview is also almost the same as that of traditional decision trees and FastXML. A test point traverses from the root node to a leaf node of a tree. At each internal node, the test point determines which child node to follow by using the linear classifier. After the test point reaches a leaf node, the label distribution corresponding to the leaf node is returned. When predicting with multiple trees, the predicted label vectors are aggregated.

In the following subsections, we describe the details of the splitting procedure for each internal node of the tree. First, the GPT constructs an approximate k-nearest neighbor graph of the label vectors (Section 4.4.1). Then, it learns the linear binary classifier using a sequential optimization procedure (Section 4.4.2). We also discuss the computational complexity of the training procedure and its scalability (Section 4.4.3). Finally, we compare the GPT with FastXML and explain the difference between them (Section 4.4.4).

Algorithm 7 Training overview of the GPT

```
Require: Training data: $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$
 1: for $i = 1, 2, \ldots, T$ do
 2:     $\mathcal{D}_{\text{root}} \leftarrow \mathcal{D}$
 3:     $\mathcal{T}_i \leftarrow$ new tree
 4:     $\mathcal{T}_i$.root $\leftarrow$ GrowTree($\mathcal{D}_{\text{root}}$)
 5: end for
 6: return $\{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_T\}$

 7: procedure GrowTree($\mathcal{D}_n$)
 8:     $n \leftarrow$ new node
 9:     if $|\mathcal{D}_n| <$ MaxInLeaf then
10:         $n.\boldsymbol{y} \leftarrow$ empirical label distribution of $\mathcal{D}_n$      ▷ process leaf node
11:     else
12:         $(\mathcal{D}_{\text{left}}, \mathcal{D}_{\text{right}}, \boldsymbol{w}_n) \leftarrow$ SplitNode($\mathcal{D}_n$)
13:         $n.\boldsymbol{w} \leftarrow \boldsymbol{w}_n$
14:         $n$.left $\leftarrow$ GrowTree($\mathcal{D}_{\text{left}}$)
15:         $n$.right $\leftarrow$ GrowTree($\mathcal{D}_{\text{right}}$)
16:     end if
17:     return $n$
18: end procedure

19: procedure SplitNode($\mathcal{D}_n$)
20:     $G_n \leftarrow$ ConstructApproximateKNNG($\mathcal{D}_n$)      ▷ Section 4.4.1
21:     $\boldsymbol{w}_n \leftarrow$ LearnPartitioner($G_n, \mathcal{D}_n$)      ▷ Algorithm 9 and Section 4.4.2
22:     $\mathcal{D}_{\text{left}} \leftarrow \{(\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathcal{D}_n \mid \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i \geq 0\}$
23:     $\mathcal{D}_{\text{right}} \leftarrow \{(\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathcal{D}_n \mid \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i < 0\}$
24:     return $(\mathcal{D}_{\text{left}}, \mathcal{D}_{\text{right}}, \boldsymbol{w}_n)$
25: end procedure
```

4.4.1 Constructing an Approximate k-nearest Neighbor Graph using Label Vectors

To split data points at each internal node of a tree, we first construct a KNNG on the label space. Each vertex of the graph corresponds to a training point. A directed edge is connected from the i-th vertex to the j-th one if the j-th point is included in the set of the "nearest neighbors" of the i-th point on the label space. In this study, the set of "nearest neighbors" of the i-th sample is defined by using the inner product between normalized label vectors $\boldsymbol{y}/|\boldsymbol{y}|$ as follows:

$$\mathcal{N}_n^{(i)} := \underset{S: S \subset \mathcal{I}_n, |S|=k, i \notin S}{\arg\max} \sum_{j \in S} \frac{\boldsymbol{y}_i^{\mathrm{T}} \boldsymbol{y}_j}{|\boldsymbol{y}_i||\boldsymbol{y}_j|},$$

Algorithm 8 Prediction overview of the GPT

```
Require: Test point: $\boldsymbol{x}_t$
 1: $\hat{\boldsymbol{y}} \leftarrow \boldsymbol{0}$                                ▷ initialize
 2: for $i = 1,\ 2,\ \dots$ do
 3:     $n \leftarrow T_i$.root                    ▷ start with root node of $i$-th tree
 4:     while $n$ is not a leaf node do
 5:         $\boldsymbol{w}_n \leftarrow n.\boldsymbol{w}$
 6:         if $\boldsymbol{w}_n^{\mathrm{T}} \boldsymbol{x}_t \geq 0$ then
 7:             $n \leftarrow n$.left                      ▷ follow the left child node
 8:         else
 9:             $n \leftarrow n$.right                    ▷ follow the right child node
10:         end if
11:     end while
12:     $\hat{\boldsymbol{y}} \leftarrow \hat{\boldsymbol{y}} + n.\boldsymbol{y}$   ▷ add a label distribution corresponding to the leaf node
13: end for
14: return $\hat{\boldsymbol{y}}$
```

where S is the index set in which the number of elements equals k and $|\boldsymbol{y}_i| = \sum_j y_{ij}$ is the number of labels a data point has.

Because the label vectors $\boldsymbol{y}$ are typically sparse, we can efficiently find the nearest neighbors $\mathcal{N}_n^{(i)}$ by using an inverted index. The estimated computational cost for all data points is $\sum_{j=1}^{L} n_j(n_j - 1)/2$, where n_j is the number of data points that have the j-th label. However, if a few n_j corresponding to "head" labels are near N, which means almost all data points have the same label, the above cost reaches $\mathcal{O}(N^2)$ at the root node. Therefore, we focus on tail labels and ignore some head labels. We only consider tail labels under the condition $n_j < n_{th}$ to find approximate nearest neighbors $\tilde{\mathcal{N}}_n^{(i)}$ by using the threshold parameter n_{th}. Using this simple approximation, we only consider tail labels when the node is close to the root and has a lot of training points. Even if we ignore some head labels, the partitioned two feature subspaces are expected to contain sufficient data points that have these labels. In contrast, we construct the graph using all labels when the node is near a leaf and it includes a relatively small number of data.

4.4.2 Learning a Linear Binary Classifier by Finding the Minimum Graph Cut

After an approximate KNNG is constructed using $\tilde{\mathcal{N}}_n^{(i)}$, we want to learn a linear classifier $\boldsymbol{w}_n \in \mathbb{R}^M$ by finding the minimum graph cut.

The graph cut is a partition that splits the vertices of the graph into two disjoint subsets [90]. In our problem setting, the set of training points $\mathcal{I}_n$ is partitioned into two disjoint subsets: $\mathcal{I}_l$ and $\mathcal{I}_r$. Here, we denote the index sets corresponding to the left and right child nodes of node n as $\mathcal{I}_l$ and $\mathcal{I}_r$, respectively. That is, $\mathcal{I}_l \cup \mathcal{I}_r = \mathcal{I}_n$, $\mathcal{I}_l \cap \mathcal{I}_r = \emptyset$. The size of the cut is defined as the number of edges between the two subsets (or edges crossing the partition). More concretely, we represent the size of the cut as follows:

$$\mathrm{cut}(\mathcal{I}_l, \mathcal{I}_r) = \sum_{i \in \mathcal{I}_n} \sum_{j \in \tilde{\mathcal{N}}_n^{(i)}} \left[\mathbb{1}(i \in \mathcal{I}_l \wedge j \in \mathcal{I}_r) + \mathbb{1}(i \in \mathcal{I}_r \wedge j \in \mathcal{I}_l) \right],$$

where $\mathbb{1}(P) = 1$ if the condition P is true, and $\mathbb{1}(P) = 0$ otherwise.

In contrast to the common minimum cut problem [90], we need to partition the graph with a hyperplane $\boldsymbol{w}_n$ on the feature space in order to predict unknown test points. $\boldsymbol{w}_n$ also splits the training points:

$$\mathcal{I}_l = \{i \in \mathcal{I}_n \mid \boldsymbol{w}_n^{\mathrm{T}} \boldsymbol{x}_i \geq 0\},$$
$$\mathcal{I}_r = \{i \in \mathcal{I}_n \mid \boldsymbol{w}_n^{\mathrm{T}} \boldsymbol{x}_i < 0\}.$$

In addition, because a balanced tree is preferable for fast training and prediction, we want $\mathcal{I}_l$ and $\mathcal{I}_r$ to be almost the same size.

Thus, similarly to stochastic k-means clustering [16], we sequentially maximize the following objective function for each i-th sample:

$$\sum_{j \in \tilde{\mathcal{N}}_n^{(i)}} \log \sigma(c_i \boldsymbol{w}_n^{\mathrm{T}} \boldsymbol{x}_j) + \sum_{j \in S_n^-} \log \sigma(-c_i \boldsymbol{w}_n^{\mathrm{T}} \boldsymbol{x}_j) - \lambda |\boldsymbol{w}_n|_1 \tag{4.3}$$

where $\sigma(z) = 1/(1+\exp(-z))$ is a sigmoid function, $S_n^- \subset \mathcal{I}_n$ is the set of indices randomly selected from data points the node of the tree has, and λ is a regularization parameter. c_i is an indicator variable representing to which side of the partition the i-th point belongs. $c_i = +1$ if $\boldsymbol{w}_n^{\mathrm{T}} \boldsymbol{x}_i \geq 0$, and $c_i = -1$ otherwise.

The first term of the Equation 4.3 is intended to assign the approximate nearest neighbors $\tilde{\mathcal{N}}_n^{(i)}$ to the same side of hyperplane c_i to which the i-th point belongs. However, the second term aims to avoid assigning the randomly selected points S_n^- to the same side c_i. This term prevents a lot of data points from being allocated to the same side of the hyperplane. The last term is the L_1-regularization term to make $\boldsymbol{w}_n$ sparse. Sparse $\boldsymbol{w}_n$'s are also preferable for faster prediction as well as small model size.

We learn the linear separator $\boldsymbol{w}_n$ using the FTRL-Proximal algorithm [72] with AdaGrad [39] learning rate scheduling. Algorithms 9 and 10 show the pseudo codes of the optimization procedure.

Because the above objective function is non-convex (when c_i is not fixed), the GPT can improve its prediction accuracy by learning an ensemble of multiple learners with a different random seed, like FastXML does.

4.4.3 Complexity Analysis

In this subsection, we present a complexity analysis of our training procedure.

Algorithm 9 Sequential optimization procedure for learning a linear classifier $\boldsymbol{w}_n$

```
1: procedure LEARNPARTITIONER($G_n, \mathcal{D}_n$)
2:     $\mathcal{I}_n \leftarrow$ index set of $\mathcal{D}_n$
3:     $(\boldsymbol{w}_n, \boldsymbol{n}, \boldsymbol{z}) \leftarrow (\boldsymbol{0}, \boldsymbol{1}, \boldsymbol{0})$
4:     for epoch = 1, 2, ... do
5:         $[i_1, i_2, \dots] \leftarrow$ random permutation of $\mathcal{I}_n$
6:         for $j = 1, 2, \dots, |\mathcal{I}_n|$ do
7:             $i \leftarrow i_j$
8:             if $\boldsymbol{w}_n^{\mathrm{T}} \boldsymbol{x}_i \geq 0$ then
9:                 $y_i \leftarrow 1$                                  ▷ $c_i = +1$ in Equation 4.3
10:            else
11:                $y_i \leftarrow 0$                                  ▷ $c_i = -1$ in Equation 4.3
12:            end if
13:            $\tilde{\mathcal{N}}_n^{(i)} \leftarrow$ taken from $G_n$
14:            $S_n^- \leftarrow$ indices randomly selected from $\mathcal{I}_n$
15:            for $j$ in $\tilde{\mathcal{N}}_n^{(i)}$ do
16:                $\beta \leftarrow y_i - \sigma(\boldsymbol{w}_n^{\mathrm{T}} \boldsymbol{x}_j)$
17:                $(\boldsymbol{w}_n, \boldsymbol{n}, \boldsymbol{z}) \leftarrow$ UPDATEVECTORS($\boldsymbol{x}_j, \beta, \boldsymbol{w}_n, \boldsymbol{n}, \boldsymbol{z}$)    ▷ Algorithm 10
18:            end for
19:            for $j$ in $S_n^-$ do
20:                $\beta \leftarrow 1 - y_i - \sigma(\boldsymbol{w}_n^{\mathrm{T}} \boldsymbol{x}_j)$
21:                $(\boldsymbol{w}_n, \boldsymbol{n}, \boldsymbol{z}) \leftarrow$ UPDATEVECTORS($\boldsymbol{x}_j, \beta, \boldsymbol{w}_n, \boldsymbol{n}, \boldsymbol{z}$)    ▷ Algorithm 10
22:            end for
23:        end for
24:    end for
25:    return $\boldsymbol{w}_n$
26: end procedure
```

The computational cost of constructing the approximate nearest neighbor graph is estimated as $\mathcal{O}(\bar{m} n_{th}^2 \hat{L})$. Here, $\bar{m}$ is the average number of nonzero features per sample and $\hat{L}$ is the number of labels corresponding to at most n_{th} training points. Typically, we set n_{th} to a value much smaller than N. $\bar{m}$ is also much smaller than M because the feature vector $\boldsymbol{x}$ is basically a sparse vector. The datasets used in our experiments (see Section 4.5) fulfill this condition. If the number of training points corresponding to all labels is smaller than or equal to n_{th}, $\hat{L}$ equals L. Therefore, the cost is basically proportional to the number of labels L and independent of N and M.

After the set of nearest neighbors is obtained, the computational cost of optimizing the Equation 4.3 is obviously made independent of the number of labels L (See Algorithms 9 and 10). The estimated cost for learning $\boldsymbol{w}_n$ is $\mathcal{O}\left(\bar{m}\left(|\tilde{\mathcal{N}}_n^{(i)}| + |S_n^-|\right)|\mathcal{I}_n|\right)$. Typically, we set $|\tilde{\mathcal{N}}_n^{(i)}|$ and $|S_n^-|$ to a value that is much smaller than N. Therefore, these values are considered to be constant, and the above cost turns out to be $\mathcal{O}(\bar{m}|\mathcal{I}_n|)$ for each

Algorithm 10 Update vectors using FTRL-Proximal with AdaGrad learning rate scheduling

```
Require: Initial learning rate: $\eta_0$, $L_1$-regularization parameter: $\lambda$
 1: procedure UpdateVectors($\boldsymbol{x}, \beta, \boldsymbol{w}, \boldsymbol{n}, \boldsymbol{z}$)
 2:     $I = \{i \mid x_i \neq 0\}$            ▷ indices corresponding to nonzero features
 3:     for $i \in I$ do
 4:         $g_i \leftarrow \beta x_i$
 5:         $s_i \leftarrow \frac{1}{\eta_0}\left(\sqrt{n_i + g_i^2} - \sqrt{n_i}\right)$
 6:         $z_i \leftarrow z_i + g_i + s_i w_i$
 7:         $n_i \leftarrow n_i + g_i^2$
 8:         if $|z_i| \leq \lambda$ then
 9:             $w_i \leftarrow 0$
10:         else
11:             $w_i \leftarrow \frac{\eta_0}{\sqrt{n_i}}(z_i - \mathrm{sign}(z_i)\lambda)$
12:         end if
13:     end for
14:     return $(\boldsymbol{w}, \boldsymbol{n}, \boldsymbol{z})$
15: end procedure
```

$\boldsymbol{w}_n$. Because the total number of data points included in nodes at any tree depth d is less than or equal to N, the sum of computational costs of the nodes at tree depth d is $\mathcal{O}(\bar{m}N)$. In addition, because the maximum tree depth is expected to be $\mathcal{O}(\log N)$ (if the tree is balanced well), the expected overall computational cost of all nodes in a single tree is $\mathcal{O}(\bar{m}N \log N)$. Thus, the cost is proportional to the number of samples N and independent of M and L.

The above analysis shows that the proposed training procedure can address the problems with large N and L.

4.4.4 Comparison with FastXML

In this subsection, we first describe the learning procedure to partition a node of FastXML and then explain the differences between the GPT and FastXML.

FastXML optimizes the following nDCG-based objective function for an internal node n of the tree:

$$\begin{aligned}\max_{\boldsymbol{w}_n, \boldsymbol{c}, \boldsymbol{r}^+, \boldsymbol{r}^-} & \sum_{i \in \mathcal{I}_n} \log \sigma(c_i \boldsymbol{w}_n^{\mathrm{T}} \boldsymbol{x}_i) - \lambda |\boldsymbol{w}_n|_1 \\ & + \sum_{i \in \mathcal{I}_n} \frac{1}{2}(1 + c_i)\mathcal{L}_{\mathrm{nDCG}@L}(\boldsymbol{r}^+, \boldsymbol{y}_i) \\ & + \sum_{i \in \mathcal{I}_n} \frac{1}{2}(1 - c_i)\mathcal{L}_{\mathrm{nDCG}@L}(\boldsymbol{r}^-, \boldsymbol{y}_i),\end{aligned}$$

where $\boldsymbol{w}_n \in \mathbb{R}^M$ is the linear separator and $\boldsymbol{c} \in \{+1, -1\}^N$ is a cluster assignment vector that consists of c_i's. $\boldsymbol{r}^{\pm} \in \Pi(1, L)$ are label ranking vectors corresponding to clusters, as we denote the set of all permutations of $\{1, 2, \ldots, L\}$ as $\Pi(1, L)$. Note that the above objective function is slightly changed from that described in the original paper [83] for easy comparison to our Equation 4.3. However, they are basically identical. The function $\mathcal{L}_{\text{nDCG@}L}(\boldsymbol{r}, \boldsymbol{y})$ in the third and fourth terms is defined on the basis of nDCG as follows:

$$\mathcal{L}_{\text{nDCG@L}}(\boldsymbol{r}, \boldsymbol{y}) = \frac{\mathcal{L}_{\text{DCG@L}}(\boldsymbol{r}, \boldsymbol{y})}{\sum_{l=1}^{\min(L, \mathbf{1}^{\mathrm{T}}\boldsymbol{y})} \frac{1}{\log_2(1+l)}},$$

$$\mathcal{L}_{\text{DCG@L}}(\boldsymbol{r}, \boldsymbol{y}) = \sum_{l=1}^{L} \frac{y_{r_l}}{\log_2(1+l)}.$$

Since the above function includes non-continuous nDCG-based terms, direct optimization is difficult. Thus, the optimization process is divided into two phases. First, FastXML optimizes $\boldsymbol{c}$ and $\boldsymbol{r}^{\pm}$ by keeping $\boldsymbol{w}_n = \mathbf{0}$. Then, it optimizes $\boldsymbol{w}_n$ with fixed $\boldsymbol{c}$.

The first step, which optimizes $\boldsymbol{c}$ and $\boldsymbol{r}^{\pm}$, is regarded as a variant of k-means clustering algorithm with the nDCG-based metric and $k = 2$. At the beginning, each cluster assignment c_i is randomly initialized by $+1$ or -1. Then the optimization procedure updates $\boldsymbol{r}^{\pm}$ using label vectors $\boldsymbol{y}_i$'s, where $c_i = \pm 1$ for increasing the objective value. This is analogous to a k-means clustering algorithm setting the mean vector using data points currently allocated in each cluster. Similarly, whereas $\boldsymbol{r}^{\pm}$ is fixed, c_i for each data point is reassigned. These update and reassignment steps are repeated alternately until cluster assignments no longer change.

The second step, which learns $\boldsymbol{w}_n$, is identical to solving the ordinal L_1-regularized logistic regression problem [119] because the first and second terms in the objective function are only considered in the optimization.

As described above, whereas FastXML considers the difference between each label vector $\boldsymbol{y}_i$ and the "mean" label ranking vector $\boldsymbol{r}^{\pm}$ corresponding to the cluster to which it belongs, GPT simply focuses on whether a data point and its nearest neighbor are allocated to the same side of the hyperplane or not. This difference should improve the prediction accuracy of the k-nearest neighbor classifier in a leaf node.

Furthermore, FastXML first determines cluster assignments $\boldsymbol{c}$ using a variant of the k-means clustering algorithm on the label space and then learns a linear separator $\boldsymbol{w}_n$ by solving a classification problem on the feature space using the cluster assignments as class labels. On the other hand, GPT optimizes $\boldsymbol{w}_n$ and updates $\boldsymbol{c}$ on the feature space sequentially and simultaneously. Therefore, GPT is expected to obtain a better optimization result. Of course, FastXML can update $\boldsymbol{w}_n$ multiple times by repeating the above two steps alternately. However, Prabhu and Varma reported that multiple updates of $\boldsymbol{w}_n$ do not improve the prediction accuracy much [83].

Table 4.2: Statistics of datasets used in experiments

Dataset	#Train N	#Test N_{test}	#Features M	#Labels L	Avg. labels per point
AmazonCat-13K	1,186,239	306,782	203,882	13,330	448.57
Wiki10-31K	14,146	6,616	101,938	30,938	18.64
Delicious-200K	196,606	100,095	782,585	205,443	75.54
WikiLSHTC-325K	1,778,351	587,084	1,617,899	325,056	3.19
Amazon-670K	490,449	153,025	135,909	670,091	5.45

4.5 Experiments

In this section, we evaluated our methods on five large scale multi-label datasets: AmazonCat-13K [71], Wiki10-31K [120], DeliciousLarge-200K [107], WikiLSHTC-325K [81], and Amazon-670K [71]. These datasets were provided by the Extreme Classification Repository [13] and had already been pre-processed and separated into training and test sets. We did not use any additional meta data. The statistics for the datasets are summarized in Table 4.2.

We compared AnnexML and GPT with several state-of-the-art methods: SLEEC [12], FastXML [83], PfastreXML [53], PLT [55], and PD-Sparse [114]. To represent the prediction difficulty of each dataset, we also show the performance of a naive baseline that makes predictions by using the k-most common labels in the training data.

We evaluated the performance of the methods with precision at k ($k \in \{1, 3, 5\}$), which is a widely used metric for extreme multi-label classification and ranking tasks:

$$\text{P@}k := \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} \frac{y_{i\pi(l)}}{k}.$$

Here, $\pi(k) = j$ means that the j-th label is ranked in the k-th position by the predicted score. We also evaluated the performances with normalized Discounted Cumulative Gain (nDCG) at k. However, since the results showed the same tendency, we only report those with P@k[2]. At least, by definition, the value of nDCG@1 is the same as that of P@1.

We implemented AnnexML and GPT in C++. In all experiments, we used 10 default hyper-parameters for training AnnexML: the number of partitions in a learner: $K = \lfloor N/6000 \rfloor$, the embedding dimension: $d = 50$, the number of learners: 15, the number of (approximate) nearest neighbors and randomly sampled points used in learning both partitionings and embeddings: $n = 10$, the number of epochs for learning both partitionings and embeddings: 10, the initial learning rate of AdaGrad: $\eta_0 = 0.1$, the L_1-regularization parameter in Equation 4.2: $\lambda = 4$, the scaling parameter for cosine: $\gamma = 10$, the adjustment parameter for finding approximate nearest neighbors: $\alpha = 5000$, and the number of edges in the KNNG for prediction: 50. We also used common hyper-

[2]We show the results of nDCG@k for AnnexML and GPT in the Appendix section.

parameters to train GPT; the threshold parameter for finding approximate nearest neighbors: $n_{th} = 50$, the number of approximate nearest neighbors and randomly sampled points used in learning: $|\tilde{\mathcal{N}}_n^{(i)}| = |S_n^-| = 10$, the number of epochs for optimization $\boldsymbol{w}_n$: 10, the initial learning rate: $\eta_0 = 0.1$, the L_1 regularization parameter: $\lambda = 4.0$, and the maximum number of data points allowed in a leaf node: MaxInLeaf = 10. These default values of hyper-parameters were determined in preliminary experiments with small-scale datasets. Thus, we avoided hyper-parameter tuning for the large-scale datasets and significantly reduced the total training time. In fact, the AnnexML and GPT models learned with these common values achieved sufficiently good results on various datasets as shown in the next subsection.

For the other baseline methods, if P@k on each dataset is reported in the original papers, we used those values for fair comparison. Otherwise, we used the C++ and MATLAB implementations for SLEEC, FastXML, PfastreXML and PD-Sparse, provided by the authors [13]. In this case, the suggested hyper-parameters were used. Since we use ordinary (not propensity scored) precision at k as the evaluation metric, the propensity scores of PfastreXML were set to the same value for all labels. Thus, trees that PfastreXML learns are basically the same as those that FastXML learns in this setting. The difference between these methods is the use additional classifiers for improving tail label predictions. For PLT, we referred only to the values reported in the original paper because hyper-parameters were tuned by an off-the-shelf optimizer in the experiments.

4.5.1 Results

The experimental results are summarized in Table 4.3. The **bold** elements indicate the best performance of the methods. The underlined scores are the best results of the tree-based methods. The scores of PD-Sparse on Delicious-200K and WikiLSTHC-325K datasets are unavailable because of excessive memory usage and training time in our experimental setting. Thus, we referred to the scores on these datasets reported in the Extreme Classification Repository [13].

AnnexML performed the best in 8 of 15 cases. There were especially large improvements for datasets that have larger label spaces. For example, AnnexML improved over SLEEC by about 8% and 6% in absolute terms of P@1 and P@5 on the WikiLSHTC-325K dataset. On the Amazon-670K dataset, AnnexML was also superior to SLEEC by approximately 7% and 4% in absolute terms of P@1 and P@5. These substantial improvements indicate AnnexML is not just a minor updated version of SLEEC. Although there is still much room to improve the prediction accuracy of AnnexML by tuning the hyper-parameters for each dataset, like SLEEC does, we leave this as future work.

The GPT achieved the best performances of the four tree-based methods for four of five datasets. For example, on the WikiLSHTC-325K dataset, it was superior to PfastreXML, the second best, by approximately 5% in absolute terms of P@1. On the Amazon-670K dataset, GPT was also superior to PfastreXML by about 3% in absolute terms of P@1. From these results, we see that the proposed graph cut approach successfully improves prediction accuracy.

Table 4.4 compares the results of the three improvements for AnnexML, proposed in Section 4.3. Method `E+P+A` equals AnnexML in Table 4.3. When we did not use the proposed partitioning or approximate nearest neighbor search, we used k-means clustering and a brute-force cosine calculation, like SLEEC does. Method `E` outperformed SLEEC on almost all datasets. Thus, our learning procedure that uses the learning-to-rank approach successfully improved the embedding quality. In comparison with `E` and `E+P`, the proposed partitioning method significantly improved P@k on the datasets that have a larger label space. In comparison with `E` and `E+A`, the approximate nearest neighbor search technique for faster prediction had slightly poorer accuracy on all datasets except Delicious-200K. These results are consistent with our reason for proposing method `A`, which is not to make prediction more accurate, but faster. One possible explanation for the unexpected improvement on Delicious-200K was that the number of nearest neighbors retrieved for prediction was small (set to be 10 for all datasets). The number suggested by SLEEC's authors is 70 for this dataset. Hence, the approximate nearest neighbor search retrieved more diverse samples, and these samples could fortunately stabilize the prediction result. Next, we present the speed-up effect of the approximate nearest neighbor search.

Figure 4.1 plots the prediction time and performances of AnnexML, AnnexML-BF, SLEEC, GPT, FastXML, and PfastreXML when the number of learners changes. A higher precision at the same prediction time (upper left line) indicates better results. Due to space limitations, we only show the P@1 and P@5 results. This experiment was conducted by using a single CPU thread on a machine with two Xeon E5-2680v3 processors and 128 GB of RAM that ran the Linux operating system. AnnexML-BF is the same method as `E+P` in Table 4.4, which uses a brute-force cosine calculation for prediction, instead of approximate nearest neighbor search. Note that AnnexML is the same as `E+P+A`. We chose SLEEC, FastXML and PfastreXML for comparison since they are also ensemble methods, and we can easily control the trade-off between prediction time and accuracy by just changing the number of learners. To conduct a fair comparison, we used our own C++ implementation for SLEEC's prediction, instead of the provided MATLAB codes. This is almost the same as the implementation of AnnexML-BF. For FastXML and PfastreXML, we used the provided C++ implementations with careful coding optimizations for improving efficiency.

Comparing AnnexML and AnnexML-BF, we see that the approximate nearest neighbor search technique successfully sped up prediction time with a slight drop in accuracy. On the WikiLSHTC-325K dataset, AnnexML made predictions more than four times faster than AnnexML-BF when using the same number of learners. In other words, the prediction time with an ensemble of four AnnexML learners was almost the same as that of a single AnnexML-BF model. In this case, AnnexML made predictions in approximately 0.34 milliseconds per test point and achieved an about 5% higher P@1 than that of AnnexML-

BF in absolute terms (0.6121 vs. 0.5657).

Compared with FastXML and PfastreXML, AnnexML also achieved a higher P@k at a 1 millisecond budget per test point in almost all cases. AnnexML achieved the same prediction accuracy as SLEEC with an ensemble of at most four learners, which also made predictions within 1 millisecond per test point. In particular, on the WikiLSHTC-325K dataset, a single model of AnnexML achieved almost the same P@1 as an ensemble of 15 SLEEC learners. This AnnexML's prediction time was about 58 times shorter than that of SLEEC (0.08 vs. 4.66 milliseconds).

The results show that the GPT achieves higher prediction accuracy than FastXML and PfastreXML in the same prediction-time budget in almost all cases. For example, on the WikiLSHTC-325K dataset, the ensemble of 16 GPT trees made predictions in approximately 0.25 milliseconds per test point. In other words, GPT can predict about 4000 examples per second using a single CPU thread. In this case, it achieves higher P@1 than the 128 PfastreXML trees (0.6159 vs. 0.5885). The difference in prediction time between FastXML and PfastreXML was caused by additional classifiers.

Table 4.5 shows the tree balance learned by the GPT and FastXML. As mentioned earlier, because FastXML and PfastreXML basically learn the same trees in our setting, we omitted the results for PfastreXML. The experimental results show that both GPT and FastXML learn well-balanced trees. On the WikiLSTHC-325K dataset, which has the largest number of training points among the datasets used in our experiments, about 18 linear classifiers are evaluated on average for a prediction of a test point using a single GPT tree. Even if an ensemble of 50 trees is used for the prediction, the number of applied classifiers is much smaller than the number of labels. Therefore, the computational cost of prediction using these classifiers is much lower. Next, we present the actual prediction time of these methods.

The sizes of learned models are presented in Table 4.6. This experiment was also conducted using the C++ implementations for a fair comparison. The GPT obtained small models on almost all datasets expect WikiLSTC-325K, for which the GPT model is about twice the size of the FastXML and PfastreXML models. However, P@1 of 16 GPT trees is higher than that of 32 PfastreXML trees on the WikiLSTC-325K dataset (see Figure 4.1). Therefore, under the condition of the same model size, GPT can predict more accurately.

By using 24 CPU threads on the aforementioned machine, the ensemble of 15 AnnexML models on WikiLSTHC-325K datasets were trained within 4 hours. Under the same conditions, the 50 trees of the GPT could be learned in approximately 6 hours on the same dataset. Thus, the AnnexML and GPT can handle the problems with hundreds of thousands of labels on a single commodity machine within a reasonable training time.

4.6 Related work

Extreme multi-label learning typically follows two major types of approaches: tree based [83, 53, 55] and embedding based [118, 12, 75, 111].

Tree-based methods are common for extreme multi-label classification because of their fast prediction. Multi-label random forests (MLRF) [4] extends random forest to efficiently handle extreme multi-label problems. The node splitting criterion is calculated using positive labels alone. The authors reported that distributed learning on a cluster of a thousand compute nodes make it possible to train on 90 million training points in less than a day. Label partitioning for sublinear ranking (LPSR) [106] improves the prediction speed of an already learned classifier in a post-hoc manner. This method converts a predictor that has linear time in the number of labels to a sublinear one via label partitioning. FastXML [83] can be trained on problems with more than a million labels on a standard desktop in a few hours. PfastreXML [53] is an improved version of FastXML that replaces the nDCG loss with its propensity scored variant and uses additional classifiers designed for tail labels. Jasinska et al. [55] developed PLTs, which are tree-based classifiers that maximize the F-measure. GBDT-Sparse [94] is the gradient boosted decision trees (GBDT) when the output space is high dimensional and sparse. Thus, GBDT-Sparse efficiently deals with extreme multi-label classification tasks.

Most embedding-based approaches reduce the effective number of labels on the basis of the low-rank label matrix assumption. Low rank empirical risk minimization for multi-Label learning (LEML) [118] learns a low-rank projection matrix, which maps features to labels, by using a generic empirical risk minimization framework. Mineiro and Karampatziakis developed another embedding-based method, named "Rembrandt" [75], by using techniques of randomized linear algebra. Robust extreme multi-label learning (REML) [111] decomposes the label matrix into a low-rank structure and sparse component, which represent label correlations and outliers, respectively. Si et al. proposed Goal-directed Inductive Matrix Completion (GIMC) [93] and applied it to multi-label classification. In this study, since SLEEC was reported to perform better than or comparably to the above embedding-based methods on larger datasets [12, 75, 111, 93], we only compared our method with SLEEC.

An extreme multi-label classifier also needs to be able to keep the model size small. Therefore, some authors have proposed sparsity-induced methods. PD-Sparse [114] uses primal and dual-sparse formulation, which consists of dual-sparse loss and a primal-sparse regularizer. By using these two types of sparsity and hashing techniques, PD-Sparse can be efficiently learned and it achieves fast prediction. PPDsparse [113] extends PD-Sparse for efficient parallelization in large-scale distributed settings.

Distributed learning is another approach to extreme multi-label classification. Babbar and Schölkopf proposed distributed sparse machines for extreme multi-label classification (DiSMEC) [8], which is a large-scale distributed framework for learning one-versus-rest linear classifiers. By using a distributed framework, DiSMEC learns a L_2-regularized L_2-loss SVM for each label in parallel. To reduce the model size and make prediction faster, they pruned *ambiguous* weights in the region near zero after training. They reported that the model for an entire WikiLSHTC-325K dataset can be trained in approximately 6 hours on 400 cores and 3 hours on 1,000 cores. In this study, we did not compare our method with DiSMEC because we focus on non-distributed approaches.

Liu et al. [66] proposed a convolutional neural network based model for extreme multi-label classification. The training and prediction procedure is efficiently done by using a graphics processing unit (GPU).

In an approximate similarity search task for neural word embeddings, Sugawara et al. [97] compared hash-based, tree-based, and graph-based algorithms. They reported that a graph-based indexing method (neighborhood graph and tree; NGT [52]) outperformed other methods. NGT is an indexing method that combines a variant of a vantage point tree [115] with an approximate KNNG. In our experiments, the KNNG combined with the ball tree also successfully accelerated the prediction speed without a noticeable drop in accuracy.

Tang et al. developed LargeVis [102] for visualizing large-scale and high-dimensional data (100 to 784 dimensions in their experiments) in 2- or 3-dimensional space. First, LargeVis constructs an approximate KNNG. Then, the model allocates the data points in low-dimensional space by preserving the structures of the graph. Thus, LargeVis is somewhat similar to AnnexML. However, AnnexML tries to find approximate k-nearest neighbors in the label space $\mathcal{Y}$ for classification, whereas LargeVis attempts to find them on feature space $\mathcal{X}$ for (unsupervised) visualization. Furthermore, AnnexML learns the projection matrix $\boldsymbol{V}_c$ that maps feature vectors $\boldsymbol{x}$ to embeddings $\boldsymbol{z}$ to predict unseen test points, although LargeVis directly learns the $\boldsymbol{z}$ of data points.

4.7 Conclusion

In this study, for extreme multi-label classification tasks, we presented AnnexML and GPT. Experimental results on several large-scale datasets showed that our proposed methods can significantly improve prediction accuracy, especially on larger datasets.

For future work, we plan to incorporate some of the ideas of approximate nearest neighbor search methods into an extreme multi-label classifier based on more complex nonlinear models, such as kernel machines and deep neural networks. In addition, we will investigate some model compression techniques. These techniques will provide compact models while keeping the prediction fast and accurate.

We released our implementation of AnnexML on the `github.com`[3]. Our code should be

[3]`https://github.com/yahoojapan/AnnexML`

useful for both researchers who want to compare their results with ours and practitioners who try to solve real-world Web-scale classification problems.

Table 4.3: Experimental results

Dataset		Embedding-based		Others	
		AnnexML	SLEEC	PD-Sparse	Most common
AmazonCat-13K	P@1	**0.9355**	0.8919	0.8931	0.2988
	P@3	**0.7838**	0.7517	0.7403	0.1878
	P@5	0.6332	0.6109	0.6011	0.1486
Wiki10-31K	P@1	**0.8650**	0.8554	0.7771	0.8079
	P@3	**0.7428**	0.7359	0.6573	0.5050
	P@5	**0.6419**	0.6310	0.5539	0.3675
Delicious-200K	P@1	0.4666	0.4703	0.3437	0.3873
	P@3	0.4079	**0.4167**	0.2948	0.3675
	P@5	0.3764	**0.3888**	0.3621	0.3552
WikiLSHTC-325K	P@1	**0.6336**	0.5557	0.6126	0.1588
	P@3	**0.4066**	0.3306	0.3948	0.0603
	P@5	**0.2979**	0.2407	0.2879	0.0380
Amazon-670K	P@1	0.4208	0.3505	0.3370	0.0028
	P@3	0.3665	0.3125	0.2962	0.0027
	P@5	0.3276	0.2856	0.2684	0.0023

Dataset		Tree-based			
		GPT	FastXML	PfastreXML	PLT
AmazonCat-13K	P@1	0.9084	0.9310	0.8994	0.9147
	P@3	0.7676	0.7818	0.7724	0.7584
	P@5	0.6255	0.6338	**0.6353**	0.6102
Wiki10-31K	P@1	0.8476	0.8295	0.8263	0.8434
	P@3	0.7322	0.6756	0.6874	0.7234
	P@5	0.6320	0.5770	0.6006	0.6272
Delicious-200K	P@1	**0.4746**	0.4320	0.3762	0.4537
	P@3	0.4165	0.3868	0.3562	0.3894
	P@5	0.3871	0.3621	0.3403	0.3588
WikiLSHTC-325K	P@1	**0.6336**	0.4975	0.5810	0.4567
	P@3	0.3997	0.3310	0.3761	0.2913
	P@5	0.2906	0.2445	0.2769	0.2195
Amazon-670K	P@1	**0.4236**	0.3697	0.3919	0.3665
	P@3	**0.3725**	0.3332	0.3584	0.3212
	P@5	**0.3384**	0.3053	0.3321	0.2885

Table 4.4: Comparing results of our three improvements for AnnexML

		AnnexML-BF			AnnexML
		`E`	`E+P`	`E+A`	`E+P+A`
Learning **E**mbeddings		✓	✓	✓	✓
Learning to **P**artition data		✗	✓	✗	✓
ANN search for prediction		✗	✗	✓	✓
	P@1	**0.9381**	0.9353	0.9374	0.9355
AmazonCat-13K	P@3	0.7835	**0.7853**	0.7809	0.7838
	P@5	0.6320	**0.6353**	0.6289	0.6332
	P@1	**0.8703**	0.8655	0.8690	0.8650
Wiki10-31K	P@3	**0.7448**	0.7431	0.7446	0.7428
	P@5	**0.6454**	0.6435	0.6447	0.6419
	P@1	**0.4670**	0.4660	0.4669	0.4666
Delicious-200K	P@3	0.4018	0.4038	0.4077	**0.4079**
	P@5	0.3705	0.3717	**0.3770**	0.3764
	P@1	0.6108	**0.6378**	0.6024	0.6336
WikiLSHTC-325K	P@3	0.3907	**0.4102**	0.3836	0.4066
	P@5	0.2875	**0.3008**	0.2819	0.2979
	P@1	0.3670	**0.4248**	0.3631	0.4208
Amazon-670K	P@3	0.3190	**0.3698**	0.3161	0.3665
	P@5	0.2862	**0.3309**	0.2834	0.3276

Table 4.5: Average length of the path traversed by a point in the trees that each tree-based method learned. `IdealDepth` is the value when perfectly balanced trees are learned, which is defined as $\log_2(N/\text{MaxInLeaf})$. The values in parentheses are the ratios to `IdealDepth` (values closer to 1 are better).

Dataset	GPT	FastXML	`IdealDepth`
AmazonCat-13K	17.38 (1.03)	17.84 (1.06)	16.86
Wiki10-31K	11.25 (1.07)	11.21 (1.07)	10.47
Delicious-200K	15.57 (1.09)	15.06 (1.06)	14.26
WikiLSHTC-325K	18.29 (1.05)	18.34 (1.05)	17.44
Amazon-670K	16.03 (1.03)	16.29 (1.05)	15.58

Table 4.6: Model sizes of tree-based methods. The number of trees for each method was set to 50.

Dataset	GPT	FastXML	PfastreXML
AmazonCat-13K	**11,310 MB**	17,089 MB	17,294 MB
Wiki10-31K	**205 MB**	522 MB	1102 MB
Delicious-200K	**5,930 MB**	6,766 MB	15,046 MB
WikiLSHTC-325K	22,414 MB	**11,018 MB**	11,585 MB
Amazon-670K	**6,102 MB**	9,019 MB	9,939 MB

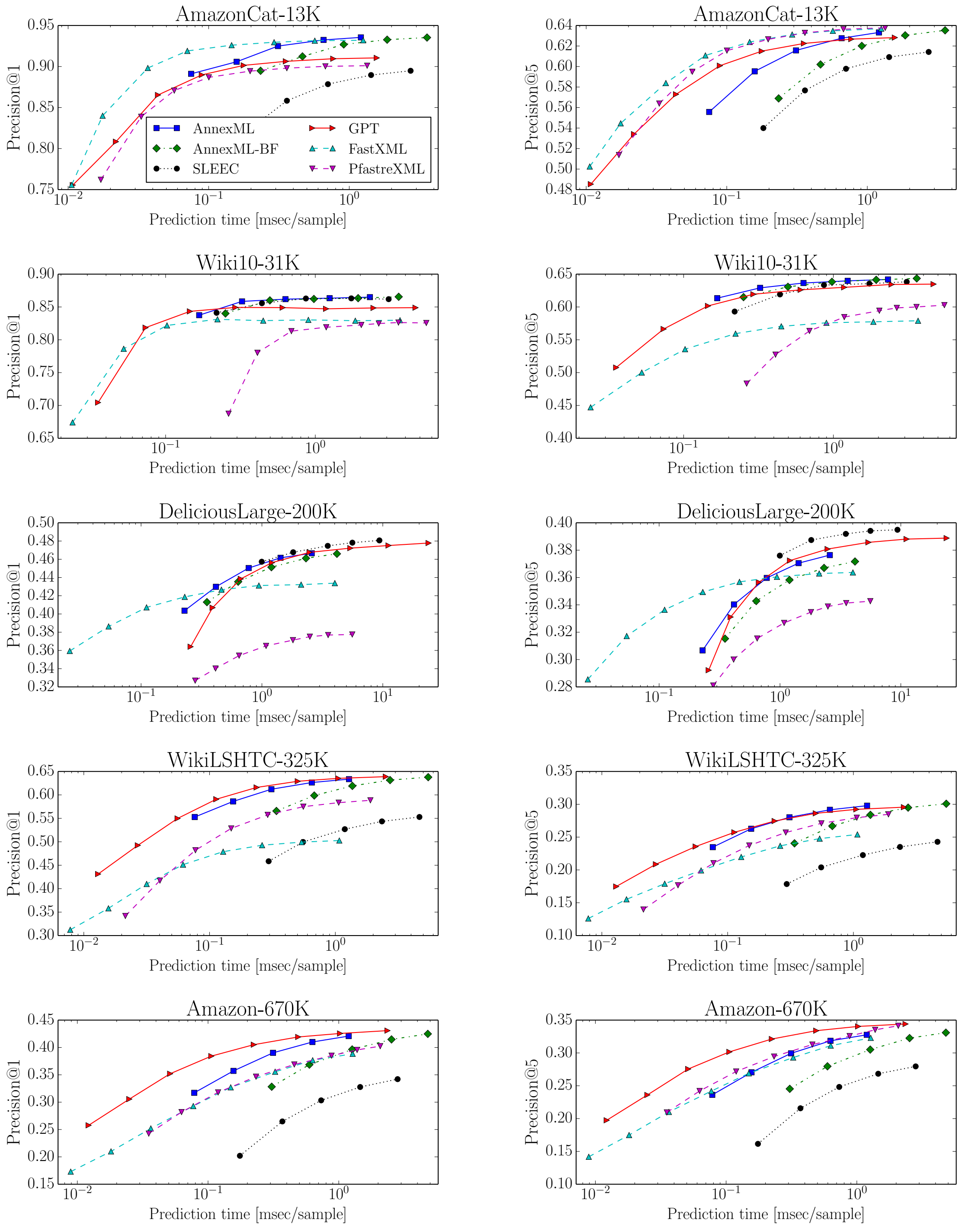


Figure 4.1: Precision versus prediction time when the number of learners changes. Number of learners was {1, 2, 4, 8, 15} for AnnexML, AnnexML-BF, and SLEEC, and {1, 2, 4, 8, 16, 32, 64, 128} for GPT, FastXML, and PfastreXML. This experiment was conducted with C++ implementations on a single CPU thread of the same machine for fair comparison.

Chapter 5

Representation Learning for Users' Web Browsing Sequences

5.1 Introduction

Large-scale Web sites that provide various Web services and mobile apps deal with a lot of user-related prediction tasks, such as news article recommendation [78] and ad click prediction [26, 73]. Feature engineering of user representations is very important to achieve high prediction accuracy for the tasks but is labor-intensive and inefficient for small-scale tasks. On the other hand, logs of user activities on the whole Web site are sufficiently available, such as Web page visits and search queries. For such cases, informative user representations, obtained via leveraging the history of user activities, are useful as features for the prediction tasks. In addition, low-dimensional feature vectors are preferable to high-dimensional sparse vectors for the tasks that have few training data. Figure 5.1 shows an overview of this approach.

In natural language processing (NLP), distributed representations of words in a vector space have received much attention [74]. The studies that use this approach represent words as fixed length dense vectors, whereas the conventional approach treats individual words as unique symbols. These vector representations, which are learned by using various training methods, capture syntactic and semantic word relationships. In addition, some researchers have proposed models to learn vector representations for variable-length pieces of text such as sentences, paragraphs, and documents [62]. In a sentiment analysis task, this approach achieves better results than the conventional word n-gram model and simple averaging of word vectors.

Following these successful techniques, in our work-in-progress paper [100], we proposed an approach that summarizes each sequence of user Web page visits using Paragraph Vector [62], which is an unsupervised method that learns continuous distributed vector representations from pieces of text. In other words, we apply the vector model to sequences

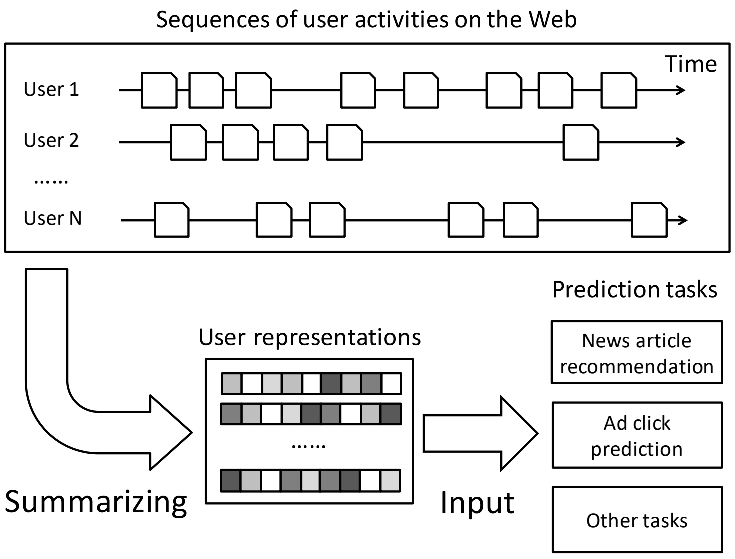


Figure 5.1: Overview of our approach. Low-dimensional vectors, in which sequences of user activities are summarized, are used as common features among various user-related prediction tasks. Note that targets of prediction tasks, such as ad clicks, do not need to be included in sequences.

of user Web page visits, considering users and visits as paragraphs (or documents) and words, respectively. The learned low-dimensional vector representations of users are used among the user-related prediction tasks in common. For each prediction task, an individual classifier or regressor, such as logistic regression, is trained by using these common vectors as features and task-specific users' properties or actions as targets. Note that we do not directly use Paragraph Vector to predict these targets but utilize it to create predictive fix-sized feature vectors from variable-length sequences.

However, do we simply treat the Web page visits data the same way as we treat natural language data? These two types of data are probably generated from different distributions. Therefore, in this study, we first investigate the difference in the distribution between Yahoo! JAPAN access logs and English Wikipedia data. Then, on the basis of the difference, we propose Backward PV-DM, which is a modified version of Paragraph Vector. We report the extensive evaluations as well as the details of the improved methods.

Our main contributions are as follows.

- By comparing our Web page visits data with English Wikipedia data, we show the similarities and differences of frequency distributions between the two types of data. (Section 5.2.1)
- On the basis of the analysis of our Web page visits data, we propose Backward PV-DM. The difference between PV-DM and the proposed model is the context window. (Section 5.4)

Table 5.1: Examples of user activities on the Web.

i	t	$a_{i,t}$
1	1	`https://www.yahoo.co.jp`
	2	`https://weather.yahoo.co.jp/weather/`
	3	`https://www.yahoo.co.jp`
	4	`https://news.yahoo.co.jp/`
	5	`https://news.yahoo.co.jp/hl?c=c_sci`
	6	`https://news.yahoo.co.jp/pickup/6270977`
	7	Session end
2	1	`https://m.yahoo.co.jp/`
	2	`https://m.finance.yahoo.co.jp/`
	3	`https://m.finance.yahoo.co.jp/stock?code=998407.O`
	4	`https://m.finance.yahoo.co.jp/stock?code=4689.T`
	5	Session end
	6	`https://m.yahoo.co.jp/`
	7	`https://auctions.yahoo.co.jp/`
	8	`https://auctions.yahoo.co.jp/reu/project?id=1131`
	9	Session end

- We evaluated our approach using two real-world datasets from an ad network and obtained better results than those of existing methods. (Section 5.5)

5.2 User Activities on the Web

We define A as a set of possible user activities that we consider. For an i-th user u_i, the sequence of activities on the Web is also defined as $(a_{i,1}, a_{i,2}, \ldots, a_{i,T_i})$ where $a_{i,t} \in A$ is the t-th activity of user u_i, and T_i is the size of this sequence.

In this work, we focus on Web page visits and represent each visit $a_{i,t}$ as a URL of the Web page. These URLs are just extracted from logs of Web services. Therefore, this method of representing the data is easy to use and scalable. Another option is to obtain hashed URLs that users have visited in the past via data partners in a similar way to the earlier studies [33, 82] for targeting tasks in display advertising. Thus, our approach is simple and widely applicable. Since we represent each Web page visit as a URL, we use "Web page visit" and "URL" interchangeably. Table 5.1 shows examples of sequences.

Our approach can be easily extended to other types of events such as search queries and ad clicks. Therefore, we describe our approach using the generic activities $a_{i,t}$ in Sections 5.3 and 5.4.

5.2.1 Data Analysis on Web Page Visits

In this section, we reveal the difference between our Web page visit data and English Wikipedia data, since we apply an NLP-based approach to our data.

We collected part of the Yahoo! JAPAN access logs of July 22, 2014 and extracted the URLs of the Web pages that each user visited. These access logs included one of the mobile apps for smartphones and tablet computers as well as ordinary Web services. The data of users whose numbers of Web page visits were between 10 and 1000 were sampled. We discarded URLs that occurred fewer than five times in the extracted data. If the interval of time between two consecutive page visits exceeded 30 minutes, we considered that it was the start of a new session. A session in a sequence of Web page visits corresponds to a sentence in a paragraph or document. Consequently, there were about 3.87 million unique URLs and one billion page visits in the data.

For English Wikipedia data, we preprocessed the latest Wikipedia dump using Matt Mahoney's script[1] and the sentence segmenter in the natural language toolkit (NLTK) [14].

In summary, we obtained two kinds of observations by comparing the data.

- The frequencies of URLs in our Web page visit data follow a power-law distribution. The frequencies of words in English Wikipedia data have the same property, as is widely known [30].
- By focusing on the relative position in a session or sentence, on the other hand, the two distributions of frequencies are significantly different.

The following part elaborates these two observations in detail.

First, the frequencies of URLs and words in the data are shown in Figure 5.2. It is widely known that the frequencies of words in most languages follow a power-law distribution [30]. A power-law distribution looks like a roughly straight line in a log-log plot. Clearly, the plot of Web page visits shows as an approximately straight line[2]. The exponents of the regression lines with power-law distribution are about -1.0. The plot of the Wikipedia data seems to be a piecewise linear function. The exponents of the regression lines are -1.1 for the early part of the data and -1.5 for all of the data. Therefore, the frequencies in both data approximately follow a power-law distribution. However, the tail part of Web page visit data is "fatter" than that of English Wikipedia.

Next, the average of log frequency ratio for relative positions is shown in Figure 5.3.

[1]`http://mattmahoney.net/dc/textdata.html`

[2]A straight line in a log-log plot is a necessary, but not sufficient, condition for the data following a power-law distribution [30]. Data generated by a log-normal distribution also looks roughly straight on the log-log plot.

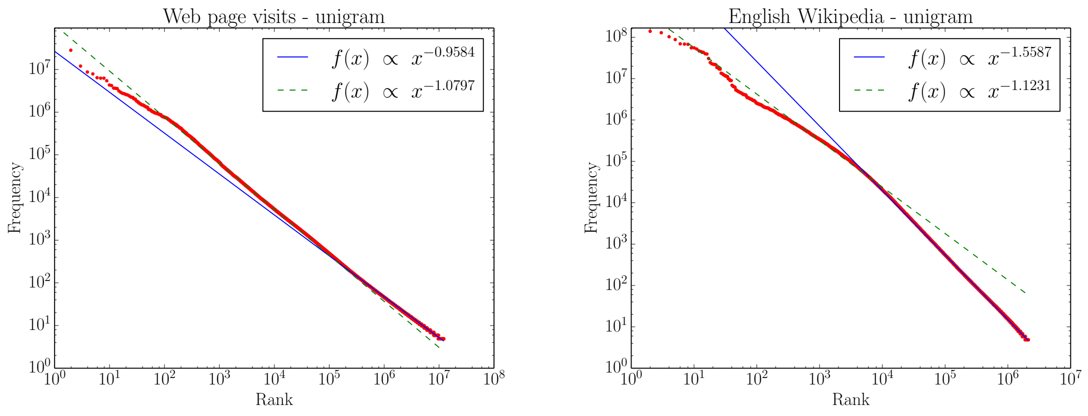


Figure 5.2: Log-log plot for Web page visit data (left) and English Wikipedia data (right). X-axis represents rank of activity or words in frequency table, and y-axis is number of occurrences. Solid and dashed lines represent regression lines for all data and early part of the data (rank less than 10^4), respectively.

The log frequency ratio for relative position k, that is $a_{i,t}$ and $a_{i,t+k}$, is defined as follows:

$$\log\left(\frac{\mathrm{freq}(a_{i,t+k})}{\mathrm{freq}(a_{i,t})}\right),$$

where $\mathrm{freq}(a_{i,t})$ represents the frequency of the Web page visit $a_{i,t}$ (or a word) in the data. We average the log frequency ratio of t and $t+k$ in a session or sentence. The average values of English Wikipedia are around zero, which indicates that word frequencies do not change depending on the position in a sentence. By contrast, the average log frequency ratio of Web page visit data decreases as the relative position k becomes larger. This suggests that URLs that appear in the latter part of a session are the "tail" URLs whereas the URLs that exist in the former part are the "head" URLs. This is caused by a trend of users' Web browsing behavior. As shown in the examples in Table 5.1, most users of Yahoo! JAPAN visit the front page [3] at the beginning of the session and then follow the hyperlinks in the Web pages to move to different sites, such as news, sports, finance, and shopping. Similarly, on each site, users visit the Web pages in which they are interested by following the hyperlinks or using the search engine.

According to the above analysis, to capture the users' interests or preferences suitably, the Web page visits of the "tail" URLs that appear in the latter part of the session are more important.

The above analysis is based on Yahoo! JAPAN access logs. However, we believe that there are access logs of other Web sites that have similar properties since most Web sites have a hierarchical structure similar to Yahoo! JAPAN. Our proposed approach in this study may work well on these sites. On the other hand, more recently developed Web sites, such as social networking sites, show a personalized listing page that consists of

[3] `https://www.yahoo.co.jp/` for PCs and `https://m.yahoo.co.jp` for mobile devices.

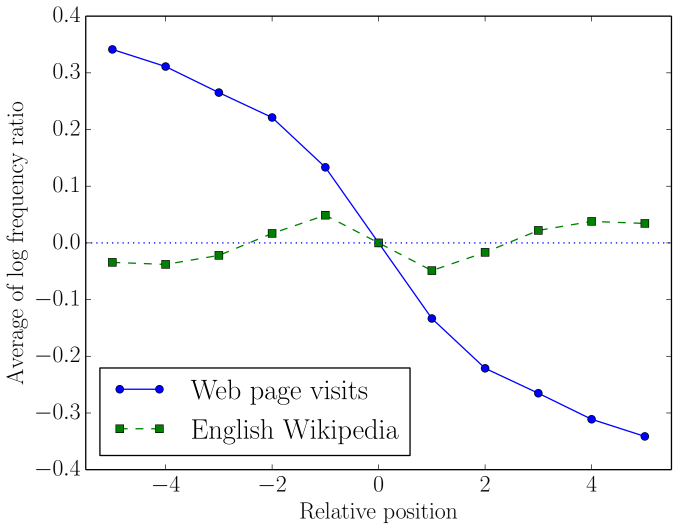


Figure 5.3: Average of log frequency ratio for relative positions. Because of symmetric property of log ratio (y-axis) and relative position (x-axis), these plots are symmetric with respect to origin.

other users' posts and news articles recommended for each user. In this case, most users just move backward and forward between the listing page and posts. Thus, the access logs of these Web sites are regarded as having different properties, so other approaches may be suitable.

5.3 Existing Vector Models

In this section, we describe Paragraph Vector [62] and other vector models [74, 36] for our problem settings. We obtain the vector representations of users and URLs via learning these vector models with users' URL sequences.

5.3.1 PV-DM

We first describe the PV-DM, Distributed Memory Model of Paragraph Vectors [62]. The objective of the vector model for an i-th user u_i's sequence is to maximize the sum of log probabilities:

$$\sum_t \log p(a_{i,t} \mid a_{i,t-1}, \ldots, a_{i,t-s}, u_i),$$

where s is the size of the context window. This means the conditional probability of the activity $a_{i,t}$ given preceding activities $a_{i,t-1}, \ldots, a_{i,t-s}$ and user u_i. The PV-DM defines the probability of this multi-class problem using the softmax function as follows:

$$p(a_{i,t} \mid a_{i,t-1}, \ldots, a_{i,t-s}, u_i) := \frac{\exp(\boldsymbol{w}_{a_{i,t}}^{\mathrm{T}} \boldsymbol{v}_I)}{\sum_{a \in A} \exp(\boldsymbol{w}_a^{\mathrm{T}} \boldsymbol{v}_I)}, \tag{5.1}$$

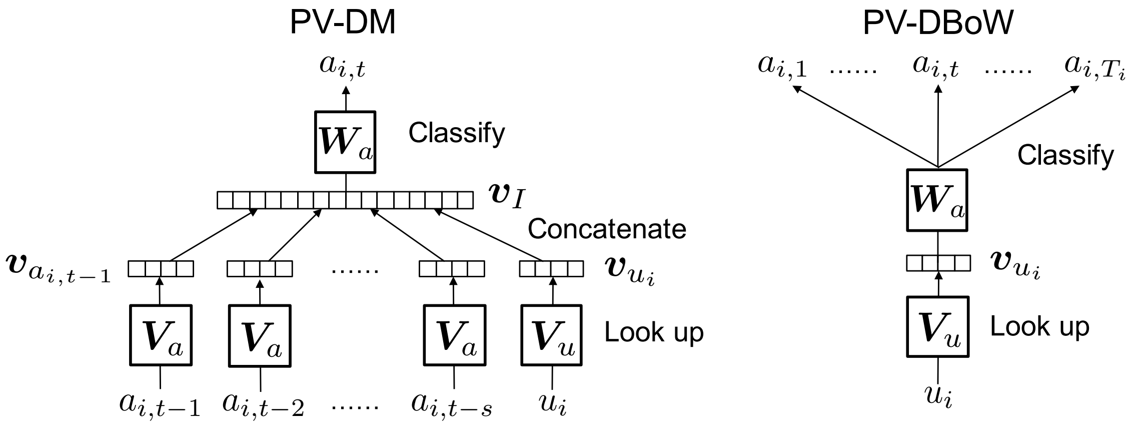


Figure 5.4: Overviews of PV-DM (left) and PV-DBoW (right) architectures

where $\boldsymbol{w}_{a_{i,t}}$ is the "output" vector corresponding to $a_{i,t}$, and $\boldsymbol{v}_I$ is the "input" vector corresponding to the previous activities $a_{i,t-1}, \ldots, a_{i,t-s}$ and user u_i. We also define the "input" activity vector corresponding to $a_{i,t}$ as $\boldsymbol{v}_{a_{i,t}}$ and user "input" vector as $\boldsymbol{v}_{u_i}$. Therefore, $\boldsymbol{v}_I$ is represented as a concatenated vector:

$$\boldsymbol{v}_I = [\boldsymbol{v}_{a_{i,t-1}}^{\mathrm{T}}, \ldots, \boldsymbol{v}_{a_{i,t-s}}^{\mathrm{T}}, \boldsymbol{v}_{u_i}^{\mathrm{T}}]^{\mathrm{T}}.$$

For the case of $j \leq 0$, an input activity vector $\boldsymbol{v}_{a_{i,j}}$ is replaced with a special padding vector $\boldsymbol{v}_{NULL}$. We define the size of input activity vector as $\boldsymbol{v}_{a_{i,j}} \in \mathbb{R}^{v_a}$ and the size of the input user vector as $\boldsymbol{v}_{u_i} \in \mathbb{R}^{v_u}$, so both the sizes of the input vector and output vector are represented as $\boldsymbol{v}_I, \boldsymbol{w}_{a_{i,j}} \in \mathbb{R}^{sv_a+v_u}$.

The overview of the model architecture is shown in Figure 5.4 (left). The matrices $\boldsymbol{V}_a$, $\boldsymbol{V}_u$ and $\boldsymbol{W}_a$ consist of the vectors $\boldsymbol{v}_{a_{i,t}}$, $\boldsymbol{v}_u$, and $\boldsymbol{w}_{a_{i,t}}$, respectively. In other words, the vectors correspond to the row vectors in the matrices. We first randomly initialize the matrices, learn them, and then obtain the vectors by using a procedure described later in Section 5.4.2.

The user vector $\boldsymbol{v}_{u_i}$ is used as a feature vector of various user-related prediction tasks, such as ad click prediction. We also use the "input" activity vectors $\boldsymbol{v}_{a_{i,j}}$ as features and show their effectiveness in the experiment.

5.3.2 PV-DBoW

The PV-DBoW, Distributed Bag of Words version of Paragraph Vector, is another version of Paragraph Vector [62]. The objective of the PV-DBoW for an i-th user u_i's sequence is to maximize the sum of log probabilities:

$$\sum_t \log p(a_{i,t} \mid u_i).$$

The probability of this multi-class problem is also defined using the softmax function as follows:

$$p(a_{i,t} \mid u_i) := \frac{\exp(\boldsymbol{w}_{a_{i,t}}^{\mathrm{T}} \boldsymbol{v}_{u_i})}{\sum_{a \in A} \exp(\boldsymbol{w}_a^{\mathrm{T}} \boldsymbol{v}_{u_i})}. \tag{5.2}$$

For PV-DBoW, the input user vector $\boldsymbol{v}_{u_i} \in \mathbb{R}^{v_u}$ and output word vector $\boldsymbol{w}_{a_{i,j}} \in \mathbb{R}^{v_u}$ are the same size. The overview of the model architecture is also presented in Figure 5.4 (right).

PV-DBoW can be viewed as a simplified version of PV-DM where the size of the context window s is zero.

5.3.3 CBoW and Skip-gram

For comparison with the above Paragraph Vectors, we also describe word vector models, CBoW and Skip-gram model [74].

Similar to Paragraph Vectors, the objective of CBoW (continuous bag of words model) and Skip-gram is also to maximize the sum of log probabilities, which is defined using the softmax function. However, these two vector models are proposed for obtaining word representation. Therefore, in our problem settings, these models just provide the representations for activities, not for users directly.

The objective function of the CBoW is defined as follows:

$$\sum_t \log p(a_{i,t} \mid a_{i,t-s}, \ldots, a_{i,t-1}, a_{i,t+1}, \ldots, a_{i,t+s}).$$

$$p(a_{i,t} \mid a_{i,t-s}, \ldots, a_{i,t-1}, a_{i,t+1}, \ldots, a_{i,t+s}) := \frac{\exp(\boldsymbol{w}_{a_{i,t}}^{\mathrm{T}} \boldsymbol{v}_I)}{\sum\limits_{a \in A} \exp(\boldsymbol{w}_a^{\mathrm{T}} \boldsymbol{v}_I)},$$

where $\boldsymbol{v}_I$ is the averaged vector of the context vectors:

$$\boldsymbol{v}_I = \frac{1}{2s} \sum_{-s \le k \le s, k \ne 0} \boldsymbol{v}_{a_{i,t+k}}.$$

On the other hand, the objective function of the Skip-gram model is as follow:

$$\sum_t \sum_{-s \le k \le s, k \ne 0} \log p(a_{i,t+k} \mid a_{i,t}),$$

$$p(a_{i,t+k} \mid a_{i,t}) := \frac{\exp(\boldsymbol{w}_{a_{i,t+k}}^{\mathrm{T}} \boldsymbol{v}_{a_{i,t}})}{\sum_{a \in A} \exp(\boldsymbol{w}_a^{\mathrm{T}} \boldsymbol{v}_{a_{i,t}})}. \tag{5.3}$$

Figure 5.5 shows the overviews of these model architectures.

The Directed Skip-gram model proposed by Djuric et al. [36] is a modified model that considers the future activities given by the past activity:

$$\sum_t \sum_{0 < k \le s} \log p(a_{i,t+k} \mid a_{i,t}).$$

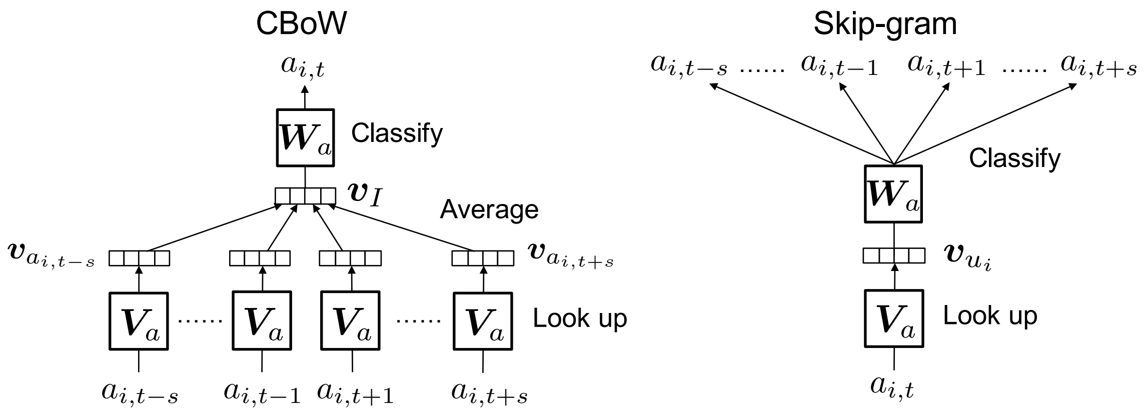


Figure 5.5: Overview of CBoW (left) and Skip-gram (right) architectures

5.4 Proposed Method

In this section, we propose Backward PV-DM. Then, we explain the learning method for these vector models.

5.4.1 Backward PV-DM

On the basis of the analysis of our Web page visit data in Section 5.2.1, the Web page visits of "tail" URLs, which appear in the latter part of the session, are more important to capture the users' interests or preferences suitably. Thus, we propose a modified model called Backward PV-DM. The difference between PV-DM and this model is the context window. The objective of the Backward PV-DM is to maximize the sum of log probabilities:

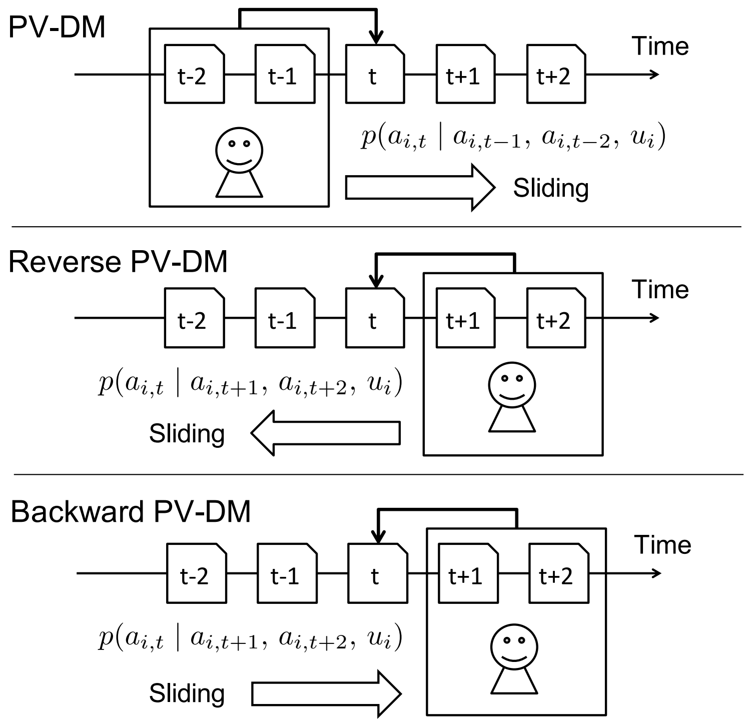
$$\sum_t \log p(a_{i,t} \mid a_{i,t+1},\, \ldots,\, a_{i,t+s},\, u_i).$$

For predicting "output" activity $a_{i,t}$, Backward PV-DM uses the following activities $a_{i,t+1},\, \ldots,\, a_{i,t+s}$ as "input" whereas PV-DM uses the previous activities $a_{i,t-1},\, \ldots,\, a_{i,t-s}$. The conditional probability is defined as follows:

$$p(a_{i,t} \mid a_{i,t+1},\, \ldots,\, a_{i,t+s},\, u_i) := \frac{\exp(\boldsymbol{w}_{a_{i,t}}^{\mathrm{T}} \boldsymbol{v}_I)}{\sum_{a \in A} \exp(\boldsymbol{w}_a^{\mathrm{T}} \boldsymbol{v}_I)}, \tag{5.4}$$

$$\boldsymbol{v}_I = [\boldsymbol{v}_{a_{i,t+1}}^{\mathrm{T}},\, \ldots,\, \boldsymbol{v}_{a_{i,t+s}}^{\mathrm{T}},\, \boldsymbol{v}_{u_i}^{\mathrm{T}}]^{\mathrm{T}}.$$

The above modification of the context window encourages the storage of the information of URLs that appear in the latter part of a session in the user vector. Equation 5.4 is considered as a prediction for a URL of current time step $a_{i,t}$ given user u_i and following URLs $a_{i,t+1}, \ldots, a_{i,t+s}$. For the former part of a session, the prediction is relatively easier because future URLs can be used. For example, in Table 5.1, a prediction of $a_{2,2}$ given $a_{2,3}$ is not difficult. Both URLs have the same domain (`m.finance.yahoo.co.jp`), and $a_{2,3}$ is



PV-DM
t-2
t-1
t
t+1
t+2
Time
$p(a_{i,t} \mid a_{i,t-1}, a_{i,t-2}, u_i)$
Sliding
Reverse PV-DM
t-2
t-1
t
t+1
t+2
Time
$p(a_{i,t} \mid a_{i,t+1}, a_{i,t+2}, u_i)$
Sliding
Backward PV-DM
t-2
t-1
t
t+1
t+2
Time
$p(a_{i,t} \mid a_{i,t+1}, a_{i,t+2}, u_i)$
Sliding

Figure 5.6: Illustration of PV-DM, Reverse PV-DM, and Backward PV-DM where size of context window s is two. Differences between these models are conditional probability to be maximized and sliding direction of context window.

more specific than $a_{2,2}$. Thus, user vector $\boldsymbol{v}_{u_2}$ does not need to memorize the information of $a_{2,2}$. The user vector in the model acts as a memory that remembers what is missing from the current context. On the other hand, for the latter part of a session, the prediction is more difficult. For example, a prediction of $a_{1,6}$ given $a_{1,7}$ is not easy because $a_{1,7}$ (Session end) has less information to predict $a_{1,6}$. Therefore, user vector $\boldsymbol{v}_{u_2}$ needs to incorporate the information of $a_{1,6}$ for the prediction. In this way, the modified objective function encourages the user vector to store the information of URLs that appear in the latter part of a session. On the other hand, plain PV-DM, which is based on Equation 5.1, is likely to memorize the information of URLs in the former part of a session.

We also present Reverse PV-DM whose input sequences are just reversed from future to past. Therefore, the conditional probabilities of Reverse PV-DM and Backward PV-DM are the same, but the sliding directions of the context window are different. The differences between PV-DM, Reverse PV-DM, and Backward PV-DM are summarized in Figure 5.6.

The sliding direction of the context window does not change the whole objective to be maximized. However, this objective is not concave because of the bilinear form, and we search for a better local maximum of the objective using a stochastic gradient descent (SGD) as described in Section 5.4.2. In our implementation, the sliding direction is the same as the input order of the SGD procedure. Since the latter input is more memorable than the former input, the sliding direction and input order affect the quality of the user vector. In other words, the informative Web page visits that occur in the latter part of a session should be inputted last. The experimental results present the effect.

For comparison with Backward PV-DM, we use Backward Skip-gram in the experiment, which is a reversed version of Directed Skip-gram:

$$\sum_{t} \sum_{-s \leq k < 0} \log p(a_{i,t+k} \mid a_{i,t}) = \sum_{t=1}^{T_i} \sum_{0 < k \leq s} \log p(a_{i,t-k} \mid a_{i,t}).$$

5.4.2 Learning vector models

Equations (5.1) – (5.4) and their first derivative are impractical to compute because the number of unique activities $|A|$ is typically large. Le and Mikolov [62] originally used hierarchical softmax with a Huffman binary tree on the basis of word frequencies for fast training. Here, instead of hierarchical softmax, we use a negative sampling approach [74]. Hence an alternate objective to $\log p(a_{i,t} \mid a_{i,t-1}, \ldots, a_{i,t-s}, u_i)$ with Eq. (5.2) is defined as:

$$\log \sigma(\boldsymbol{w}_{a_{i,t}}^{\mathrm{T}} \boldsymbol{v}_I) + k \cdot \mathbb{E}_{a_n \sim p_n(a)} \left[\log \sigma(-\boldsymbol{w}_{a_n}^{\mathrm{T}} \boldsymbol{v}_I) \right],$$

where $\sigma(z) = 1/(1+\exp(-z))$ is a sigmoid function, k is the number of randomly sampled negative instances, and $p_n(a)$ is noise distribution generating negative instances. We use the "unigram" distribution $U(a)$ raised to the 3/4th power as $p_n(a)$ in the same way as Mikolov et al. [74] did. We train the model using asynchronous SGD [87] with AdaGrad [39]. In the inference step for new users, the user vectors $\boldsymbol{v}_u$ are learned while input and output activity vectors $\boldsymbol{v}_a$ and $\boldsymbol{w}_a$ are fixed.

5.5 Experiments

In this section, we evaluated our approach using two real-world datasets from the Web services of Yahoo! JAPAN.

5.5.1 Datasets

We evaluated the proposed method using two supervised learning datasets: *AdClicker* and *SiteVisitor*. *AdClicker* consists of the users who clicked contextual ads that are included in the five selected ad campaigns. Similarly, *SiteVisitor* consists of the users who visited Web sites of the five selected advertisers.

In real application settings, sequences of user activities are typically summarized on a daily basis. Thus, we created these two datasets in view of predicting users' particular actions on a day on the basis of the history of Web pages visited the previous day. The

Table 5.2: Statistics for two datasets. #Features is the number of unique URLs that occurred five or more times in each dataset.

Dataset	#Train	#Validation	#Test	#Features
AdClicker	51,576	10,000	10,000	66,957
SiteVisitor	1,862,693	20,000	20,000	1,219,850

training and validation sets were generated from logs of July 22 and 23, 2014. Web page visits on the former day are used as features, and the target activity in the latter day is treated as labels. Similarly, a test set was generated from logs of July 23 and 24, 2014, as features and labels, respectively. Since these features were extracted from Web service logs of Yahoo! JAPAN, they are only a small fraction of the entire user activities on the Web. These features do not include visits to advertisers' sites, which are the labels of *SiteVisitor*, as well as ad clicks.

The contextual ads displayed in *AdClicker* are determined by the Web page content as well as user information. Therefore, learning each Web page representation is also helpful for this task. On the other hand, *SiteVisitor* is the dataset based on more active user interests.

The statistics for datasets are summarized in Table 5.2.

5.5.2 Evaluation settings

AdClicker and *SiteVisitor* are multi-label datasets because a user can click more than one ad or visit various advertisers' sites. In the experiment, we transformed the multi-label problem into a set of binary classification problems. We represent the binary classification tasks for *AdClicker* as Ac1 to Ac5 and for *SiteVisitor* as Sv1 to Sv5.

For each binary classification task, we trained logistic regression classifiers using features extracted by each method. The evaluation measure is area under receiver operating characteristic (ROC) Curve (AUC).

5.5.3 Proposed methods and baselines

We compared the methods using Paragraph Vector with some baselines. *Bin* and *Freq* are simple baselines that use raw URLs as features. *Freq* takes into account the frequencies of the user's site visits, whereas *Bin* considers only whether a user visits the Web page or not (binary features). The feature vectors of these two methods are high dimensional sparse vectors.

We refer to CBoW, Skip-gram, Directed Skip-gram, and Backward Skip-gram as word vector models. We also refer to PV-DM, Reverse PV-DM, Backward PV-DM, and PV-DBoW as Paragraph Vectors. By using the word vectors models, a user is represented as

Table 5.3: Method summary

Method	Dimension	Density	#Models
Bin	High	Sparse	–
Freq	High	Sparse	–
CBoW	400	Dense	1
Skip-gram	400	Dense	1
Directed Skip-gram	400	Dense	1
Backward Skip-gram	400	Dense	1
PV-DM	400	Dense	1
Reverse PV-DM	400	Dense	1
Backward PV-DM	400	Dense	1
PV-DBoW	400	Dense	1
PV-DM(both)	800	Dense	1
Backward PV-DM(both)	800	Dense	1
PV-DM+Skip-gram	800	Dense	2

the simple averaging of input activity vectors $\boldsymbol{v}_a$ in the sequence, which is similar to the approach of Djuric et al. [36]. We use the user vectors $\boldsymbol{v}_u$ in Paragraph Vectors as user representations. These methods using the vector models are represented in *italics*. For example, the proposed method using the PV-DM model is represented as *PV-DM*.

For PV-DM and Backward PV-DM, we also use the averaging of input activity vectors $\boldsymbol{v}_a$ in the same way as for the word vector models. We concatenated the user vectors and the averaged vector for the input of prediction tasks. These methods are called *PV-DM(both)* and *Backward PV-DM(both)*. In addition, we evaluated a method that uses the concatenated vectors learned by the PV-DM and Skip-gram models. This method is called *PV-DM+Skip-gram*.

The settings of learning the vector models are as follows: the size of input vectors $v_a = v_u = 400$, the size of context window $s = 5$, the number of randomly sampled negative instances $k = 5$, and the number of epochs (full pass through the data) is five. For Paragraph Vectors, we create the user vectors $\boldsymbol{v}_u$ via an inference step, considering all the users as new users. Because of the stochastic behavior of asynchronous SGD and random initialization, we report the mean value of five runnings for the methods using vector models.

The methods we described above are summarized in Table 5.3. Note that each vector model is learned by only using sequences of users' activities extracted from logs of July 22, 2014. Therefore, any label information of the prediction tasks is not included in these models.

5.5.4 Results

The experimental results are summarized in Table 5.4. The **bold** elements indicate the best performance of the methods. The underlined scores are the best results of the word vector models and Paragraph Vectors.

The proposed *Backward PV-DM* achieved statistically significant improvements over *PV-DM* on almost all datasets. In addition, the results of *Backward PV-DM* are better than those of *Reverse PV-DM*. The difference between these models is just the sliding direction of the context window, in other words, the input order of the SGD procedure. However, since the Web page visits that appear in the latter part of a session have more information on the user's interests, the direction and input order are important to improve the quality of the user vectors, which can act as a memory of the interests. On the other hand, the results of *Backward Skip-gram* are almost the same as or slightly worse than those of *Skip-gram* and *Directed Skip-gram*.

Backward PV-DM and *PV-DM* achieved better results than *Skip-gram* in *SiteVisitor* whereas the opposite trend is shown in *AdClicker* because of the difference between the two datasets as described in Section 5.5.1. For *AdClicker* datasets, displayed ads are determined by using Web page information as well as user information. In addition, users passively see the ads and click them only if they are interested in them. Thus, learning Web page representations $\boldsymbol{v}_{a_{i,t}}$ is also helpful for these tasks. On the other hand, Paragraph Vectors consistently performed superiorly on the *SiteVisitor* dataset. This dataset is just based on users' active interests since users willingly visit the advertisers' Web sites. Therefore, Paragraph Vectors achieved better results. Note that the proposed *Backward PV-DM* achieved statistically significant improvements over *PV-DM* on the *AdClicker* dataset as well. Thus, our proposed improvement for PV-DM, which was inspired by data analyses, worked effectively.

Backward PV-DM(both) achieved the best results in seven of ten tasks. Since this method utilizes both user and URL vectors obtained by Backward PV-DM training, it achieved statistically significant improvements over *Skip-gram*, even on three of five *AdClicker* tasks. As described above, *Skip-gram* achieved better results than plain *Backward PV-DM* for this dataset. *Backward PV-DM(both)* is based on the vectors obtained from Backward PV-DM whereas *PV-DM+Skip-gram* needs to train both PV-DM and Skip-gram models. Thus, *Backward PV-DM(both)* is easy to train.

To summarize the above results, *Backward PV-DM* outperformed plain *PV-DM*, and *Backward PV-DM(both)* which utilizes both user and URL vectors achieved the best results in most tasks.

5.5.5 Effect of the data size

As described in Section 5.1, we focus on achieving better prediction accuracy for the small-scale tasks, leveraging the history of user activities. The experimental results when changing train data size of *AdClicker* and *SiteVisitor* are shown in Figure 5.7. Compared with *Bin* and *Freq*, whose features are high-dimensional sparse vectors, the vector models achieve the better results when the size of training data are small. Since the vector models learn the low-dimensional dense vector, they have an advantage in this case.

We also evaluate whether the prediction accuracy is improved when Web page visit data are bigger. Figure 5.8 shows the experimental results when changing the number of Web page visits in learning the vector models. The results of some tasks show improvements. In other words, the prediction accuracies for user-related tasks can be improved by simply increasing the data when learning the vector models. The degree of the improvements is relatively small. However, since the learned user representations are used among the user-related prediction tasks in common, it is not a small effect for all small-scale tasks.

5.6 Related Work

In the online advertising field, some previous works focused on finding the user segments that might be interested in a given advertiser's products, inferred from web-browsing behavior information. These approaches are known as behavioral targeting [5] or conversion optimization [68, 82]. Advertisers increase the effectiveness of advertising to deliver their ads to the audience found by the approaches.

Perlich et al. [82] presented a transfer learning approach for online display targeting. In the first stage of the approach, users are represented as a bag-of-words representation of the users browsing history, with each URL hashed into its own binary feature. The *Bin* method, which is compared with the proposed method in Section 5.5, is similar to this approach.

Djuric et al. [36] proposed an approach for improving estimation of ad click or conversion probability on the basis of a sequence of a user's online actions modeled using the Hidden Conditional Random Fields (HCRF) model [84]. To address the sparsity problem at the input side of the HCRF model, they proposed a directed version of the Skip-gram model, which maximizes the log-probabilities of future activities given users' preceding activities. Input "words" of the Directed Skip-gram model consist of entities found on a Web page visited by the user and tokens in search queries.

Okura et al. [78] tackled news article recommendation tasks. They generated user representations by using Recurrent Neural Networks (RNNs) with browsing histories as input sequences. Therefore, this approach is similar to ours. However, they learned task-specific RNNs by using users' clicks as labels whereas we simply learned common vector models for various user-related prediction tasks in an unsupervised manner.

White et al. [109] focused on Web site recommendations and studied the effectiveness of various sources of contextual information for user interest modeling. Similar to ours, their method modeled user interests by using the URLs that the user visited. However, they represented each Web page as pre-defined Open Directory Project (ODP) categories by using trained classifiers, and these aggregated categories are regarded as user interests. On the other hand, we obtained both the URL and user vectors simultaneously by learning the vector models from sequences of user visits without supervision. In addition, they just aimed at recommending Web pages to each user whereas we intend to use learned low-dimensional vectors among the user-related prediction tasks in common.

Yan et al. [112] studied Web caching and prefetching. They improved these policies by mining frequent access patterns of Web documents and building association-based prediction models. Therefore, their objective is predicting users' upcoming Web accesses. On the other hand, our study focuses on extracting common user representations from sequences of Web page visits via prediction.

For an English-to-French translation task, Sutskever et al. [99] reported that the reversed input of the words in the source sentence when using a Long Short-Term Memory (LSTM) model achieved the better results. This technique is related to our discussion of the difference between Reverse PV-DM and Backward PV-DM.

For obtaining continuous word representations, Bojanowski et al. [15] proposed an approach where each word is represented as a bag of character n-grams. A vector representation of each character n-gram is learned using a large corpus, and each word is represented as the sum of these vectors. The authors showed that their word vectors achieved state-of-the-art performance on several word similarity and analogy tasks. This idea may be applied to our approach by representing each URL as a bag of substrings, such as domain names.

5.7 Conclusion

In this study, we presented an approach that summarizes each sequence of user Web page visits using the Paragraph Vector, considering users and Web page visits as paragraphs and words, respectively. The learned user representations are used among the user-related prediction tasks in common, such as news article recommendation and ad click prediction. In addition, on the basis of the analysis of our Web page visit data, we proposed Backward PV-DM, which is a modified version of Paragraph Vector. We evaluated this approach on two ad-related datasets based on logs from Yahoo! JAPAN Web services. The experimental results demonstrated its effectiveness.

Table 5.4: Experimental results. Values are AUC. We report mean values of five runnings with different random initialization for methods using vector models (see Section 5.5.3 for more details). $\diamondsuit$ and $\clubsuit$ indicate statistically significant improvements (p-value < 0.05) over **Skip-gram** and **PV-DM**, respectively.

	AdClicker				
	Ac1	Ac2	Ac3	Ac4	Ac5
Bin	0.9753	0.8063	**0.6641**	0.7052	0.7524
Freq	0.9814	0.8184	0.6580	0.6961	0.7509
CBoW	$0.9903^{\clubsuit}$	$0.8323^{\clubsuit}$	$0.6533^{\clubsuit}$	0.7154	$0.7700^{\clubsuit}$
Skip-gram	$\underline{0.9906}^{\clubsuit}$	$0.8354^{\clubsuit}$	$\underline{0.6562}^{\clubsuit}$	0.7163	$\underline{0.7725}^{\clubsuit}$
Directed Skip-gram	$0.9904^{\clubsuit}$	$\underline{\mathbf{0.8374}}^{\clubsuit}$	0.6533	0.7159	$0.7706^{\clubsuit}$
Backward Skip-gram	$0.9905^{\clubsuit}$	$0.8328^{\clubsuit}$	0.6525	0.7138	$0.7712^{\clubsuit}$
PV-DM	0.9899	0.8151	0.6483	$0.7242^{\diamondsuit}$	0.7633
Reverse PV-DM	0.9884	$0.8263^{\clubsuit}$	0.6481	$0.7274^{\diamondsuit}$	0.7618
Backward PV-DM	$0.9902^{\clubsuit}$	$0.8247^{\clubsuit}$	$0.6537^{\clubsuit}$	$\underline{0.7345}^{\diamondsuit\clubsuit}$	$0.7661^{\clubsuit}$
PV-DBoW	0.9894	$0.8288^{\clubsuit}$	0.6507	$0.7290^{\diamondsuit}$	0.7581
PV-DM(both)	$0.9910^{\clubsuit}$	0.8193	$0.6531^{\clubsuit}$	$0.7379^{\diamondsuit\clubsuit}$	$0.7704^{\clubsuit}$
Backward PV-DM(both)	$\mathbf{0.9914}^{\diamondsuit\clubsuit}$	$0.8281^{\clubsuit}$	$0.6575^{\clubsuit}$	$\mathbf{0.7463}^{\diamondsuit\clubsuit}$	$\mathbf{0.7760}^{\diamondsuit\clubsuit}$
PV-DM+Skip-gram	$0.9912^{\diamondsuit\clubsuit}$	$0.8358^{\clubsuit}$	$0.6622^{\diamondsuit\clubsuit}$	$0.7391^{\diamondsuit\clubsuit}$	$0.7752^{\diamondsuit\clubsuit}$

	SiteVisitor				
	Sv1	Sv2	Sv3	Sv4	Sv5
Bin	0.7619	0.8188	0.7087	0.7920	0.7292
Freq	0.7821	0.8163	0.7006	0.7781	0.7256
CBoW	0.7999	0.8277	0.7067	0.7849	0.7339
Skip-gram	0.8017	0.8328	0.7135	0.7931	0.7417
Directed Skip-gram	0.8019	0.8308	0.7120	0.7914	0.7394
Backward Skip-gram	0.8018	0.8307	0.7125	0.7909	0.7388
PV-DM	$0.8051^{\diamondsuit}$	$0.8343^{\diamondsuit}$	$0.7180^{\diamondsuit}$	$0.7964^{\diamondsuit}$	$0.7479^{\diamondsuit}$
Reverse PV-DM	0.8011	$0.8343^{\diamondsuit}$	$0.7207^{\diamondsuit\clubsuit}$	$0.7992^{\diamondsuit\clubsuit}$	$0.7488^{\diamondsuit}$
Backward PV-DM	$\underline{0.8092}^{\diamondsuit\clubsuit}$	$\underline{0.8366}^{\diamondsuit\clubsuit}$	$\underline{0.7222}^{\diamondsuit\clubsuit}$	$\underline{0.8028}^{\diamondsuit\clubsuit}$	$\underline{0.7491}^{\diamondsuit}$
PV-DBoW	0.7965	0.8294	$0.7198^{\diamondsuit\clubsuit}$	0.7945	$0.7489^{\diamondsuit}$
PV-DM(both)	$0.8134^{\diamondsuit\clubsuit}$	$0.8373^{\diamondsuit\clubsuit}$	$0.7229^{\diamondsuit\clubsuit}$	$0.7998^{\diamondsuit\clubsuit}$	$0.7506^{\diamondsuit\clubsuit}$
Backward PV-DM(both)	$\mathbf{0.8162}^{\diamondsuit\clubsuit}$	$\mathbf{0.8396}^{\diamondsuit\clubsuit}$	$\mathbf{0.7276}^{\diamondsuit\clubsuit}$	$\mathbf{0.8069}^{\diamondsuit\clubsuit}$	$0.7513^{\diamondsuit\clubsuit}$
PV-DM+Skip-gram	$0.8128^{\diamondsuit\clubsuit}$	$0.8396^{\diamondsuit\clubsuit}$	$0.7252^{\diamondsuit\clubsuit}$	$0.8026^{\diamondsuit\clubsuit}$	$\mathbf{0.7531}^{\diamondsuit\clubsuit}$

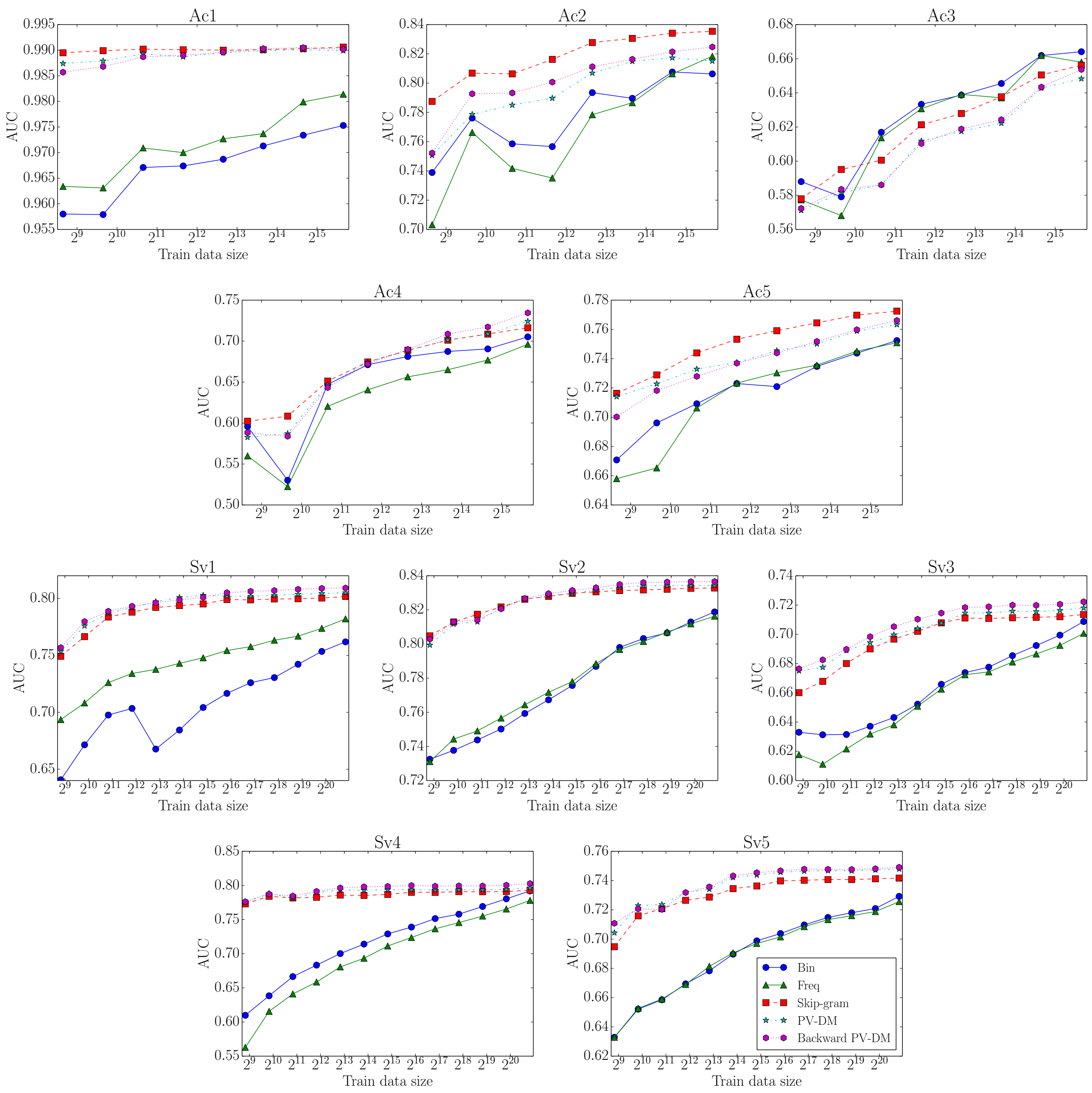


Figure 5.7: Experimental results when changing training data size. Horizontal axis (train data size) is logarithmic scale.

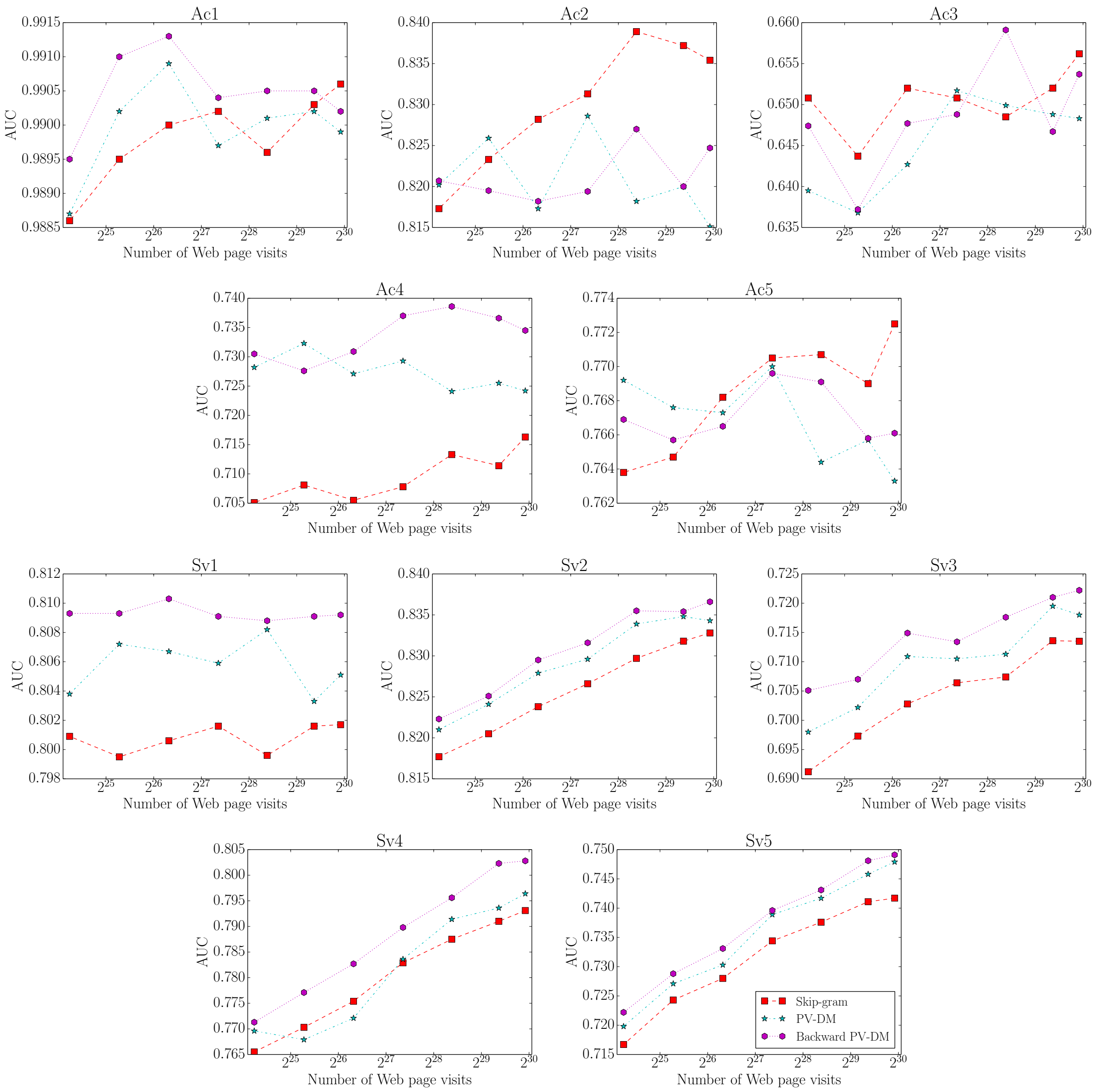


Figure 5.8: Experimental results when changing the number of Web site visits in learning the vector models. Horizontal axis (number of Web site visits) is logarithmic scale.

Chapter 6

Conclusion

Recommender systems have become ubiquitous on the today's World Wide Web and are utilized by various Web sites, such as Facebook, Instagram, Twitter, and Youtube, to help users discover personalized content. These real-world systems need to deal with scalability challenges caused by a huge number of users and candidate items while there is much user and context information that can be used for recommendation.

In this thesis, we focused on two main challenges for constructing "practical Web-scale recommender systems":

1. Selecting suitable items for each user in a very short time
2. Making valuable users' representations from users' activity histories

For the former challenge, we developed an inverted index-based retrieval system for contextual advertising in Chapter 3. We applied this approach with a real ad serving system and conducted A/B testing for evaluating the online performance. Our approach achieved significant improvements over the existing production system. In Chapter 4, we tackled extreme multi-label classification and proposed two methods that enable faster and more accurate predictions to be made by utilizing tree and graph-based index. Experimental results on several large-scale public datasets showed that our methods improve the trade-off between prediction and accuracy. At the same level of accuracy, the prediction time of our classifiers was up to 58 times shorter than that of a recent state-of-the-art method (0.08 vs. 4.66 milliseconds per test point).

For the latter challenge, in Chapter 5, we presented a method for learning representations of users' Web browsing sequences in an unsupervised manner on the basis of analysis of our real-world Web visits data. The learned users' representations, which are represented as low-dimensional vectors, are used among the user-related prediction tasks in common. For each prediction task, an individual classifier or regressor is trained by using the common vectors as features and task-specific users ' properties or actions as targets.

Our proposed method achieved better results than existing methods on two ad-related prediction tasks based on logs from large-scale Web services. In addition, the prediction accuracies of some tasks were successfully improved by simply increasing the data size of users' Web browsing sequences as the training data sizes of prediction tasks themselves were not changed.

Our findings from these studies are summarized as follows:

- For making more favorable recommendations in a very limited time, it is promising to construct a machine learning model suitable for an efficient retrieval system (Chapter 3) or a model that includes a search index (Chapter 4).
- To capture the users' interests or preferences from their Web browsing sequences suitably, it is important to construct a model in consideration of their appearance positions in a session (Chapter 5).

We hope that these findings will help to construct scalable recommender systems more efficiently and that these systems will enrich the people's daily lives.

6.1 Future Work

Although we proposed approaches to construct "practical Web-scale recommender systems" more efficiently, a lot of problems remain as future work.

First, we want to develop models suitable for efficient retrieval on computing devices other than common CPU. New computing devices such as GPU have also been developed in parallel with the improvement in neural networks. By learning models in the form where these devices efficiently computes, fast retrieval can be done. Shan et al. [89] used an approach with exhaustive search on GPUs instead of using some pruning techniques or approximate nearest neighbor search methods on CPUs. They proposed the Recurrent Binary Embedding (RBE) model, which learns compact representations suitable for real-time retrieval. Top-k items are retrieved quickly by computing bit-wise operations on GPUs in parallel. We will explore this research direction in future work.

There are some studies that construct efficient index structures by considering target data distribution. Kraska et al. [61] claimed that all existing structures, such as B-trees, hash maps and Bloom filters, can be replaced with machine learning models. For example, bloom filters are regarded as binary classifiers to determine whether a target key exists in a set. AnnexML presented in Chapter 4 can be regarded as search index-included classifiers. Therefore, we expect to improve the tradeoff between prediction speed and accuracy[1] by optimizing a whole model including a search index in an end-to-end manner.

[1]The model size can also be optimized.

We mainly focused on retrieval for the vector space model in this thesis. On the other hand, in online advertising, advertisers want to show their ads to only a subset of users. For example, some ads are requested to be shown to women in their twenties or thirties. Typically, this condition is represented by Boolean expressions like:

$$\text{Gender} \in \{\text{female}\} \wedge \text{Age} \in \{\text{20s} \vee \text{30s}\}.$$

In information retrieval, methods have been proposed that efficiently evaluate Boolean expressions by using some data structures have been proposed [108, 44]. We will investigate optimization methods to these data structures by utilizing machine learning techniques.

In real-world recommendation setting, since the candidate items typically change as time progress, systems are required to balance exploration and exploitation [65]. We can not determine actual worth of a new coming item without presenting the item to users. On the other hand, users have been annoyed if unfavorable items are shown to them too much. Therefore, exploration and exploitation must be balanced. Development of an efficient exploration method for a lot of candidates by utilizing a search index is also a promising direction of our future work.

Our representation learning method obtains low-dimensional vectors from users' Web browsing sequences in an unsupervised manner, as described in Chapter 5. These learned vectors are used as common features among various user-related prediction tasks. Thus, this method does not utilizes users' properties or actions, which are targets of each task, for generating user representations. To remedy this problem, we plan to investigate a method that obtains user representations via learning other than unsupervised learning, such as semi-supervised, multi-label, and multi-task learning. We also consider that meta-learning or learning-to-learn approaches [42, 76, 98] can be helpful for this task.

Diversity [6] and fairness [48] in recommendation have received much attention. These topics are not necessarily required to simply improve recommendation quality or speed. However, we believe that these topics will play crucial role in future recommendation systems as systems become integrated into people's daily lives.

Appendix

In this chapter, some additional experimental results are shown.

Table 1 shows the experimental results of AnnexML and GPT in terms of nDCG@k. Experimental settings are the same as those of Figure 4.3.

Table 1: Experimental results for nDCG@k

		AnnexML	GPT
AmazonCat-13K	nDCG@1	0.9355	0.9084
	nDCG@3	0.8730	0.8496
	nDCG@5	0.8512	0.8314
Wiki10-31K	nDCG@1	0.8650	0.8476
	nDCG@3	0.7714	0.7599
	nDCG@5	0.6944	0.6834
Delicious-200K	nDCG@1	0.4666	0.4746
	nDCG@3	0.4219	0.4303
	nDCG@5	0.3981	0.4079
WikiLSHTC-325K	nDCG@1	0.6336	0.6336
	nDCG@3	0.5664	0.5623
	nDCG@5	0.5626	0.5568
Amazon-670K	nDCG@1	0.4208	0.4236
	nDCG@3	0.3881	0.3935
	nDCG@5	0.3680	0.3771

Figures 1 and 2 plot the prediction time and performances of AnnexML, AnnexML-BF, SLEEC, GPT, FastXML, and PfastreXML in terms of Precision@3, nDCG@3, nDCG@5. These experiments were performed in settings the same as those in Figure 4.1.

Table 2 shows the experimental results of AnnexML and GPT in terms of propensity scored Precision@k (PSP@k) and propensity scored nDCG@k (PSnDCG@k) [53]. Again, experimental settings are the same as those in Figure 4.3.

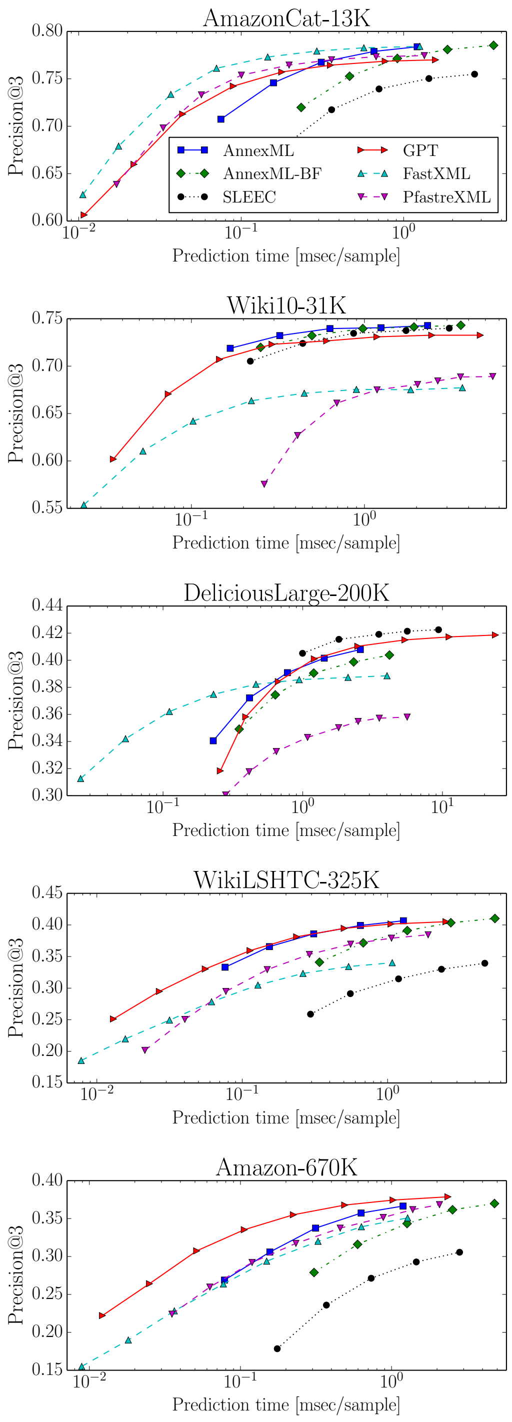


Figure 1: Precision@3 versus prediction time when the number of learners changes.

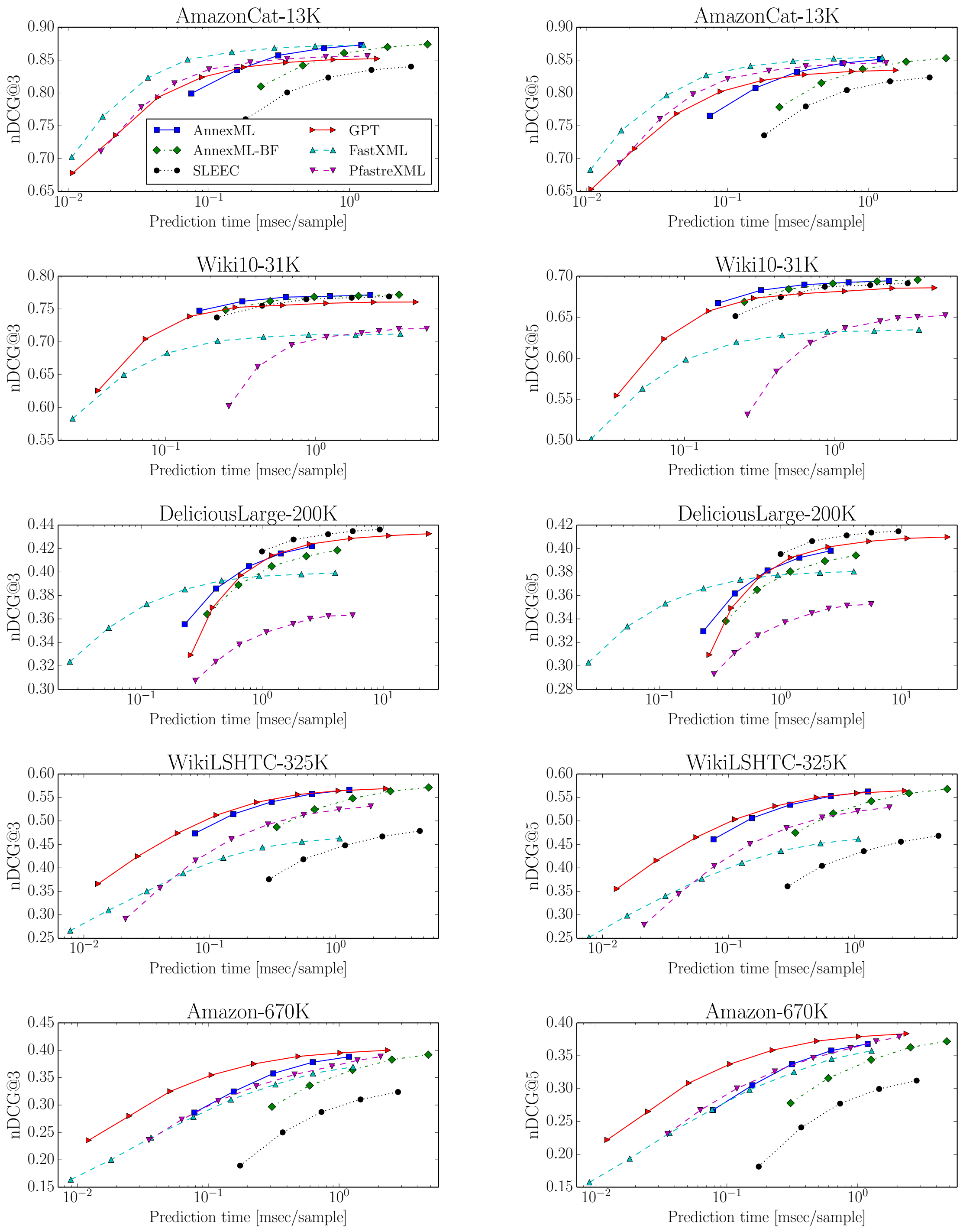


Figure 2: nDCG versus prediction time when the number of learners changes.

Table 2: Experimental results for propensity scored Precision@k (PSP@k) and propensity scored nDCG@k (PSnDCG@k) [53]

		AnnexML	GPT
AmazonCat-13K	PSP@1	0.4902	0.4720
	PSP@3	0.6111	0.5967
	PSP@5	0.6964	0.6933
Wiki10-31K	PSP@1	0.1190	0.1088
	PSP@3	0.1277	0.1250
	PSP@5	0.1328	0.1358
Delicious-200K	PSP@1	0.0716	0.0725
	PSP@3	0.0806	0.0819
	PSP@5	0.0873	0.0894
WikiLSHTC-325K	PSP@1	0.2537	0.2657
	PSP@3	0.3069	0.3057
	PSP@5	0.3440	0.3335
Amazon-670K	PSP@1	0.2147	0.2321
	PSP@3	0.2467	0.2667
	PSP@5	0.2755	0.3007

		AnnexML	GPT
AmazonCat-13K	PSnDCG@1	0.4902	0.4720
	PSnDCG@3	0.5882	0.5701
	PSnDCG@5	0.6547	0.6419
Wiki10-31K	PSnDCG@1	0.1190	0.1088
	PSnDCG@3	0.1211	0.1254
	PSnDCG@5	0.1310	0.1269
Delicious-200K	PSnDCG@1	0.0716	0.0725
	PSnDCG@3	0.0779	0.0790
	PSnDCG@5	0.0821	0.0837
WikiLSHTC-325K	PSnDCG@1	0.2537	0.2657
	PSnDCG@3	0.3133	0.3192
	PSnDCG@5	0.3443	0.3459
Amazon-670K	PSnDCG@1	0.2147	0.2321
	PSnDCG@3	0.2327	0.2521
	PSnDCG@5	0.2466	0.2690

Bibliography

[1] Netflix prize data. `https://www.kaggle.com/netflix-inc/netflix-prize-data`. Last Accessed: May 23, 2018.

[2] Deepak Agarwal, Rahul Agrawal, Rajiv Khanna, and Nagaraj Kota. Estimating rates of rare events with multiple hierarchies through scalable log-linear models. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, pages 213–222. ACM, 2010.

[3] Deepak Agarwal and Maxim Gurevich. Fast top-k retrieval for model based recommendation. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, WSDM '12, pages 483–492. ACM, 2012.

[4] Rahul Agrawal, Archit Gupta, Yashoteja Prabhu, and Manik Varma. Multi-label learning with millions of labels: Recommending advertiser bid phrases for web pages. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 13–24. ACM, 2013.

[5] Mohamed Aly, Andrew Hatch, Vanja Josifovski, and Vijay K. Narayanan. Web-scale user modeling for targeting. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12 Companion, pages 3–12. ACM, 2012.

[6] Arda Antikacioglu and R. Ravi. Post processing recommender systems for diversity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 707–716. ACM, 2017.

[7] Kevin Aydin, MohammadHossein Bateni, and Vahab Mirrokni. Distributed balanced partitioning via linear embedding. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, WSDM '16, pages 387–396. ACM, 2016.

[8] Rohit Babbar and Bernhard Schölkopf. Dismec: Distributed sparse machines for extreme multi-label classification. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, WSDM '17, pages 721–729. ACM, 2017.

[9] Yoram Bachrach, Yehuda Finkelstein, Ran Gilad-Bachrach, Liran Katzir, Noam Koenigstein, Nir Nice, and Ulrich Paquet. Speeding up the xbox recommender

system using a euclidean transformation for inner-product spaces. In *Proceedings of the 8th ACM Conference on Recommender Systems*, RecSys '14, pages 257–264. ACM, 2014.

[10] Erik Bernhardsson, Martin Aumüller, and Alexander Faithfull. Benchmarking nearest neighbors. `https://github.com/erikbern/ann-benchmarks`, 2015. Last Accessed: June 2, 2018.

[11] Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed H. Chi. Latent cross: Making use of context in recurrent recommender systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, WSDM '18, pages 46–54. ACM, 2018.

[12] Kush Bhatia, Himanshu Jain, Purushottam Kar, Manik Varma, and Prateek Jain. Sparse local embeddings for extreme multi-label classification. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 730–738. Curran Associates, Inc., 2015.

[13] Kush Bhatia, Himanshu Jain, Yashoteja Prabhu, and Manik Varma. The extreme classification repository. `http://manikvarma.org/downloads/XC/XMLRepository.html`, 2016. Last Accessed: May 22, 2018.

[14] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, Inc., 2009.

[15] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *CoRR*, abs/1607.04606, 2016.

[16] Léon Bottou and Yoshua Bengio. Convergence properties of the k-means algorithms. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 585–592. MIT Press, 1995.

[17] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.

[18] Andrei Broder, Marcus Fontoura, Vanja Josifovski, and Lance Riedel. A semantic approach to contextual advertising. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 559–566. ACM, 2007.

[19] Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. Efficient query evaluation using a two-level retrieval process. In *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, CIKM '03, pages 426–434. ACM, 2003.

[20] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 89–96. ACM, 2005.

[21] Chris J. C. Burges, Krysta M. Svore, Qiang Wu, and Jianfeng Gao. Ranking, boosting, and model adaptation. Technical report, October 2008.

[22] Christopher J. Burges, Robert Ragno, and Quoc V. Le. Learning to rank with nonsmooth cost functions. In B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 193–200. MIT Press, 2007.

[23] Christopher J.C. Burges. From ranknet to lambdarank to lambdamart: An overview. Technical report, June 2010.

[24] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: From pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 129–136. ACM, 2007.

[25] Deepayan Chakrabarti, Deepak Agarwal, and Vanja Josifovski. Contextual advertising by combining relevance with click feedback. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, pages 417–426. ACM, 2008.

[26] Haibin Cheng and Erick Cantú-Paz. Personalized click prediction in sponsored search. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, WSDM '10, pages 351–360. ACM, 2010.

[27] Haibin Cheng, Roelof van Zwol, Javad Azimi, Eren Manavoglu, Ruofei Zhang, Yang Zhou, and Vidhya Navalpakkam. Multimedia features for click prediction of new ads in display advertising. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 777–785. ACM, 2012.

[28] Wei-Sheng Chin, Yong Zhuang, Yu-Chin Juan, and Chih-Jen Lin. A fast parallel stochastic gradient method for matrix factorization in shared memory systems. *ACM Trans. Intell. Syst. Technol.*, 6(1):2:1–2:24, March 2015.

[29] Wei-Sheng Chin, Yong Zhuang, Yu-Chin Juan, and Chih-Jen Lin. A learning-rate schedule for stochastic gradient methods to matrix factorization. In Tru Cao, Ee-Peng Lim, Zhi-Hua Zhou, Tu-Bao Ho, David Cheung, and Hiroshi Motoda, editors, *Advances in Knowledge Discovery and Data Mining*, pages 442–455. Springer International Publishing, 2015.

[30] Aaron Clauset, Cosma Rohilla Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Review*, 51(4):661–703, 2009.

[31] Josh Constine. How instagram ' s algorithm works. `https://techcrunch.com/2018/06/01/how-instagram-feed-works/`. Last Accessed: June 7, 2018.

[32] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pages 191–198. ACM, 2016.

[33] Brian Dalessandro, Daizhuo Chen, Troy Raeder, Claudia Perlich, Melinda Han Williams, and Foster Provost. Scalable hands-free transfer learning for online advertising. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 1573–1582. ACM, 2014.

[34] Kushal S. Dave and Vasudeva Varma. Learning the click-through rate for rare/new ads from similar ads. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '10, pages 897–898. ACM, 2010.

[35] Shuai Ding and Torsten Suel. Faster top-k document retrieval using block-max indexes. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, pages 993–1002. ACM, 2011.

[36] Nemanja Djuric, Vladan Radosavljevic, Mihajlo Grbovic, and Narayan Bhamidipati. Hidden conditional random fields with deep user embeddings for ad targeting. In *2014 IEEE International Conference on Data Mining*, pages 779–784, Dec 2014.

[37] Wei Dong, Charikar Moses, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 577–586. ACM, 2011.

[38] Pinar Donmez, Krysta M. Svore, and Christopher J.C. Burges. On the local optimality of lambdarank. In *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, pages 460–467. ACM, 2009.

[39] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, July 2011.

[40] Ali Mamdouh Elkahky, Yang Song, and Xiaodong He. A multi-view deep learning approach for cross domain user modeling in recommendation systems. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 278–288. International World Wide Web Conferences Steering Committee, 2015.

[41] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, June 2008.

[42] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR, 06–11 Aug 2017.

[43] Marcus Fontoura, Vanja Josifovski, Jinhui Liu, Srihari Venkatesan, Xiangfei Zhu, and Jason Zien. Evaluation strategies for top-k queries over memory-resident inverted indexes. *Proceedings of the VLDB Endowment*, 4(12):1213–1224, 2011.

[44] Marcus Fontoura, Suhas Sadanandan, Jayavel Shanmugasundaram, Sergei Vassilvitski, Erik Vee, Srihari Venkatesan, and Jason Zien. Efficiently evaluating complex boolean expressions. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 3–14. ACM, 2010.

[45] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 518–529. Morgan Kaufmann Publishers Inc., 1999.

[46] Carlos A. Gomez-Uribe and Neil Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems*, 6(4):13:1–13:19, December 2015.

[47] Thore Graepel, Joaquin Quiñonero Candela, Thomas Borchert, and Ralf Herbrich. Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft's bing search engine. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 13–20. Omnipress, 2010.

[48] Moritz Hardt. Fairness in machine learning. `https://fairmlclass.github.io/`. Last Accessed: June 20, 2018.

[49] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems*, 5(4):19:1–19:19, December 2015.

[50] Andrew Hatch, Abraham Bagherjeiran, and Adwait Ratnaparkhi. Clickable terms for contextual advertising. In *Proceedings of the Fourth International Workshop on Data Mining and Audience Intelligence for Advertising*, 2010.

[51] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22Nd ACM International Conference on Information & Knowledge Management*, CIKM '13, pages 2333–2338. ACM, 2013.

[52] Masajiro Iwasaki. Pruned bi-directed k-nearest neighbor graph for proximity search. In *International Conference on Similarity Search and Applications*, pages 20–33. Springer, 2016.

[53] Himanshu Jain, Yashoteja Prabhu, and Manik Varma. Extreme multi-label loss functions for recommendation, tagging, ranking &#38; other missing label appli-

cations. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 935–944. ACM, 2016.

[54] Prateek Jain, Raghu Meka, and Inderjit S. Dhillon. Guaranteed rank minimization via singular value projection. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 937–945. Curran Associates, Inc., 2010.

[55] Kalina Jasinska, Krzysztof Dembczynski, Robert Busa-Fekete, Karlson Pfannschmidt, Timo Klerx, and Eyke Hullermeier. Extreme f-measure maximization using sparse probability estimates. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1435–1444. PMLR, 20–22 Jun 2016.

[56] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 133–142. ACM, 2002.

[57] Amruta Joshi, Abraham Bagherjeiran, and Adwait Ratnaparkhi. User demographic and behavioral targeting for content match advertising. In *Proceedings of the Fifth International Workshop on Data Mining and Audience Intelligence for Advertising*, 2011.

[58] Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. Field-aware factorization machines for ctr prediction. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pages 43–50. ACM, 2016.

[59] Maryam Karimzadehgan, Wei Li, Ruofei Zhang, and Jianchang Mao. A stochastic learning-to-rank algorithm and its application to contextual advertising. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 377–386. ACM, 2011.

[60] Nicolas Koumchatzky and Anton Andryeyev. Using deep learning at scale in twitter's timelines. `https://blog.twitter.com/engineering/en_us/topics/insights/2017/using-deep-learning-at-scale-in-twitters-timelines.html`, 2017. Last Accessed: May 22, 2018.

[61] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 489–504. ACM, 2018.

[62] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages II–1188–II–1196. JMLR.org, 2014.

[63] Yann LeCun, Leon Bottou, Genevieve B. Orr, and Klaus Robert Müller. *Efficient BackProp*, pages 9–50. Springer Berlin Heidelberg, 1998.

[64] Kuang-chih Lee, Burkay Orten, Ali Dasdan, and Wentong Li. Estimating conversion rate in display advertising from past performance data. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 768–776. ACM, 2012.

[65] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 661–670. ACM, 2010.

[66] Jingzhou Liu, 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*, SIGIR '17, pages 115–124. ACM, 2017.

[67] Ting Liu, Andrew W. Moore, Ke Yang, and Alexander G. Gray. An investigation of practical approximate nearest neighbor algorithms. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 825–832. MIT Press, 2005.

[68] Yandong Liu, Sandeep Pandey, Deepak Agarwal, and Vanja Josifovski. Finding the right consumer: Optimizing for conversion in display advertising campaigns. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, WSDM '12, pages 473–482. ACM, 2012.

[69] Tianyi Luo, Dong Wang, Rong Liu, and Yiqiao Pan. Stochastic top-k listnet. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 676–684. Association for Computational Linguistics, September 2015.

[70] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval.* Cambridge University Press, 2008.

[71] Julian McAuley and Jure Leskovec. Hidden factors and hidden topics: Understanding rating dimensions with review text. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 165–172. ACM, 2013.

[72] H. Brendan McMahan. Follow-the-regularized-leader and mirror descent: Equivalence theorems and l1 regularization. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

[73] H. Brendan McMahan, Gary Holt, D. Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg, Arnar Mar Hrafnkelsson, Tom Boulos, and Jeremy Kubica. Ad click prediction: A view from the trenches. In *Proceedings of the 19th*

ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '13, pages 1222–1230. ACM, 2013.

[74] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

[75] Paul Mineiro and Nikos Karampatziakis. Fast label embeddings via randomized linear algebra. In Annalisa Appice, Pedro Pereira Rodrigues, Vítor Santos Costa, Carlos Soares, João Gama, and Alípio Jorge, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 37–51. Springer International Publishing, 2015.

[76] Tsendsuren Munkhdalai and Hong Yu. Meta networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2554–2563. PMLR, 06–11 Aug 2017.

[77] Vanessa Murdock, Massimiliano Ciaramita, and Vassilis Plachouras. A noisy-channel approach to contextual advertising. In *Proceedings of the 1st International Workshop on Data Mining and Audience Intelligence for Advertising*, ADKDD '07, pages 21–27. ACM, 2007.

[78] Shumpei Okura, Yukihiro Tagami, Shingo Ono, and Akira Tajima. Embedding-based news recommendation for millions of users. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 1933–1942. ACM, 2017.

[79] Stephen M. Omohundro. Five balltree construction algorithms. Technical report, International Computer Science Institute, 1989.

[80] Junwei Pan, Jian Xu, Alfonso Lobos Ruiz, Wenliang Zhao, Shengjun Pan, Yu Sun, and Quan Lu. Field-weighted factorization machines for click-through rate prediction in display advertising. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, pages 1349–1357. International World Wide Web Conferences Steering Committee, 2018.

[81] Ioannis Partalas, Aris Kosmopoulos, Nicolas Baskiotis, Thierry Artières, George Paliouras, Éric Gaussier, Ion Androutsopoulos, Massih-Reza Amini, and Patrick Gallinari. LSHTC: A benchmark for large-scale text classification. *CoRR*, abs/1503.08581, 2015.

[82] C. Perlich, B. Dalessandro, T. Raeder, O. Stitelman, and F. Provost. Machine learning for targeted display advertising: Transfer learning in action. *Machine Learning*, 95(1):103–127, April 2014.

[83] Yashoteja Prabhu and Manik Varma. Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 263–272. ACM, 2014.

[84] Ariadna Quattoni, Sybor Wang, Louis-Philippe Morency, Morency Collins, and Trevor Darrell. Hidden conditional random fields. *IEEE transactions on pattern analysis and machine intelligence*, 29(10), 2007.

[85] Parikshit Ram and Alexander G. Gray. Maximum inner-product search using cone trees. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 931–939. ACM, 2012.

[86] Adwait Ratnaparkhi. A hidden class page-ad probability model for contextual advertising. In *Workshop on Targeting and Ranking for Online Advertising at the 17th International World Wide Web Conference*, 2008.

[87] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 693–701. Curran Associates, Inc., 2011.

[88] Rómer Rosales, Haibin Cheng, and Eren Manavoglu. Post-click conversion modeling and analysis for non-guaranteed delivery display advertising. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, WSDM '12, pages 293–302. ACM, 2012.

[89] Ying Shan, Jian Jiao, Jie Zhu, and J. C. Mao. Recurrent binary embedding for gpu-enabled exhaustive retrieval from billion-scale semantic vectors. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '18. ACM, 2018. To appear.

[90] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.

[91] Anshumali Shrivastava and Ping Li. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2321–2329. Curran Associates, Inc., 2014.

[92] Anshumali Shrivastava and Ping Li. Improved asymmetric locality sensitive hashing (alsh) for maximum inner product search (mips). In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*, UAI'15, pages 812–821. AUAI Press, 2015.

[93] Si Si, Kai-Yang Chiang, Cho-Jui Hsieh, Nikhil Rao, and Inderjit S. Dhillon. Goal-directed inductive matrix completion. In *Proceedings of the 22Nd ACM SIGKDD*

International Conference on Knowledge Discovery and Data Mining, KDD '16, pages 1165–1174. ACM, 2016.

[94] Si Si, Huan Zhang, S. Sathiya Keerthi, Dhruv Mahajan, Inderjit S. Dhillon, and Cho-Jui Hsieh. Gradient boosted decision trees for high dimensional sparse output. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3182–3190. PMLR, 06–11 Aug 2017.

[95] Pablo Sprechmann, Roee Litman, Tal Ben Yakar, Alexander M Bronstein, and Guillermo Sapiro. Supervised sparse analysis and synthesis operators. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 908–916. Curran Associates, Inc., 2013.

[96] Trevor Strohman and W. Bruce Croft. Efficient document retrieval in main memory. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 175–182. ACM, 2007.

[97] Kohei Sugawara, Hayato Kobayashi, and Masajiro Iwasaki. On approximately searching for similar word embeddings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2265–2275. Association for Computational Linguistics, August 2016.

[98] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip H.S. Torr, and Timothy M. Hospedales. Learning to compare: Relation network for few-shot learning. In *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[99] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014.

[100] Yukihiro Tagami, Hayato Kobayashi, Shingo Ono, and Akira Tajima. Modeling user activities on the web using paragraph vector. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15 Companion, pages 125–126. ACM, 2015.

[101] Yukihiro Tagami, Shingo Ono, Koji Yamamoto, Koji Tsukamoto, and Akira Tajima. Ctr prediction for contextual advertising: Learning-to-rank approach. In *Proceedings of the Seventh International Workshop on Data Mining for Online Advertising*, ADKDD '13, pages 4:1–4:8. ACM, 2013.

[102] Jian Tang, Jingzhou Liu, Ming Zhang, and Qiaozhu Mei. Visualizing large-scale and high-dimensional data. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, pages 287–297. International World Wide Web Conferences Steering Committee, 2016.

[103] Ilya Trofimov, Anna Kornetova, and Valery Topinskiy. Using boosted trees for click-through rate prediction for sponsored search. In *Proceedings of the Sixth International Workshop on Data Mining for Online Advertising and Internet Economy*, ADKDD '12, pages 2:1–2:6. ACM, 2012.

[104] Chi Wang, Rajat Raina, David Fong, Ding Zhou, Jiawei Han, and Greg Badros. Learning relevance from heterogeneous social network and its application in online targeting. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, pages 655–664. ACM, 2011.

[105] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 1113–1120. ACM, 2009.

[106] Jason Weston, Ameesh Makadia, and Hector Yee. Label partitioning for sublinear ranking. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 181–189. PMLR, 17–19 Jun 2013.

[107] Robert Wetzker, Carsten Zimmermann, and Christian Bauckhage. Analyzing social bookmarking systems: A del. icio. us cookbook. In *Proceedings of the ECAI 2008 Mining Social Data Workshop*, pages 26–30, 2008.

[108] Steven Euijong Whang, Hector Garcia-Molina, Chad Brower, Jayavel Shanmugasundaram, Sergei Vassilvitskii, Erik Vee, and Ramana Yerneni. Indexing boolean expressions. *Proc. VLDB Endow.*, 2(1):37–48, August 2009.

[109] Ryen W. White, Peter Bailey, and Liwei Chen. Predicting user interests from contextual information. In *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, pages 363–370. ACM, 2009.

[110] Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. Listwise approach to learning to rank: Theory and algorithm. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 1192–1199. ACM, 2008.

[111] Chang Xu, Dacheng Tao, and Chao Xu. Robust extreme multi-label learning. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1275–1284. ACM, 2016.

[112] Qiang Yang, Haining Henry Zhang, and Tianyi Li. Mining web logs for prediction models in www caching and prefetching. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 473–478. ACM, 2001.

[113] Ian E.H. Yen, Xiangru Huang, Wei Dai, Pradeep Ravikumar, Inderjit Dhillon, and Eric Xing. Ppdsparse: A parallel primal-dual sparse method for extreme classification. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 545–553. ACM, 2017.

[114] Ian En-Hsu Yen, Xiangru Huang, Pradeep Ravikumar, Kai Zhong, and Inderjit Dhillon. Pd-sparse : A primal and dual sparse approach to extreme multiclass and multilabel classification. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 3069–3077. PMLR, 20–22 Jun 2016.

[115] Peter N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '93, pages 311–321. Society for Industrial and Applied Mathematics, 1993.

[116] Wen-tau Yih and Ning Jiang. Similarity models for ad relevance measures. In *MLOAD - NIPS 2010 Workshop on online advertising*, 2010.

[117] Wen-tau Yih, Kristina Toutanova, John C. Platt, and Christopher Meek. Learning discriminative projections for text similarity measures. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning*, CoNLL '11, pages 247–256. Association for Computational Linguistics, 2011.

[118] Hsiang-Fu Yu, Prateek Jain, Purushottam Kar, and Inderjit Dhillon. Large-scale multi-label learning with missing labels. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 593–601. PMLR, 22–24 Jun 2014.

[119] Shuai Yuan, Ahmad Zainal Abidin, Marc Sloan, and Jun Wang. Internet advertising: An interplay among advertisers, online publishers, ad exchanges and web users. *CoRR*, abs/1206.1754, 2012.

[120] Arkaitz Zubiaga. Enhancing navigation on wikipedia with social tags. In *Proceedings of Wikimania 2009, 5th International Conference of the Wikimedia Community*, 2009.