Ranking Distillation: Learning Compact Ranking Models With High Performance for Recommender System

KDD 2018, August 19–23, 2018, London, United Kingdom

报告人：王昕毅

指导老师：高旻 副教授

Knowledge Distillation Concepts

Background

随着深度学习理论的普及和PC计算能力的提高, Neural Network也日益趋于复杂化–越来越深和越来越大, 比如VGG系列, 深度均在10层以上, 残差网络的深度更是从数十到一百以上. 复杂的网络模型带来的是超强的规则学习能力, 虽然以现在动辄几十或上百多机GPU并行的计算能力而言, 这样的巨大模型并不是拦路虎, 但是考虑到当今**微小智能设备**的普及, smart phone, 可穿戴设备, 智能手表等, 要在这些嵌入式设备中放入动辄几十上百M的模型是不现实的. 就算能放下, 其高昂的computation cost也是难以承受的. 那么有没有办法既保证模型能力又能大幅削减cost呢? 答案是肯定的. 在深度学习领域有一个专门的小分支研究model compression, 即模型压缩. 模型压缩方法可以大致分为4类: **参数共享**(Parameter sharing), **网络裁剪**(Network pruning), **矩阵分解**(Matrix decomposition), 以及**知识蒸馏**(Knowledge Distillation)。

Advantage

结论:一个具有有限个神经元的单一隐藏层的BP网络就可以拟合任意目标函数

在**Fitnets: Hints for thin deep nets**中, 通过pre-train和teach-student的训练方法, 作者仅仅使用了2.5M的参数就达到了接近state-of-the-art的效果, 而使用的网络参数数目比传统方法采用的模型小得多.

Knowledge Distillation就提供了一条很好的思路—-使用大模型(Teacher)来teaching小模型(Student). 有人可能会问了, 就算大模型精度再高, predict acc也不可能达到100%的准确度(至少是非常难的), 为啥不直接使用更为准确的training dataset? 这里有几个原因:

1.Teacher的输出往往带有一些额外的信息, 即Dark Knowledge. 手写体识别的classifier为例, 该classifier的作用是识别0-9的手写体数字, 其最后输出的是一个样本属于10个数字的distribution. 比如样本x的输出为[1e-10, 1e-10, 0.98, 0.01, 1e-10, 0.009, 1e-10, 1e-10, 1e-10, 1e-10] (依次对应数字0-9), 根据输出distribution, 最大概率为数字2, 故模型对x的预测标签为2. 需要注意的是, 3和5对应的概率虽然小但远大于除2以外的数字概率, 这说明模型认为x虽然代表数字2, 但它也有点像数字2和5. 这种附加的信息其实反映出classifier在学习到了某种规律(函数), 通过该规律来计算x与不同数字的相似度. 相比original training data中生硬的one-hot标签, 这种具有额外信息的soft label携带更多的数据结构信息。

2.相对于one-hot label, 由teacher提供的soft label更容易学习, 相比于输出概率集中在一个class上的one-hot, teacher模型将概率分散在了多个class上, 这样的好处是, 使模型的预测更加的贴近实际的情况(比如在第1点中提到的)。

3.通常在人工标注的训练数据中都存在错误标记, 而训练良好的Teacher模型的predict结果可以很好的”纠正”这些错误。

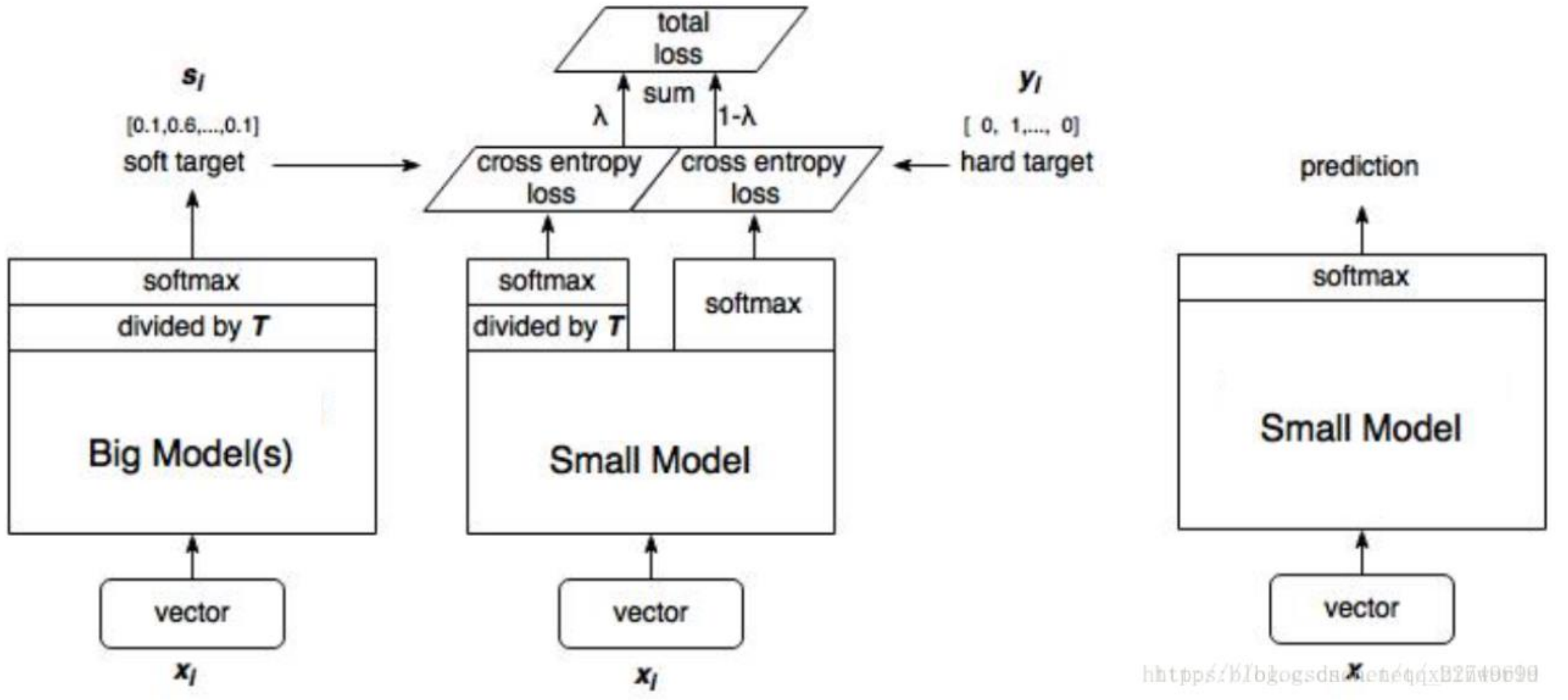
Model

Knowledge distillation的重点就是提出用soft label来辅助hard label一起训练，而soft label即是来自于teacher model的预测输出。Soft label的好处已在上文介绍。但是如果soft label是像这样的信息[0.98 0.01 0.01]，就意义不大了，所以需要在softmax中增加温度参数T,将label软化，使大模型学到的label能够提供更多的信息。

$$q_i = \frac{exp(z_i/T)}{\Sigma_j exp(z_j/T)}$$

最终的损失函数即为对soft label和hard label损失的加权和。

$$L = \alpha L^{(soft)} + (1-\alpha) L^{(hard)}$$



1、训练大模型：先用hard label，也就是正常的label训练大模型。

2、计算soft label：利用训练好的大模型来计算soft label。也就是大模型“软化后”再经过softmax的output。

3、训练小模型，在小模型的基础上再加一个额外的soft target的loss function，通过lambda来调节两个loss functions的比重。

4、预测时，将训练好的小模型按常规方式（右图）使用。

Ranking Distillation Concepts

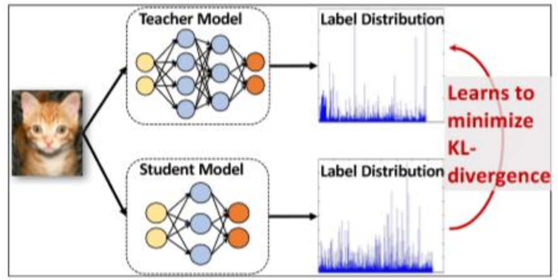
Abstract

1)We propose a **KD technique for learning to rank problems**, called **ranking distillation**(RD).

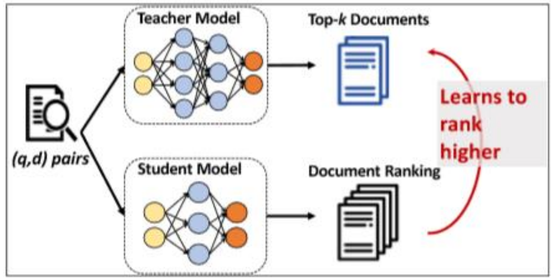
2)We address the **challenges** of RD for ranking problems.

3)Experiments on public data sets show: the student model learnt with RD **has a model size less than half of the teacher model while achieving a ranking performance similar to the teacher model** and much better than the student model learnt without RD.

Comparison between KD and RD



(a) Knowledge Distillation for classification



(b) Ranking Distillation for ranking problems

Figure 1: (a) Knowledge Distillation: given an input, student model learns to minimize the KL Divergence of its label distribution and teacher model's. (b) Ranking Distillation: given an query, student model learns to give higher rank for it's teacher model's top-*K* ranking of documents.

A small student model is trained to learn to rank from two sources of information, i.e., the training set and the **top-K documents for each query generated by a large well-trained teacher ranking model.**

Comparison between KD and RD

1.It is not straight forward to apply KD to ranking models and ranking problems (e.g., recommendation) . First, the existing **KD is designed for classification problems, not for ranking problems**. In ranking problems, the focus is on predicting the relative order of documents or items, instead of predicting a label or class as in classification.

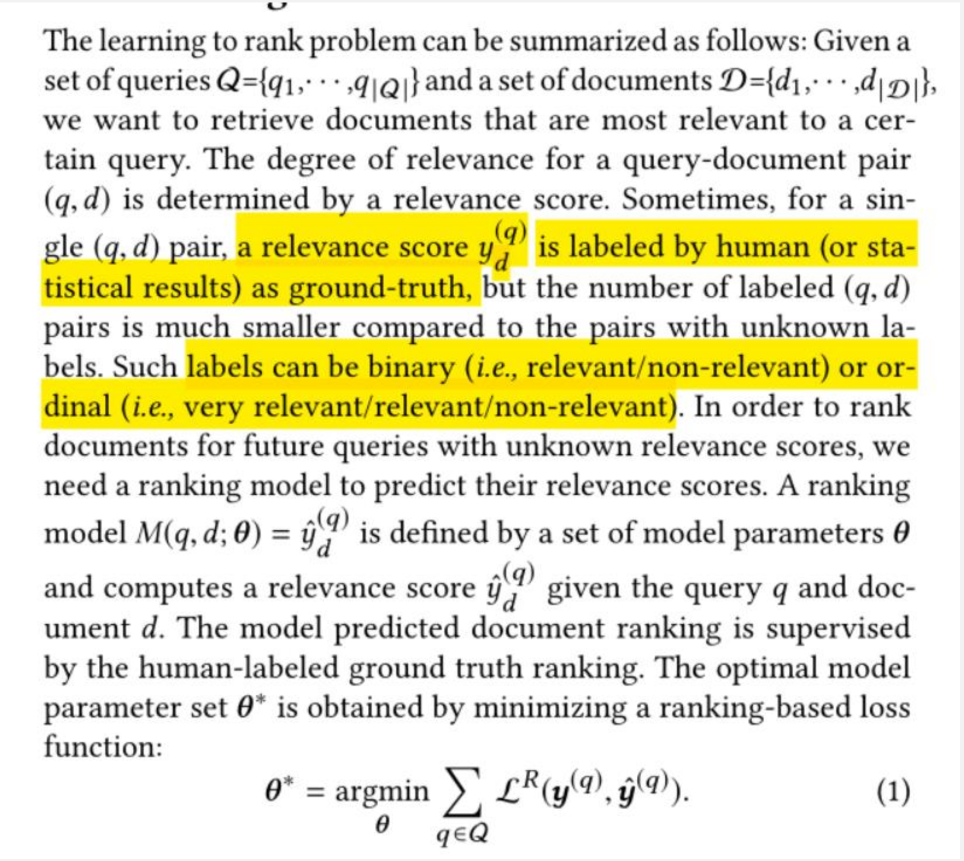
2.**KD requires computing the label distribution of documents for each query using both teacher and student models**, which is feasible for image classification where there are a small number of labels, for example, no more than 1000 for the ImageNet data set; **however, for ranking and recommendation problems, the total number of documents or items could be several orders of magnitudes larger**, say millions, and computing the distribution for all documents or items for each training instance makes little sense, **especially only the highly ranked documents or items near the top of the ranking will matter**.

3.With the large model size, the teacher model captures more ranking patterns from the training set and provides top-K ranked unlabeled documents as an extra training data for the student model. This makes RD differ from KD, as **teacher model in KD only generate additional labels on existing data, while RD generate additional training data and labels from unlabeled data set.**

Ranking Distillation Model

Ranking problem

Without loss of generality, we use the IR terms “query“ q and “document “ d in our discussion, but these terms can be replaced with “user profile" and “item" when applied to recommender systems.



The learning to rank problem can be summarized as follows: Given a set of queries $\mathcal{Q}=\{q_1,\cdots,q_{|\mathcal{Q}|}\}$ and a set of documents $\mathcal{D}=\{d_1,\cdots,d_{|\mathcal{D}|}\}$, we want to retrieve documents that are most relevant to a certain query. The degree of relevance for a query-document pair (q, d) is determined by a relevance score. Sometimes, for a single (q, d) pair, a relevance score $y_d^{(q)}$ is labeled by human (or statistical results) as ground-truth, but the number of labeled (q, d) pairs is much smaller compared to the pairs with unknown labels. Such labels can be binary (*i.e.*, relevant/non-relevant) or ordinal (*i.e.*, very relevant/relevant/non-relevant). In order to rank documents for future queries with unknown relevance scores, we need a ranking model to predict their relevance scores. A ranking model $M(q, d; \boldsymbol{\theta}) = \hat{y}_d^{(q)}$ is defined by a set of model parameters $\boldsymbol{\theta}$ and computes a relevance score $\hat{y}_d^{(q)}$ given the query q and document d. The model predicted document ranking is supervised by the human-labeled ground truth ranking. The optimal model parameter set $\boldsymbol{\theta}^*$ is obtained by minimizing a ranking-based loss function:

$$\boldsymbol{\theta}^* = \operatorname*{argmin}_{\boldsymbol{\theta}} \sum_{q\in\mathcal{Q}} \mathcal{L}^R(\boldsymbol{y}^{(q)}, \hat{\boldsymbol{y}}^{(q)}). \quad (1)$$

For simplicity, **we focus on a single query q** and omit the super-scripts related to queries (i.e., y_d^q will becomes y_d). Which means considering ranking result for a single user at a time.

The ranking-based loss could be categorized as point-wise, pair-wise, and list-wise. Since the first two are more widely adopted, we don't discuss list-wise loss in this work. The point-wise loss is widely used when relevance labels are binary [14, 33]. One typical point-wise loss is taking the negative logarithmic of the likelihood function:

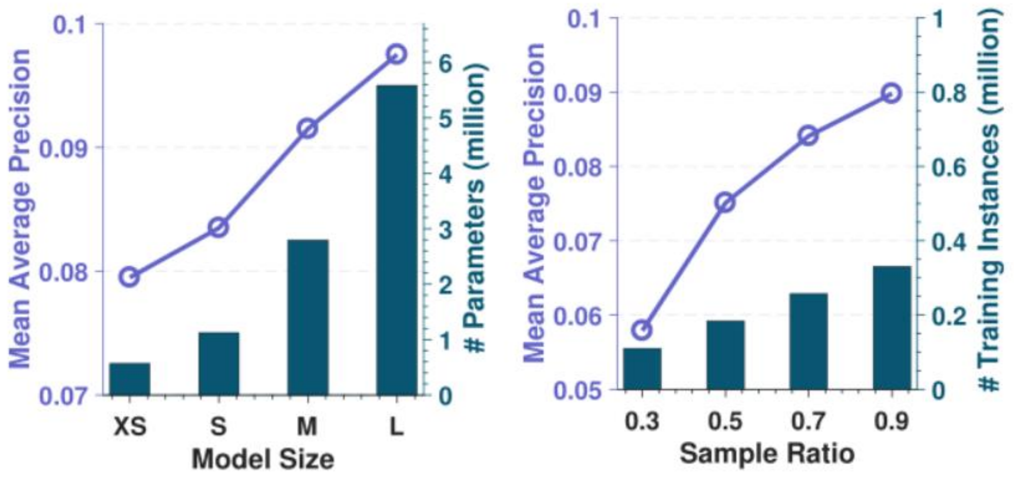
$$\begin{aligned}\mathcal{L}^R(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -(&\sum_{d \in \boldsymbol{y}_{d+}} \log(P(rel = 1|\hat{y}_d)) \\ &+ \sum_{d \in \boldsymbol{y}_{d-}} \log(1 - P(rel = 1|\hat{y}_d))),\end{aligned} \tag{2}$$

where $\boldsymbol{y}_{d+}$ and $\boldsymbol{y}_{d-}$ are the sets of relevant and non-relevant documents, respectively. We could use the *logistic* function $\sigma(x) = 1/(1+e^{-x})$ and $P(rel = 1|\hat{y}_d) = \sigma(\hat{y}_d)$ to transform a real-valued relevance score to the probability of a document being relevant ($rel = 1$). For ordinal relevance labels, pair-wise loss better models the partial order information:

$$\mathcal{L}^R(\boldsymbol{y}, \hat{\boldsymbol{y}}) = - \sum_{d_i, d_j \in C} \log(P(d_i \succ d_j|\hat{y}_i, \hat{y}_j)), \tag{3}$$

where C is the set of document pairs $\{(d_i, d_j) : y_i \succ y_j\}$ and the probability $P(d_i \succ d_j)$ can be modeled using the *logistic* function $P(d_i \succ d_j|y_i, y_j) = \sigma(y_i - y_j)$.

effectiveness and efficiency for ranking problems



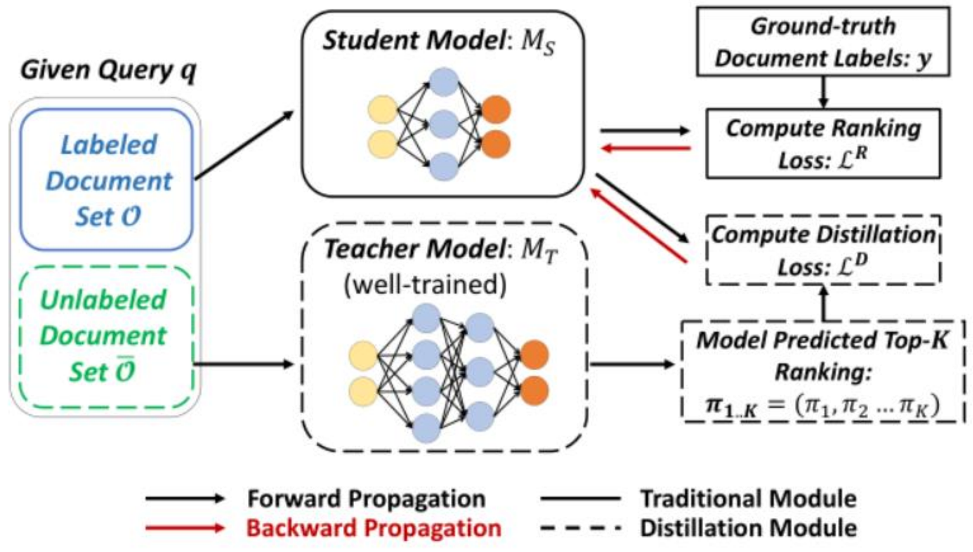
(a) MAP *vs.* model size (b) MAP *vs.* training instances

Figure 2: Two ways of boosting mean average precision (MAP) on Gowalla data for recommendation. (a) shows that a larger model size in number of parameters, indicated by the bars, leads to a higher MAP. (b) shows that a larger sample size of training instances leads to a higher MAP.

(1) By having a large model size, as long as it doesn't overfit the data, the model could better capture complex query-document interaction patterns and has more predictive capability.
(2) When more training instances are sampled from the Underlying data distribution, a ranking model could achieve a better performance.

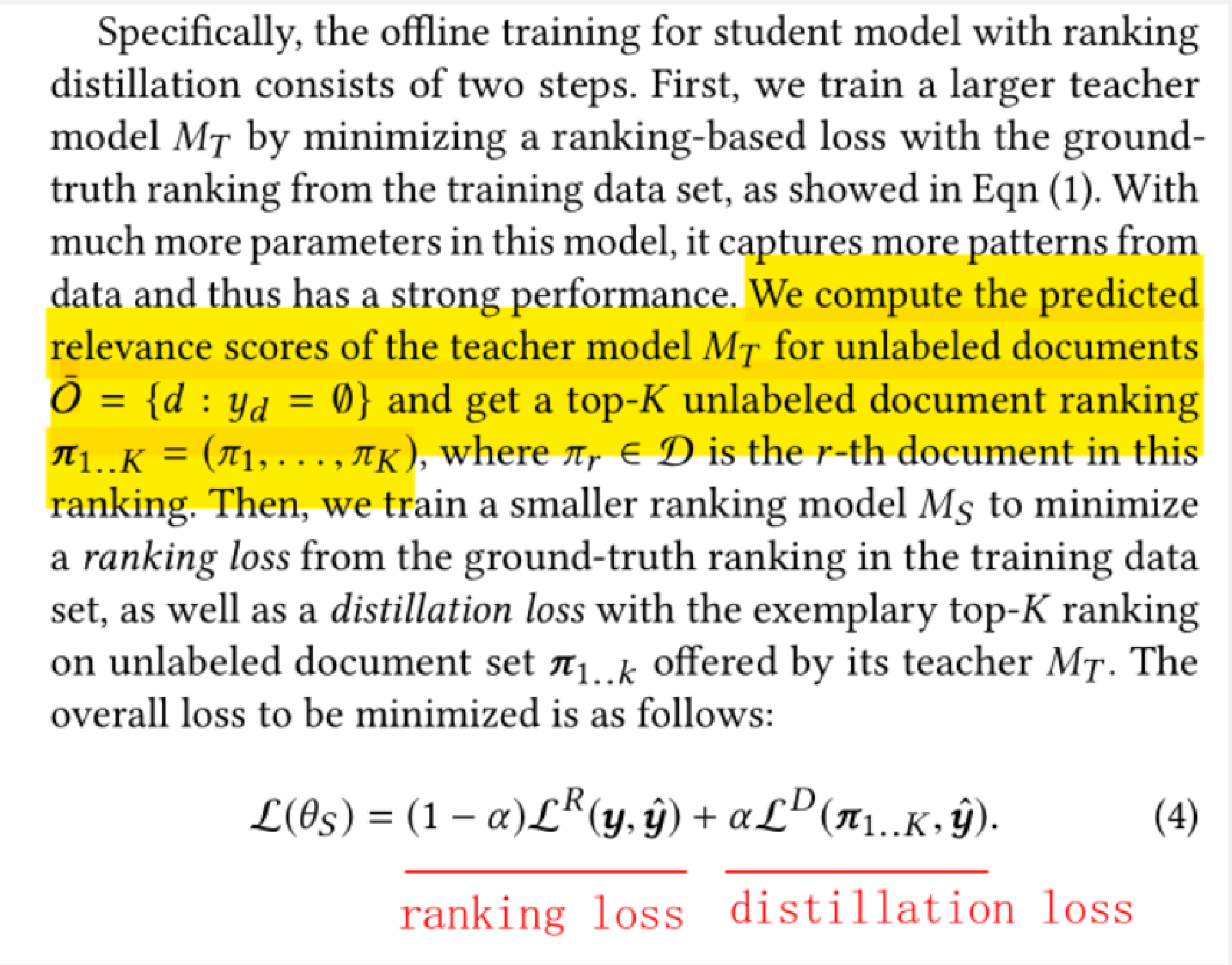
1)To address the **efficiency** of online inference, we use a **smaller ranking model** so that we can rank documents for a given query more efficiently.
2)To address the **effectiveness** issue without requiring more training data, we introduce **extra information generated from a well-trained teacher model** and make the student model as effective as the teacher.

Training process



Given Query q
Labeled Document Set $\mathcal{O}$
Unlabeled Document Set $\bar{\mathcal{O}}$
Student Model: M_S
Teacher Model: M_T (well-trained)
Ground-truth Document Labels: y
Compute Ranking Loss: $\mathcal{L}^R$
Compute Distillation Loss: $\mathcal{L}^D$
Model Predicted Top-K Ranking: $\pi_{1..K} = (\pi_1, \pi_2 \ldots \pi_K)$
Forward Propagation
Backward Propagation
Traditional Module
Distillation Module

Figure 3: The learning paradigm with ranking distillation. We first train a teacher model and let it predict a top-K ranked list of unlabeled (unobserved) documents for a given query q. The student model is then supervised by both ground-truth ranking from the training data set and teacher model's top-K ranking on unlabeled documents.



Specifically, the offline training for student model with ranking distillation consists of two steps. First, we train a larger teacher model M_T by minimizing a ranking-based loss with the ground-truth ranking from the training data set, as showed in Eqn (1). With much more parameters in this model, it captures more patterns from data and thus has a strong performance. We compute the predicted relevance scores of the teacher model M_T for unlabeled documents $\bar{O} = \{d : y_d = \emptyset\}$ and get a top-K unlabeled document ranking $\boldsymbol{\pi}_{1..K} = (\pi_1, \ldots, \pi_K)$, where $\pi_r \in \mathcal{D}$ is the r-th document in this ranking. Then, we train a smaller ranking model M_S to minimize a *ranking loss* from the ground-truth ranking in the training data set, as well as a *distillation loss* with the exemplary top-K ranking on unlabeled document set $\boldsymbol{\pi}_{1..k}$ offered by its teacher M_T. The overall loss to be minimized is as follows:

$$\mathcal{L}(\theta_S) = \underbrace{(1-\alpha)\mathcal{L}^R(\boldsymbol{y}, \hat{\boldsymbol{y}})}_{\text{ranking loss}} + \underbrace{\alpha\mathcal{L}^D(\boldsymbol{\pi}_{1..K}, \hat{\boldsymbol{y}})}_{\text{distillation loss}}. \quad (4)$$

It should be noted that ranking loss come from **ground truth** labels, while the distillation loss uses teacher model's top-K ranking(**including those unlabeled**

Documents) to guide the student model learning. α is the hyper-parameter used for balancing these two losses.

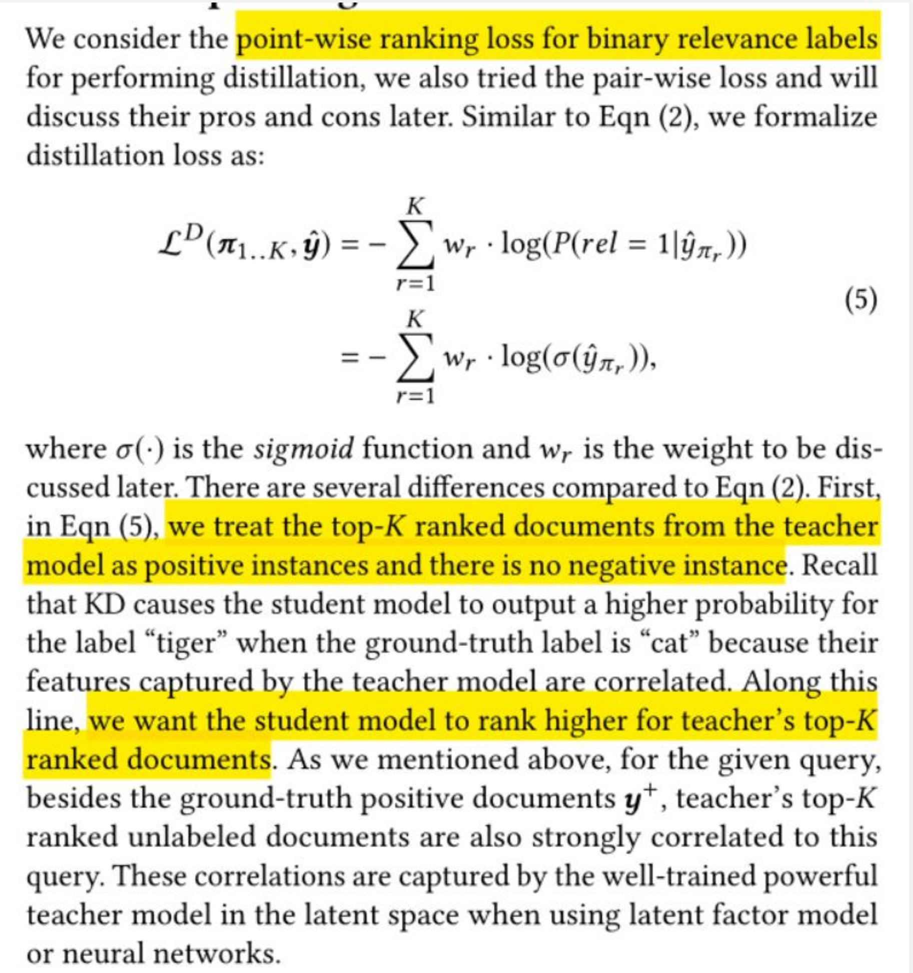
Case

For a given query, the **top documents** ranked by the well-trained teacher can be regarded to have **a strong correlation to this query**, although they are not labeled in the training set(Thus introduce more information other than the ground Truth).

For example, if a user watches many action movies, the teacher’s top-ranked documents may contain some other action movies as well as some adventure movies because they are correlated. In this sense, the proposed RD lets the teacher model teach its student to find the correlations and capture their patterns, thus, makes the student **more generalizable** and perform well on unseen data in the future. We use the top- K ranking from the teacher instead of the whole ranked list because the **noisy ranking at lower positions** tends to cause the student model to overfit its teacher and lose generalizability. Besides, **only top positions are considered important for ranking problems**. **K is a hyper-parameter that represents the trust level on the teacher during teaching**, i.e., how much we adopt from teacher.

Incorporating Distillation Loss

Incorporating Distillation Loss



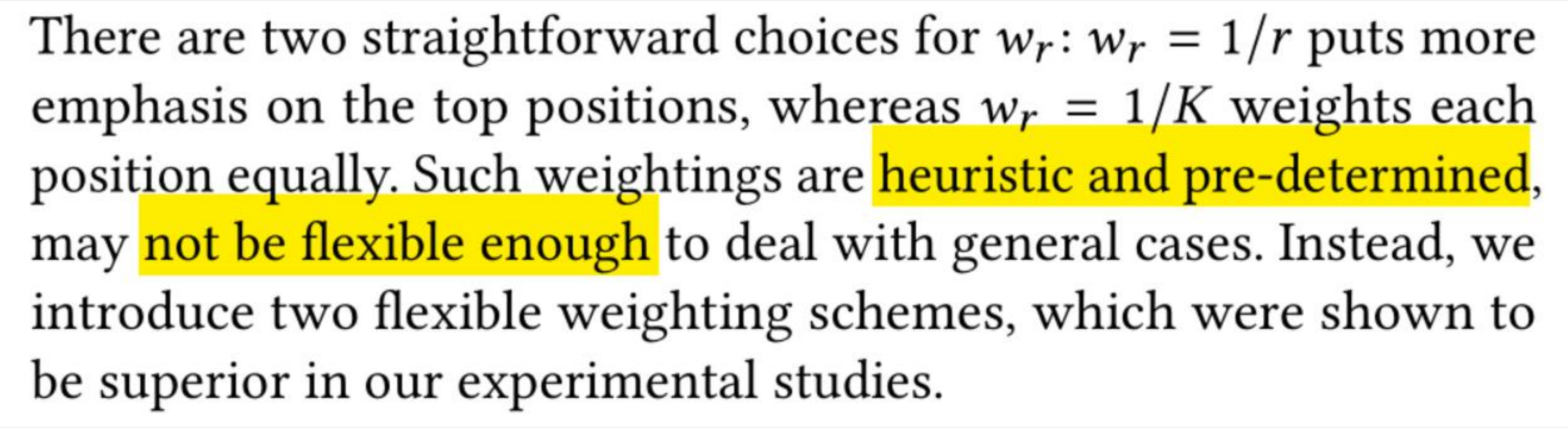
We consider the point-wise ranking loss for binary relevance labels for performing distillation, we also tried the pair-wise loss and will discuss their pros and cons later. Similar to Eqn (2), we formalize distillation loss as:

$$\begin{aligned}\mathcal{L}^D(\boldsymbol{\pi}_{1..K}, \hat{\boldsymbol{y}}) &= -\sum_{r=1}^{K} w_r \cdot \log(P(rel = 1|\hat{y}_{\pi_r})) \\ &= -\sum_{r=1}^{K} w_r \cdot \log(\sigma(\hat{y}_{\pi_r})),\end{aligned} \tag{5}$$

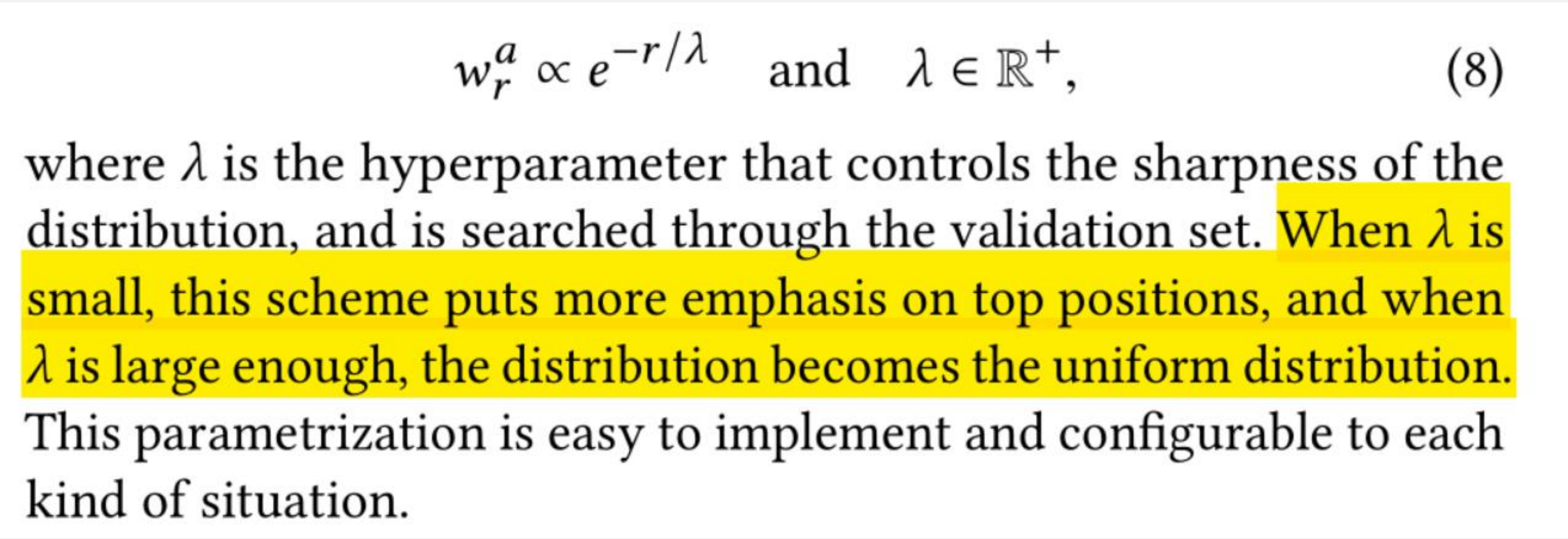
where $\sigma(\cdot)$ is the *sigmoid* function and w_r is the weight to be discussed later. There are several differences compared to Eqn (2). First, in Eqn (5), we treat the top-K ranked documents from the teacher model as positive instances and there is no negative instance. Recall that KD causes the student model to output a higher probability for the label "tiger" when the ground-truth label is "cat" because their features captured by the teacher model are correlated. Along this line, we want the student model to rank higher for teacher's top-K ranked documents. As we mentioned above, for the given query, besides the ground-truth positive documents $\boldsymbol{y}^+$, teacher's top-K ranked unlabeled documents are also strongly correlated to this query. These correlations are captured by the well-trained powerful teacher model in the latent space when using latent factor model or neural networks.

r is the position of documents from 1 to k,w_r is the weight of documents at different place. It should be noted that, as r increases, the relevance of the top- K ranked unlabeled documents becomes weaker. Thus, intuitively, importance of different document should be different.

General approaches for deciding weight w_r


There are two straightforward choices for w_r: $w_r = 1/r$ puts more emphasis on the top positions, whereas $w_r = 1/K$ weights each position equally. Such weightings are heuristic and pre-determined, may not be flexible enough to deal with general cases. Instead, we introduce two flexible weighting schemes, which were shown to be superior in our experimental studies.

Weighting by Position Importance



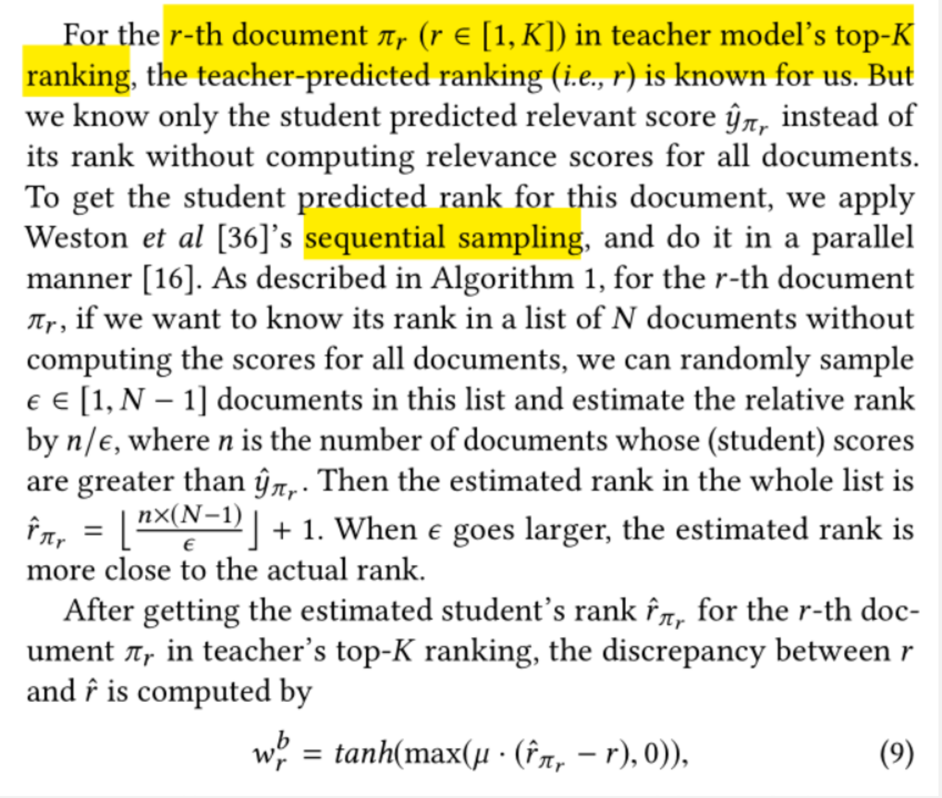
$$w_r^a \propto e^{-r/\lambda} \quad \text{and} \quad \lambda \in \mathbb{R}^+, \tag{8}$$

where λ is the hyperparameter that controls the sharpness of the distribution, and is searched through the validation set. When λ is small, this scheme puts more emphasis on top positions, and when λ is large enough, the distribution becomes the uniform distribution. This parametrization is easy to implement and configurable to each kind of situation.

Weighting by Ranking Discrepancy

The weighting by position importance is **static**, meaning that the weight at the same position is fixed during training process.

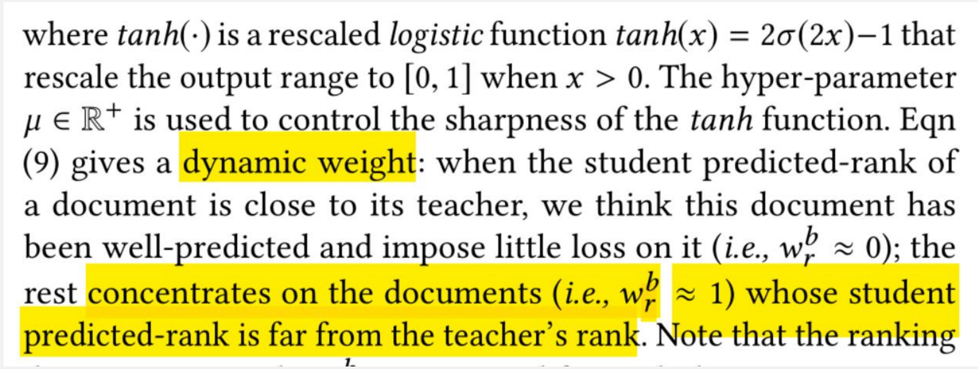
Our second scheme **is dynamic that considers the discrepancy** between the student-predicted rank and the teacher-predicted rank for a given unlabeled document, and uses it as another weight w^b. This weighting scheme allows the training to gradually concentrate on the documents in teacher’s top-K ranking that are not well-predicted by the student.



For the r-th document π_r ($r \in [1, K]$) in teacher model's top-K ranking, the teacher-predicted ranking (*i.e.*, r) is known for us. But we know only the student predicted relevant score $\hat{y}_{\pi_r}$ instead of its rank without computing relevance scores for all documents. To get the student predicted rank for this document, we apply Weston *et al* [36]'s sequential sampling, and do it in a parallel manner [16]. As described in Algorithm 1, for the r-th document π_r, if we want to know its rank in a list of N documents without computing the scores for all documents, we can randomly sample $\epsilon \in [1, N-1]$ documents in this list and estimate the relative rank by n/ϵ, where n is the number of documents whose (student) scores are greater than $\hat{y}_{\pi_r}$. Then the estimated rank in the whole list is $\hat{r}_{\pi_r} = \lfloor \frac{n \times (N-1)}{\epsilon} \rfloor + 1$. When ϵ goes larger, the estimated rank is more close to the actual rank.

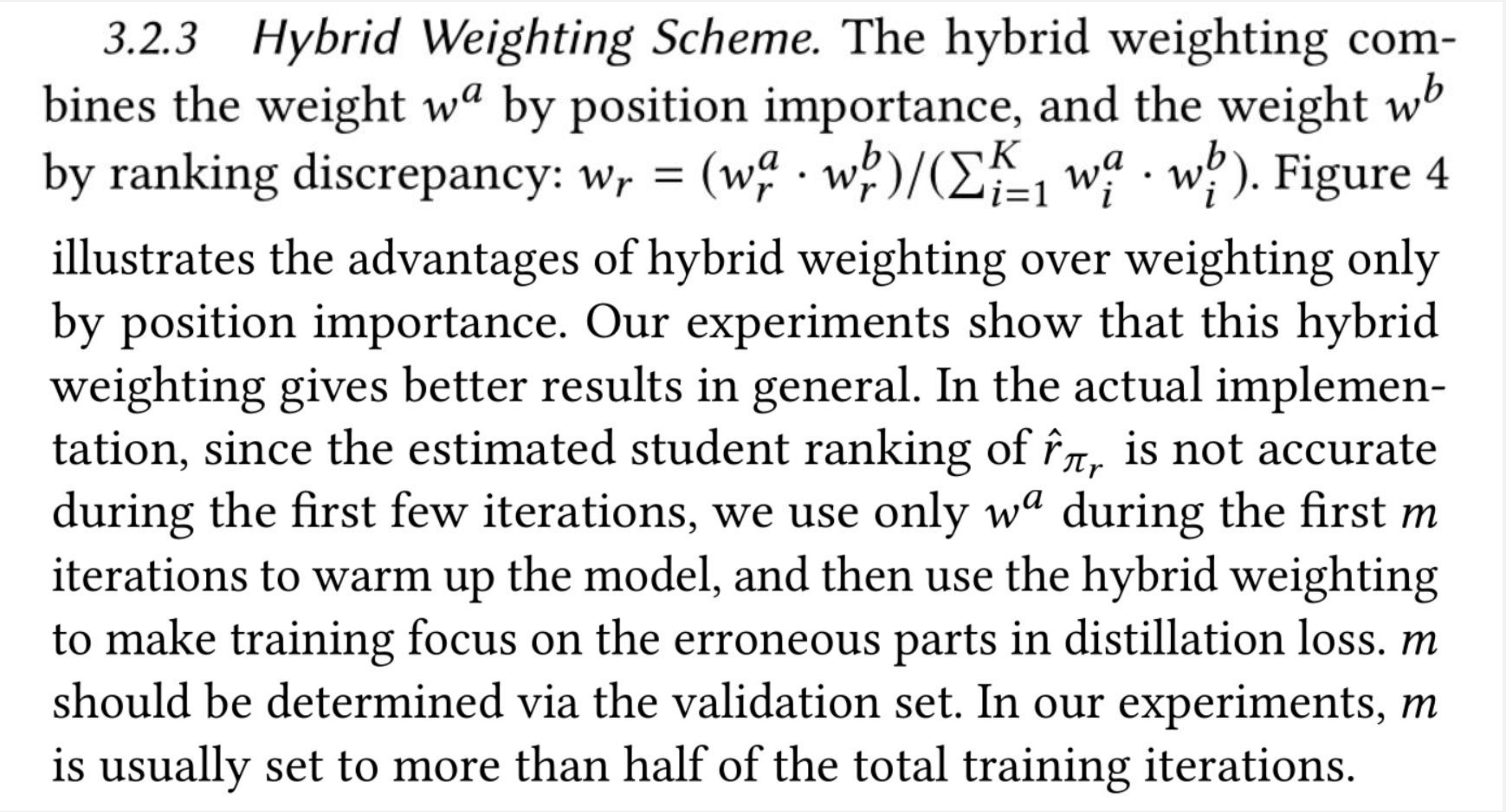
After getting the estimated student's rank $\hat{r}_{\pi_r}$ for the r-th document π_r in teacher's top-K ranking, the discrepancy between r and $\hat{r}$ is computed by

$$w_r^b = tanh(\max(\mu \cdot (\hat{r}_{\pi_r} - r), 0)), \quad (9)$$



where $tanh(\cdot)$ is a rescaled *logistic* function $tanh(x) = 2\sigma(2x) - 1$ that rescale the output range to $[0, 1]$ when $x > 0$. The hyper-parameter $\mu \in \mathbb{R}^+$ is used to control the sharpness of the *tanh* function. Eqn (9) gives a dynamic weight: when the student predicted-rank of a document is close to its teacher, we think this document has been well-predicted and impose little loss on it (*i.e.*, $w_r^b \approx 0$); the rest concentrates on the documents (*i.e.*, $w_r^b \approx 1$) whose student predicted-rank is far from the teacher's rank. Note that the ranking

Hybrid Weighting Scheme



3.2.3 *Hybrid Weighting Scheme.* The hybrid weighting combines the weight w^a by position importance, and the weight w^b by ranking discrepancy: $w_r = (w_r^a \cdot w_r^b)/(\sum_{i=1}^{K} w_i^a \cdot w_i^b)$. Figure 4 illustrates the advantages of hybrid weighting over weighting only by position importance. Our experiments show that this hybrid weighting gives better results in general. In the actual implementation, since the estimated student ranking of $\hat{r}_{\pi_r}$ is not accurate during the first few iterations, we use only w^a during the first m iterations to warm up the model, and then use the hybrid weighting to make training focus on the erroneous parts in distillation loss. m should be determined via the validation set. In our experiments, m is usually set to more than half of the total training iterations.

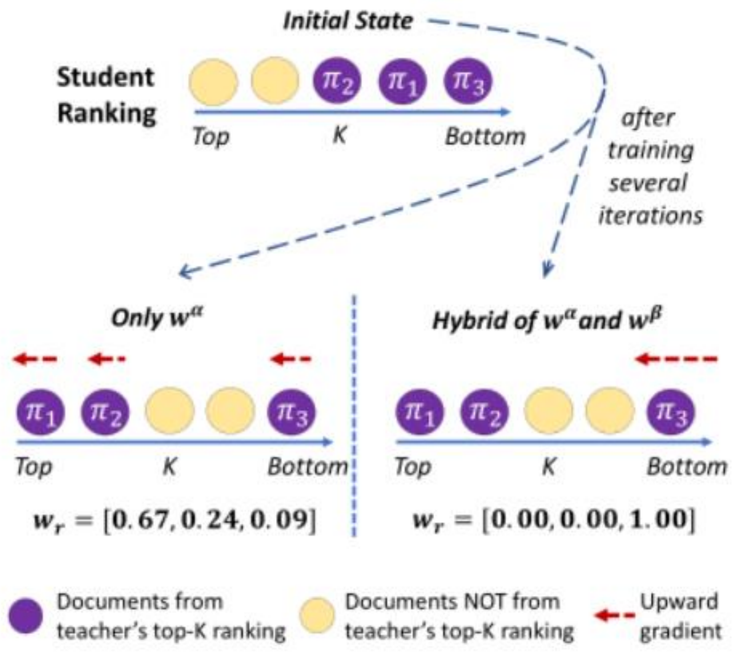

Initial State
Student Ranking
π_2
π_1
π_3
Top
K
Bottom
after training several iterations
Only w^α
π_1
π_2
π_3
Top
K
Bottom
$w_r = [0.67, 0.24, 0.09]$
Hybrid of w^α and w^β
π_1
π_2
π_3
Top
K
Bottom
$w_r = [0.00, 0.00, 1.00]$
Documents from teacher's top-K ranking
Documents NOT from teacher's top-K ranking
Upward gradient

Figure 4: An illustration of hybrid weighting scheme. We use $K = 3$ in this example.

Experiments

Datasets

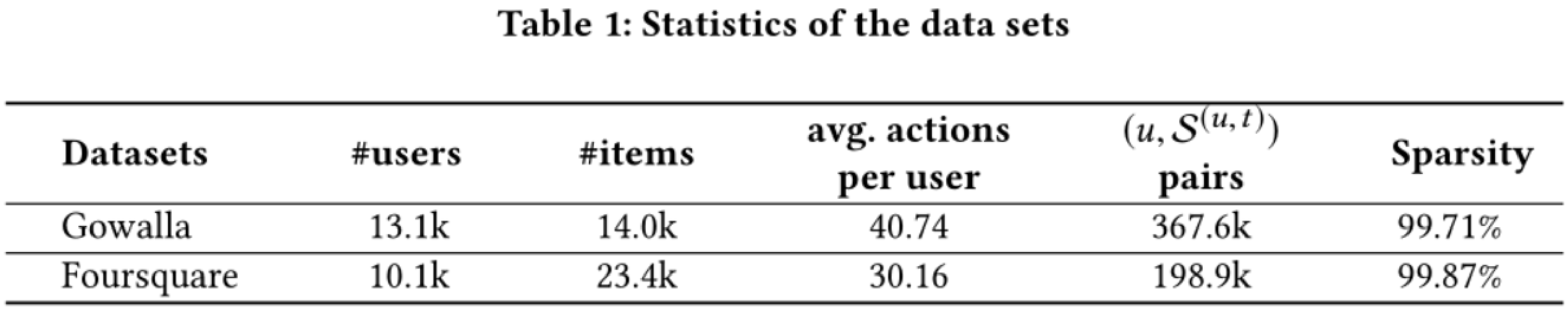
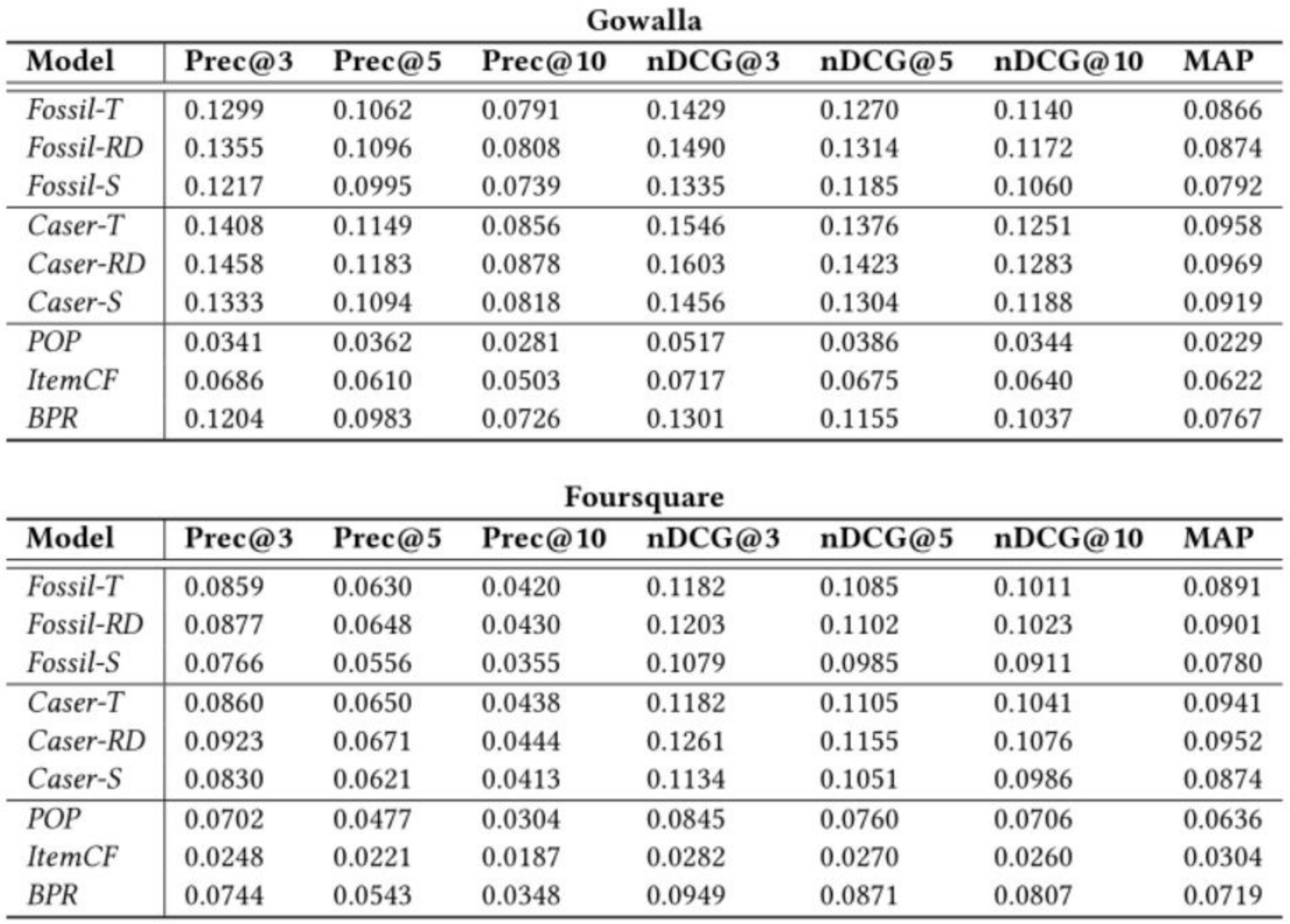


Table 1: Statistics of the data sets

Datasets	#users	#items	avg. actions per user	$(u, \mathcal{S}^{(u,t)})$ pairs	Sparsity
Gowalla	13.1k	14.0k	40.74	367.6k	99.71%
Foursquare	10.1k	23.4k	30.16	198.9k	99.87%

Performance comparison

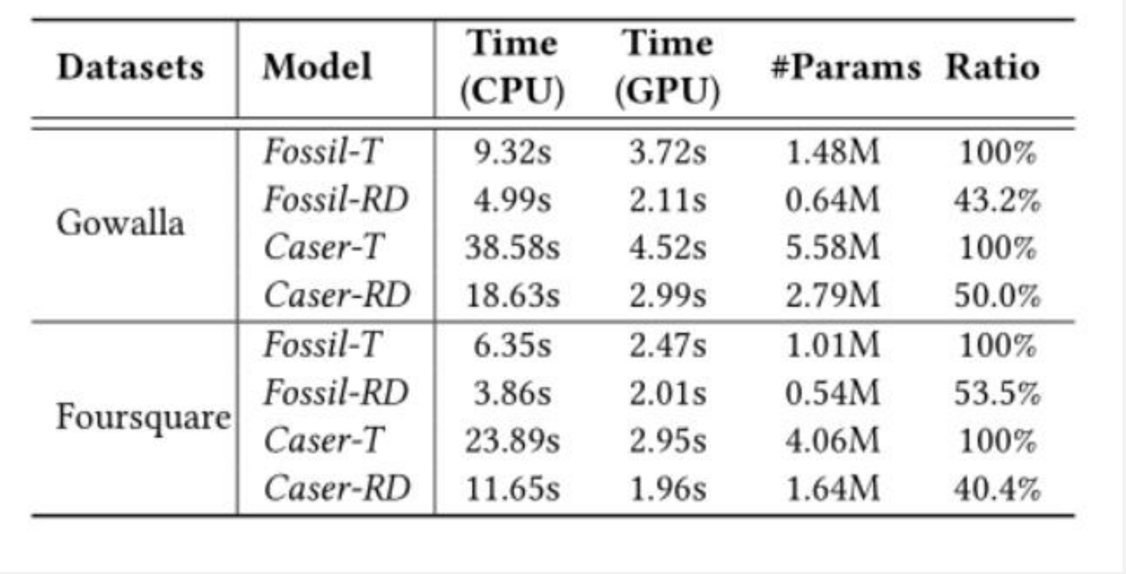


Gowalla

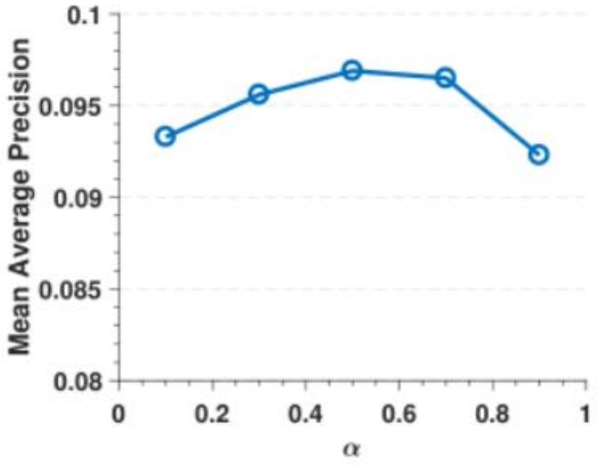
Model	Prec@3	Prec@5	Prec@10	nDCG@3	nDCG@5	nDCG@10	MAP
Fossil-T	0.1299	0.1062	0.0791	0.1429	0.1270	0.1140	0.0866
Fossil-RD	0.1355	0.1096	0.0808	0.1490	0.1314	0.1172	0.0874
Fossil-S	0.1217	0.0995	0.0739	0.1335	0.1185	0.1060	0.0792
Caser-T	0.1408	0.1149	0.0856	0.1546	0.1376	0.1251	0.0958
Caser-RD	0.1458	0.1183	0.0878	0.1603	0.1423	0.1283	0.0969
Caser-S	0.1333	0.1094	0.0818	0.1456	0.1304	0.1188	0.0919
POP	0.0341	0.0362	0.0281	0.0517	0.0386	0.0344	0.0229
ItemCF	0.0686	0.0610	0.0503	0.0717	0.0675	0.0640	0.0622
BPR	0.1204	0.0983	0.0726	0.1301	0.1155	0.1037	0.0767

Foursquare

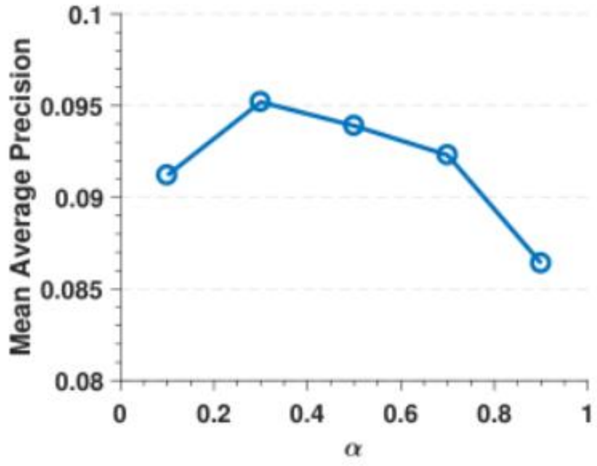
Model	Prec@3	Prec@5	Prec@10	nDCG@3	nDCG@5	nDCG@10	MAP
Fossil-T	0.0859	0.0630	0.0420	0.1182	0.1085	0.1011	0.0891
Fossil-RD	0.0877	0.0648	0.0430	0.1203	0.1102	0.1023	0.0901
Fossil-S	0.0766	0.0556	0.0355	0.1079	0.0985	0.0911	0.0780
Caser-T	0.0860	0.0650	0.0438	0.1182	0.1105	0.1041	0.0941
Caser-RD	0.0923	0.0671	0.0444	0.1261	0.1155	0.1076	0.0952
Caser-S	0.0830	0.0621	0.0413	0.1134	0.1051	0.0986	0.0874
POP	0.0702	0.0477	0.0304	0.0845	0.0760	0.0706	0.0636
ItemCF	0.0248	0.0221	0.0187	0.0282	0.0270	0.0260	0.0304
BPR	0.0744	0.0543	0.0348	0.0949	0.0871	0.0807	0.0719



Datasets	Model	Time (CPU)	Time (GPU)	#Params	Ratio
Gowalla	*Fossil-T*	9.32s	3.72s	1.48M	100%
	Fossil-RD	4.99s	2.11s	0.64M	43.2%
	Caser-T	38.58s	4.52s	5.58M	100%
	Caser-RD	18.63s	2.99s	2.79M	50.0%
Foursquare	*Fossil-T*	6.35s	2.47s	1.01M	100%
	Fossil-RD	3.86s	2.01s	0.54M	53.5%
	Caser-T	23.89s	2.95s	4.06M	100%
	Caser-RD	11.65s	1.96s	1.64M	40.4%



(a) Gowalla



(b) Foursquare

Figure 6: MAP vs. balancing parameter α

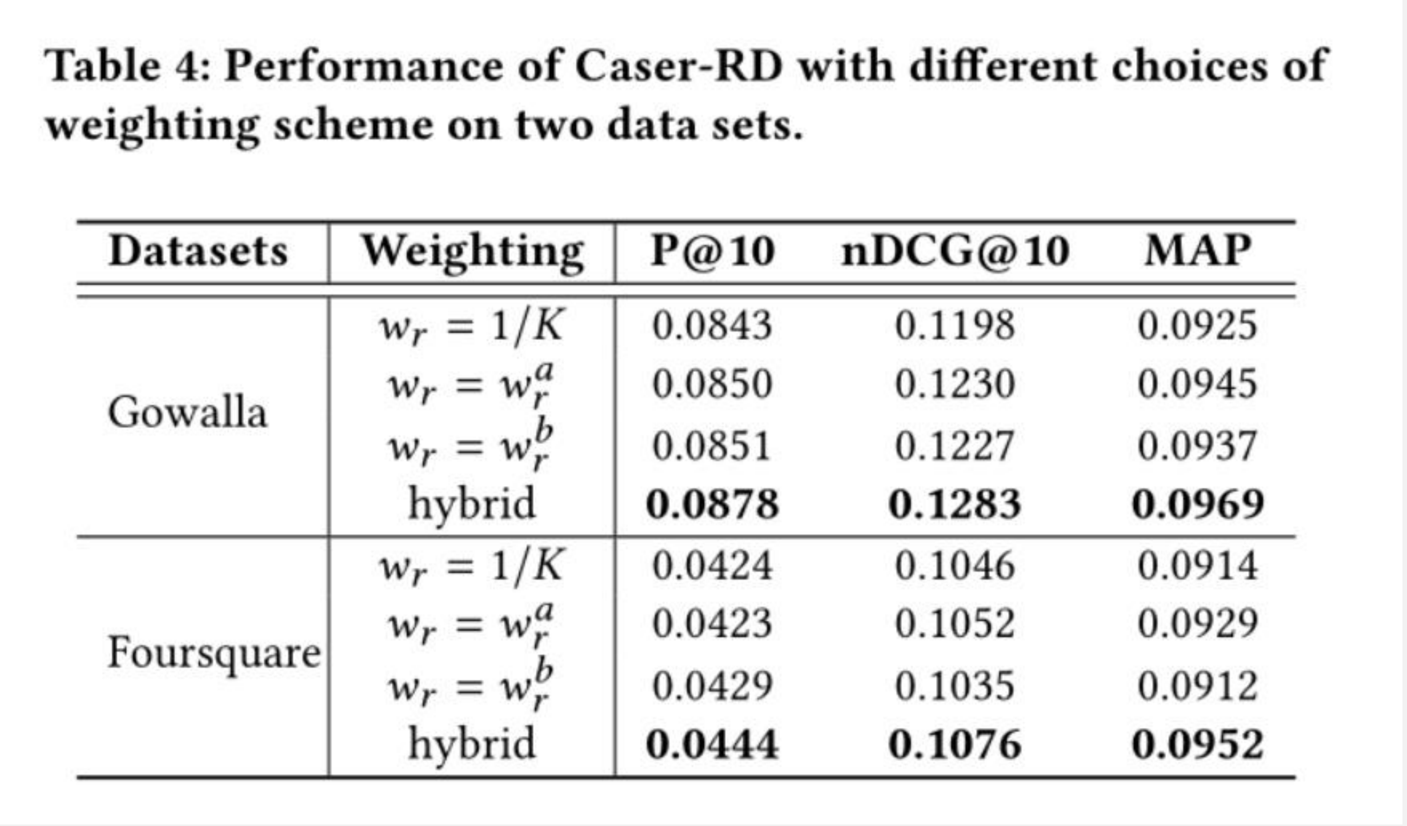


Table 4: **Performance of Caser-RD with different choices of weighting scheme on two data sets.**

Datasets	Weighting	P@10	nDCG@10	MAP
Gowalla	$w_r = 1/K$	0.0843	0.1198	0.0925
	$w_r = w_r^a$	0.0850	0.1230	0.0945
	$w_r = w_r^b$	0.0851	0.1227	0.0937
	hybrid	**0.0878**	**0.1283**	**0.0969**
Foursquare	$w_r = 1/K$	0.0424	0.1046	0.0914
	$w_r = w_r^a$	0.0423	0.1052	0.0929
	$w_r = w_r^b$	0.0429	0.1035	0.0912
	hybrid	**0.0444**	**0.1076**	**0.0952**

Thanks