JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 20, No. 7, pp. 1401-1408, Jul. 2019

개인화 추천 시스템의 온라인 러닝을 위한 확률 블룸 필터

유 찬 우[1] · 강 환 수[2] · 김 희 천[3*]
[1]라인플러스
[2]동양미래대학교 컴퓨터정보공학과
[3]한국방송통신대학교 컴퓨터과학과

Probabilistic Bloom Filter for Online Learning in Personalized Recommender Systems

Chan-Woo Yoo[1] · Hwan-Soo Kang[2] · Hee-Chern Kim[3]*

[1]Line Plus Corporation

[2]Department of Computer Information Engineering, Dongyang Mirae University, Seoul, Korea

[3]Department of Computer Science, Korea National Open University, Seoul, Korea

[요 약]

블룸 필터는 적은 공간으로 항목을 기억하는 특성을 가지므로 머신러닝 분야에서 임베딩에 사용된다. 온라인 러닝에서는 새 항목이 등장할 때 디케이를 적용함으로써 블룸 필터가 가득 차지 않도록 하는 특별한 블룸 필터가 필요하다. 개인화 추천 관련 데이터를 사용하여 정확성과 항목간 유사성 유지의 관점에서 디케이 적용이 가능한 블룸 필터들을 비교하였다. 또한 두 가지 장점을 가지는 새로운 '확률 블룸 필터'를 제안하였다. 첫 번째 장점은 유사성 기준과의 트레이드-오프는 있으나 정확성에 있어서 다른 필터에 비해 우수하며 종합적 성능에서는 안정 블룸 필터 다음으로 우수하다는 점이다. 두 번째는 장기적으로 파라미터의 계속적 조정 없이도 필터가 가득 차거나 또는 비는 일이 일어나지 않는다는 점이다.

[Abstract]

Bloom filters are used as embeddings in machine learning domain because of its characteristic of remembering items with less storage. As to online learning, a specific kind of bloom filter is needed, which can prevent filters being full as new items appear by decay. We compared bloom filters which can decay in perspective of precision and the degree of preservation of similarities between items using dataset for personalized recommendation. We also suggested a new bloom filter named as 'probabilistic bloom filter', which has two advantages. First, it showed higher precision than other filters at the expense of similarities between items while overall performance was the second to 'stable bloom filter' when it comes to precision-similarity trade-off. Second, even in a long term, it ensures that filters are not full or empty without continuous parameter calibration.

색인어 : 블룸 필터, 디케이, 임베딩, 온라인 러닝, 추천 시스템

Key word : Bloom Filter, Decay, Embedding, Online Learning, Recommender System

http://dx.doi.org/10.9728/dcs.2019.20.7.1401

Received 13 June 2019; **Revised** 10 July 2019
Accepted 25 July 2019

***Corresponding Author; Hee-Chern Kim**

Tel: +82-2-3668-4657
E-mail: hckim@knou.ac.kr

Ⅰ. Introduction

In machine learning domain, the importance of online learning is growing. Although cloud environment to deal with big data is rising, there is a limit to data digestion capacity and it is still painful to use large batch data for training at once. Online learning can reduce the pain by using a handful of data each time. While online learning has the property to make big data manageable, it is also important to quickly reflect the concept drift.

The importance of embedding is also growing. In domains with a large users and items, how to reduce dimensions is an important issue in modeling. Bloom filters are emerging as storage saving[1] and embeddings in machine learning domain, especially in recommender systems[2][3][4].

In online learning environment, since new data keeps coming in, there is a need to reset the existing cells of a bloom filter in order to prevent bloom filters becoming full, and there are some algorithms related to it. However, there is no comparison as to what characteristics each algorithm shows in the online learning environment.

In this paper, we propose a new decay bloom filter which has a different characteristic and compare algorithms in online learning situation of a recommender system.

Ⅱ. Related Work

Stable bloom filter[5] sets cell values as a predefined max cell value when inserting an item. When decaying, it decreases 1 on randomly selected cells. It provides an algorithm to decide the number of cells to be randomly chosen where a target false positive rate, a dimension of bloom filter, and a max cell value are given. This makes it easier to apply stable bloom filter in practice. In addition, unlike double buffering bloom filter which uses two filters, a warm-up filter and an active filter, stable bloom Filter uses a single filter, so it costs a half amount of storage compared to double buffering bloom filter.

Temporal counting bloom filter[6] is similar to stable bloom filter. It sets cells as 'initial counter value' which is the same as max cell value of stable bloom filter. The difference is that temporal counting bloom filter decreases a cell value as much as 'decay factor', which is calculated by a provided algorithm, rather than constant 1. Temporal counting bloom filter also uses the same amount as stable bloom filter.

Double buffering bloom filter[7] uses two filters, a warm-up filter and an active filter. When inserting an item, if the item is in the active filter, it checks if it also exists in the warm-up filter, then set corresponding cells of warm-up filter only when the active filter is filled more than half of its capacity. If the item is not in the active filter, it inserts the item into the active filter, then also insert it to the warm-up filter only when the active filter is filled more than half. Decaying of the bloom filter, 'renewing' can be more appropriate in this case, is performed by exchanging the warm-up filters and the active filters when the active filters are full.

There are works in which bloom filters are used as embeddings which represent items or users of recommender systems[2][3][4]. Usually, the dimension of user or item vectors are enormous, dimensionality reduction is inevitable, and bloom filters can be a way of reducing dimensions. For example, a user embedding can be made by setting cells of a bloom filter of which indexes are generated by applying hashing with the value of an item. An item embedding can be generated in the same way.

Ⅲ. Probabilistic Bloom Filter

We suggest probabilistic bloom filter which resets a cell in proportion to the inverse of the count of the cell, which results in longer existence of important cells. We regard a cell as being 'important' when its count is high because it is used to check existence of many items.

Inserting is performed by adding 1 to all corresponding cells, and it updates the total sum of each count of cells and the number of non-zero cells of the filter. To make the filter decay, it first calculates the number of cells to reset to zero by subtracting a predefined configured number of non-zero cells from the current number of non-zero cells. Then it chooses non-zero cells to reset with probabilities in proportion to the inverse of the count of the cells. Inserting and decaying procedures are shown in Figure1 and Figure 2, respectively.

```
index_list = hash(item)
for index in index_list:
    cell_value_sum += 1
    if bloom_filter[index] == 0:
        bloom_filter[index] = 1
        non_zero_cell_count += 1
    else:
        bloom_filter[index] += 1
```

그림 1. 블룸 필터에 항목을 삽입하는 알고리즘
Fig 1. Algorithm for inserting an item into the bloom filter

```
if non_zero_cell_count > threshold:
    number_of_cells_to_reset = non_zero_cell_count - threshold
    randomly choose cells with a probability in proportion to 1 / bloom_filter[index]
    reset chosen cells to zero
    decrease cell_value_sum and non_zero_cell_count by calculated amounts
```

그림 2. 블룸 필터에 디케이를 적용하는 알고리즘
Fig 2. Algorithm for Decaying of the bloom filter

Since it always maintains a specific number of zero cells, it never becomes full and users don't have to worry about how to find out the correct parameter for decay. This problem about deciding parameters related to decay is not trivial in online learning. For stable bloom filter and temporal counting bloom filter, algorithms are provided to decide the number of cells to decay or the decay factor, but the resulting values are not usually integers. So users can't help using rounded values which can result in not guaranteeing properties like false positive rate when they are used over a long period in online learning situations.

Ⅳ. Comparison between decay bloom filters

We wanted to see how each bloom filter performs as an embedding in a context of online learning of a recommender system. So we made a bloom filter represent a user by inserting items which the user interacted with into the bloom filter. As a result, bloom filters were created as many as the number of users.

Movielens[8] dataset was used for the experiment because it has been used many times as a benchmark dataset for online learning[9][10][11] and personalized recommendation[12]. u1.test ~ u4.test of ml-100k dataset were used to calculate 4-fold cross validation results on various parameters. Only 4 and 5 ratings were treated as positive feedbacks and corresponding items were inserted into a user's bloom filter. Records of the dataset were sorted by their timestamps and the insertions were made by time since we meant to simulate an online learning situation.

Evaluation as to how well a bloom filter performs as an embedding was done in two aspects. The first is about precision of a bloom filter. It describes how accurately a filter judge if an item is stored in the filter. We measured the mean of precision of users' bloom filters by checking if all items appeared in users' records. The second is about the degree of preservation of similarity between embeddings. Good user embeddings should preserve close users closely after the dimensionality reduction is done. We measured the degree of preservation of cosine similarity using rank correlation(Kendall's tau) between two similarity rank sets - one was generated by bloom filter embeddings and the other was generated by user vectors without dimensionality reduction of which dimension is the number of items. Cosine similarity ranks were generated between a user and all the other users, and this procedure was repeated on all users.

The experimental parameters for each bloom filter are shown in Table 1. The dimension of filters and the number of hashing are commonly applied to all bloom filters. For stable bloom filter, the number of cells to decay is decided by other parameters like cell max and false positive rate using the provided algorithm[5]. Likewise, the decay factor of temporal counting bloom filter is calculated with other parameters by the algorithm[6]. As to double buffering bloom filter, various thresholds are examined for double buffering bloom filter rather than only 'half' for inserting an item into a warm-up filter and 'full' for exchanging a warm-up filter and an active filter because a full bloom filter is not so much useful as an embedding and there is a possibility that other thresholds for inserting are better than the threshold of 'half' when domain differs. Probabilistic bloom filter has the smallest number of parameters among all, and actually non-zero cell ratio, which means how much the filter can be filled maximally (0.2 means that more than 20% cells cannot be set), is the only one needed.

표 1. 실험에서 사용된 블룸 필터의 파라미터
Table 1. Parameters of Bloom Filters in Experiments

	Filter Dimension	Number of Hashing
Common for all bloom filters	10, 20, ..., 150	2, 3
	Cell Max	False Positive Rate
Stable Bloom Filter	3, 5, 10	0.1, 0.2, ..., 0.9
	Initial Counter Value	False Positive Rate
Temporal Counting Bloom Filter	3, 5, 10	0.1, 0.2, ..., 0.9
	Warm-Up Threshold	Swap Threshold
Double Buffering Bloom Filter	0.3, 0.4, 0.5	0.7, 0.8, 0.9, 1.0
	Non-zero Cell Ratio	
Probabilistic Bloom Filter	0.2, 0.3, ..., 0.9	

Some experimental results are shown in Figure 3~6. Cyan is stable bloom filter, blue is temporal counting bloom filter, green is double buffering bloom filter, and red is probabilistic bloom filter. Results of every dimension can be found in Appendix. Since there is a tradeoff between precision and rank correlation, there are no such 'best' parameters per bloom filters. So the results need to be interpreted by judging how far each graph of bloom

filters appears from its origin as a whole. When the dimension of bloom filters was 20, stable bloom filter showed the best results, followed by probabilistic bloom filter and double buffering bloom filter, and temporal counting bloom filter came at last. When the dimension was increased to 40, 60, and 80, stable bloom filter was the best and the second was the probability bloom filter. Beside the results of overall performance, a unique trait of probabilistic bloom filter was found by this experiment. It shows far better precision than other filters when the dimension increases. For example, in Figure 5, the best precision achieved by probabilistic bloom filter is about 0.17 and no other filters reach near the value. This can be a useful property when precision is especially more important than rank correlation. We suppose that this result is due to the trait of probabilistic bloom filter that rarely resets important cells which are set by many items.

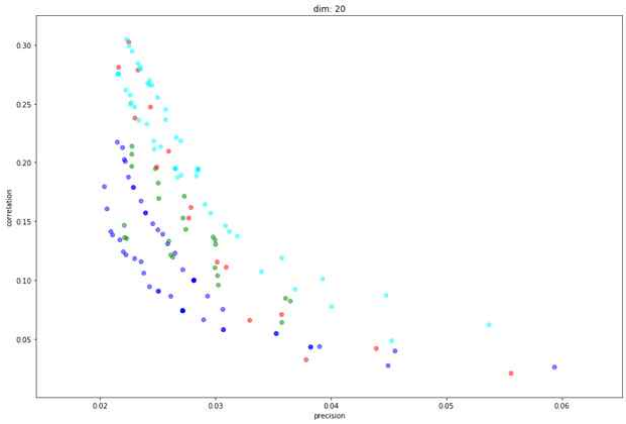


그림 3. 블룸 필터의 차원이 20인 경우
(청녹- 안정 블룸 필터, 파랑- 시간 카운팅 블룸 필터, 녹색- 더블 버퍼링 블룸 필터, 빨강- 확률 블룸 필터)
Fig 3. Comparison of correlation and precision with dimension of bloom filters = 20
(Cyan- stable bloom filter, Blue- temporal counting bloom filter, Green- double buffering bloom filter, Red- probabilistic bloom filter)

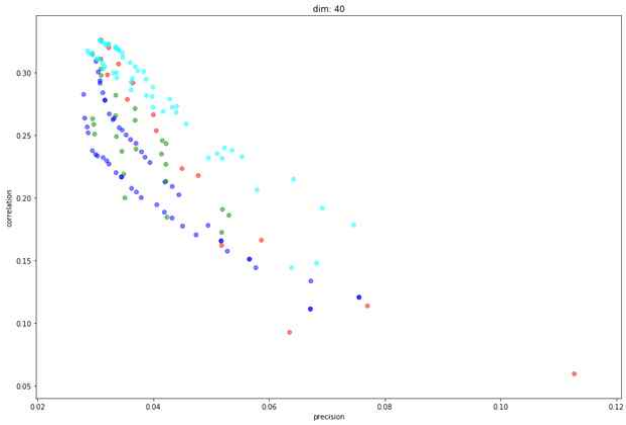


그림 4. 블룸 필터의 차원이 40인 경우
Fig 4. Comparison of correlation and precision with dimension of bloom filters = 40

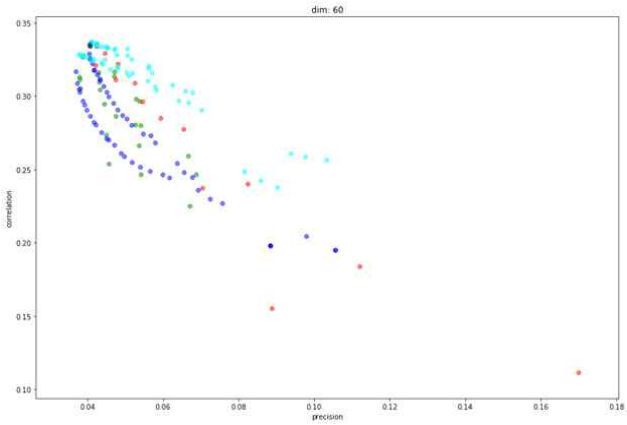


그림 5. 블룸 필터의 차원이 60인 경우
Fig 5. Comparison of correlation and precision with dimension of bloom filters = 60

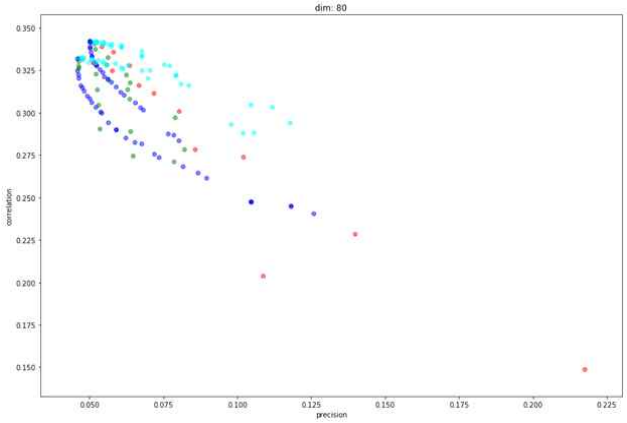


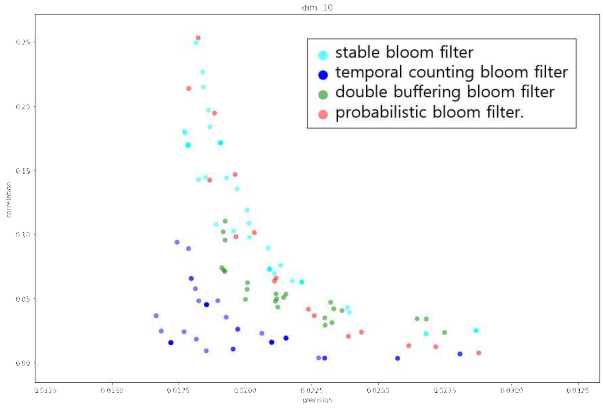
그림 6. 블룸 필터의 차원이 80인 경우
Fig 6. Comparison of correlation and precision with dimension of bloom filters = 80

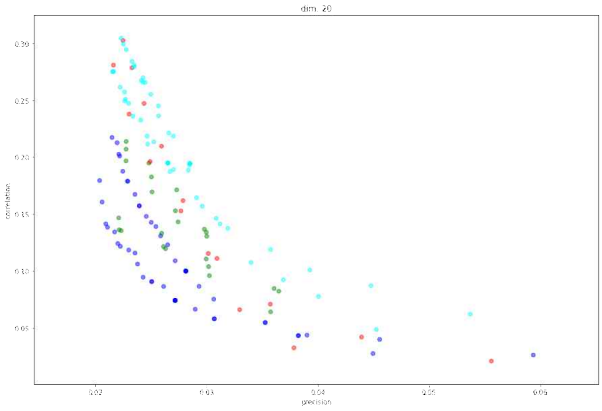
Ⅴ. Conclusion

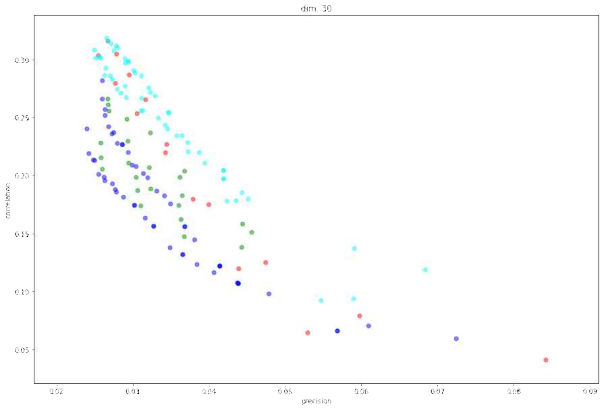
We compared various bloom filters which can decay as embeddings in online learning context of recommender systems. Besides existing bloom filters, we suggested 'probabilistic bloom filter' which can remember important cells longer. It never makes bloom filters full or emptier than necessary in an online learning situation while other bloom filters can suffer such problems because the calculated number of cells to decay or a decay factor must be rounded to become an integer. The overall performance over precision - rank correlation tradeoff was measured using Movielens dataset. Stable bloom filter showed the best results overall in this setting and probabilistic bloom filter was the second. When it comes to precision, probabilistic bloom filter showed that it can achieve higher precision than all the other bloom filters at the expense of rank correlation, so we expect it to be used in an online learning situation where precision is more important than rank correlation.

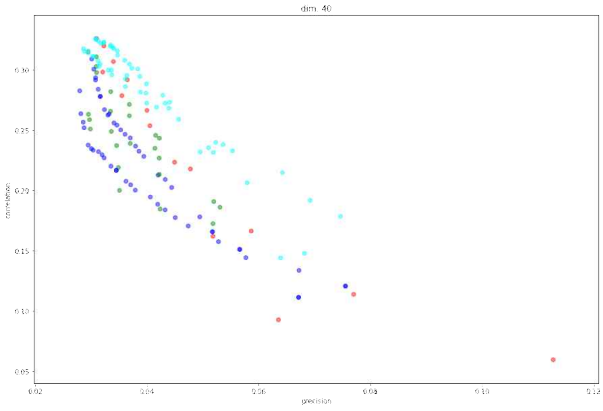
Appendix

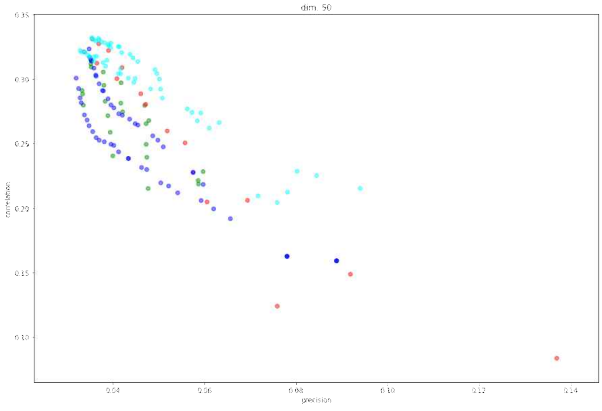
The figures below show the experimental results comparing the correlation and precision of bloom filters according to the dimension size of filters.

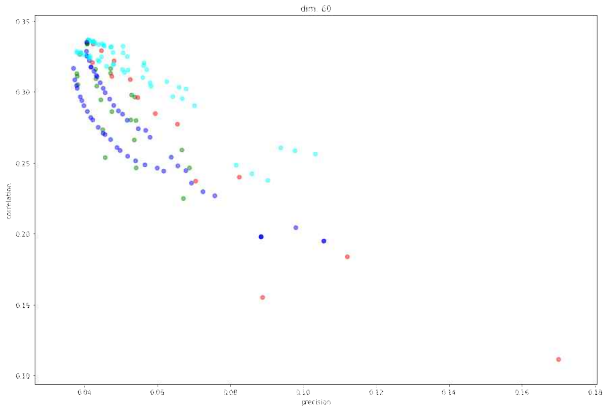


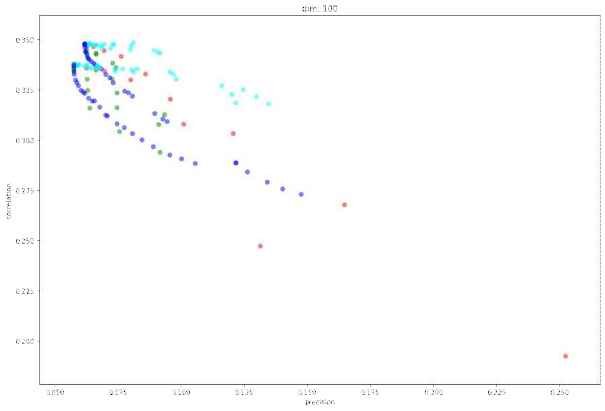


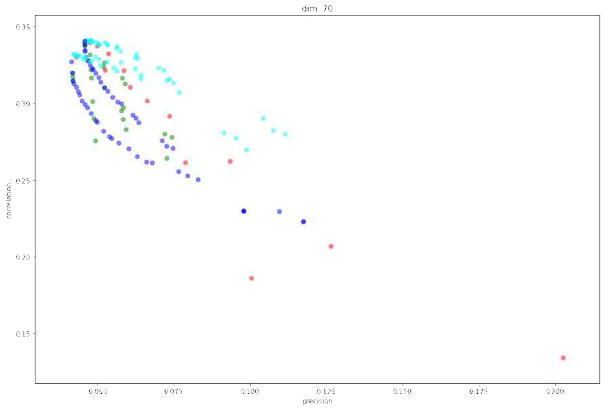


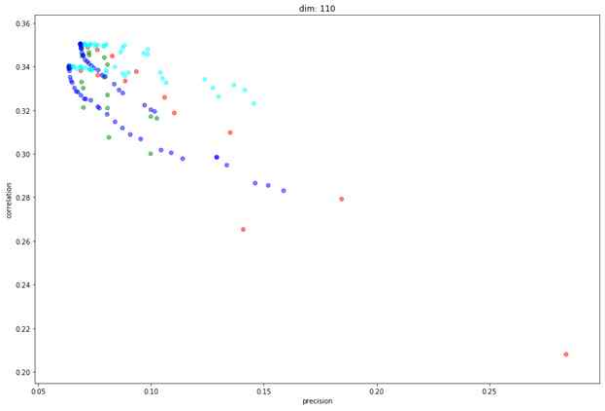


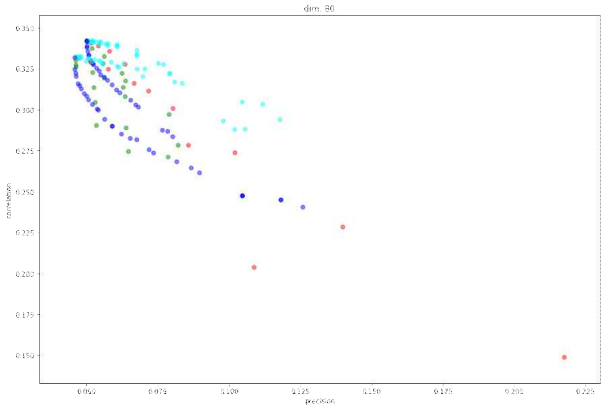


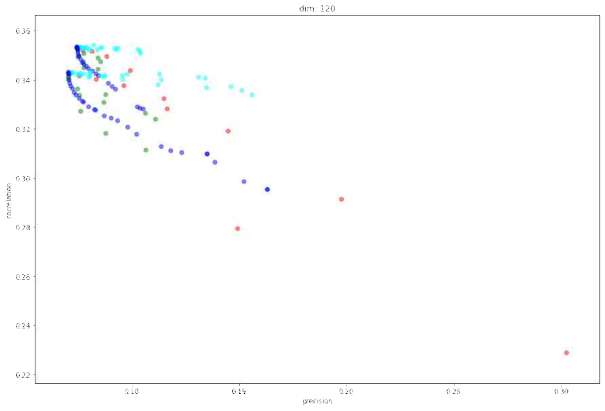

dim: 60
correlation
precision

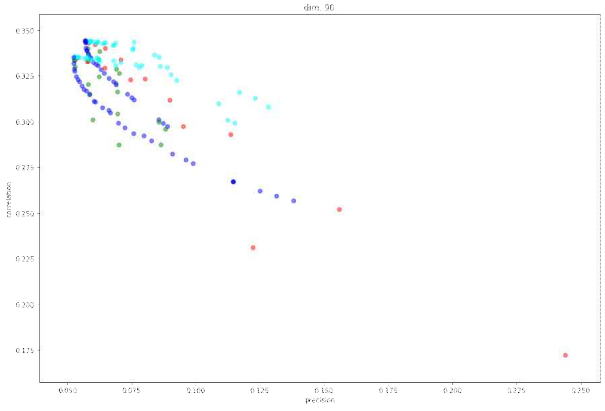

dim: 100
correlation
precision

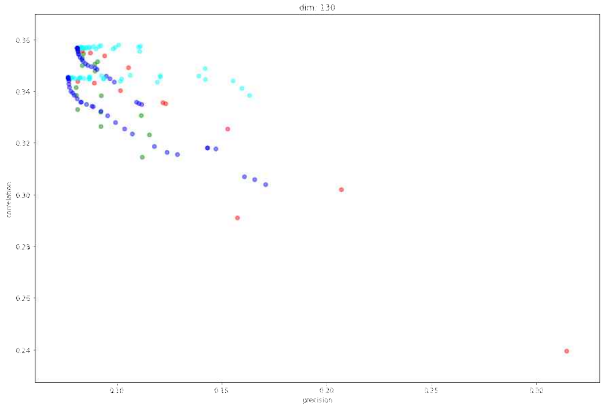

dim: 70
correlation
precision

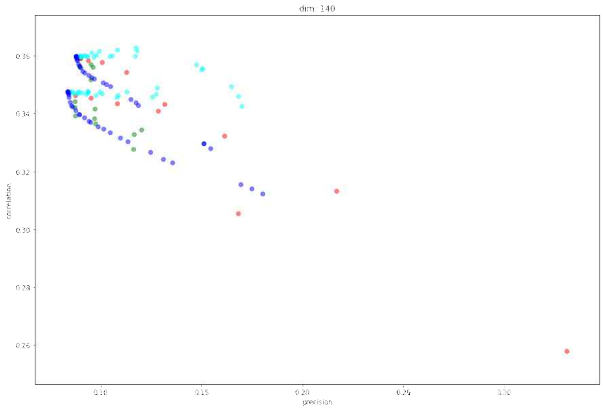

dim: 110
correlation
precision

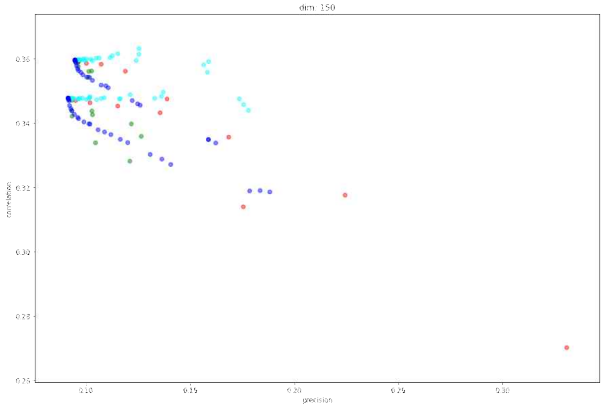

dim: 80
correlation
precision


dim: 120
correlation
precision


dim: 90
correlation
precision


dim: 130
correlation
precision





Acknowledgements

This research was supported by Korea National Open University Research Fund in 2017.

References

[1] McMahan, H. Brendan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie et al., "Ad click prediction: a view from the trenches," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, Chicago, USA, pp. 1222-1230, Aug. 2013.

[2] Pagare, Reena, and Anita Shinde, "Recommendation system using bloom filter in mapreduce," *International Journal of Data Mining & Knowledge Management Process*, Vol. 3, No. 6, pp. 127-134, Nov. 2013.

[3] Pozo, Manuel, Raja Chiky, Farid Meziane, and Elisabeth Métais, "An item/user representation for Recommender systems based on Bloom filters," in *IEEE Tenth International Conference on Research Challenges in Information Science* (RCIS 2016), Grenoble, France, pp. 1-12, June 2016.

[4] Serrà, Joan, and Alexandros Karatzoglou, "Getting deep recommenders fit: Bloom embeddings for sparse binary input/output networks," in *Proceedings of the Eleventh ACM Conference on Recommender Systems*, Como, Italy, pp. 279-287, Aug. 2017.

[5] Deng, Fan, and Davood Rafiei, "Approximately detecting duplicates for streaming data using stable bloom filters," in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, Chicago, USA, pp. 25-36, June 2006.

[6] Zhao, Yaxiong, and Jie Wu, "The design and evaluation of an information sharing system for human networks," *IEEE Transactions on Parallel and Distributed Systems*, Vol 25, No. 3, pp. 796-805, Mar. 2014.

[7] Chang, Francis, Wu-chang Feng, and Kang Li, "Approximate caches for packet classification," in *IEEE INFOCOM* (Vol. 4), HongKong, China, pp. 2196-2207, Mar. 2004.

[8] GroupLens at the Univ. of Minnesota. MovieLens 100K Dataset[Internet]. Available: https://grouplens.org/datasets/movielens/100k/ (Accessed 03 June 2019)

[9] Blondel, Mathieu, Akinori Fujino, and Naonori Ueda, "Convex factorization machines," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Porto, Portugal, pp. 19-35, Sep. 2015.

[10] Yamada, Makoto, Wenzhao Lian, Amit Goyal, Jianhui Chen, Kishan Wimalawarne, Suleiman A. Khan, Samuel Kaski, Hiroshi Mamitsuka, and Yi Chang, "Convex factorization machine for toxicogenomics prediction," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Halifax, Canada, pp. 1215-1224, Aug. 2017.

[11] Lin, Xiao, Wenpeng Zhang, Min Zhang, Wenwu Zhu, Jian Pei, Peilin Zhao, and Junzhou Huang, "Online Compact Convexified Factorization Machine," in *Proceedings of the 2018 World Wide Web Conference*, Lyon, France, pp. 1633-1642, Apr. 2018.

[12] Wang, Jun, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang, "Irgan: A minimax game for unifying generative and discriminative information retrieval models," in *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, Shinjuku, Japan, pp. 515-524, Aug. 2017.

유찬우(Yoo, Chan Woo)

2003년 : 서울대학교 (컴퓨터공학 학사, 경영학 학사)
2012년 : 서울대학교 대학원 (공학박사-소프트웨어 공학)
2012년~2014년: LG전자
2016년~2018년: 네무스텍

2018년~현 재: 라인플러스 연구원
※관심분야 : 머신러닝, 온라인 러닝, 추천 알고리즘

강환수(Kang, Hwan Soo)

1991년 : 서울대학교 대학원 (이학석사)
2002년 : 서울대학교 대학원 (공학박사수료-컴퓨터그래픽스)

1992년~1998년: 삼성에스디에스
1998년~현 재: 동양미래대학교 컴퓨터정보공학과 교수
※관심분야 : 컴퓨터교육, 인공지능, 프로그래밍언어

김희천(Kim, Hee Chern)

1989년 : 서울대학교 (이학사)
1991년 : 서울대학교 대학원 (이학석사)
1998년 : 서울대학교 대학원 (이학박사)

2004년~현 재: 한국방송통신대학교 컴퓨터과학과 교수
※관심분야 : 머신러닝, 소프트웨어 공학, 컴퓨터교육