Active learning in recommender systems for predicting vulnerabilities in software

A thesis for the master Artificial Intelligence

Author
E.J. Stijger
s6472346

First examiner
dr. G.M. Krempl
Utrecht University

Second examiner
dr. M. van Ommen
Utrecht University

Daily supervisor
Amber Poppelier
Ordina

External advisors
Lukas Wegmeth
Tobias Vente
University of Siegen

Department of Information and Computing Sciences
Utrecht University
The Netherlands
October, 2023

Abstract

Due to a rapid advancement of digital technology and growing reliance on the internet, cybersecurity has become a paramount issue for individuals, organizations, and governments. To address this challenge, penetration testing has emerged as a critical tool to ensure the security of computer systems and networks. The reconnaissance phase of penetration testing plays a crucial role in identifying vulnerabilities in a system by gathering relevant information. Although various tools are available to automate this process, most of them are limited to identifying reported vulnerabilities, and they do not provide suggestions or predictions about vulnerabilities. Therefore, this research aims to investigate the application of recommender systems to predict common vulnerabilities during the reconnaissance phase. The main objective of this research is to investigate how active learning affects the performance of a recommender system to identify vulnerabilities in software products.

Item-Based k-NN Collaborative Filtering, a recommender system, can improve the identification of potential vulnerabilities and the effectiveness of penetration testing by analyzing information from similar data points. This research involves a comprehensive data preprocessing phase, which utilizes data from the National Vulnerability Database (NVD). Several recommender systems are built using this data, which enables the prediction of potential vulnerabilities during the reconnaissance phase of penetration testing. The performances of these recommender systems are evaluated, and the top-performing recommender system implements active learning to enhance its performance.

The findings of this research demonstrate that Item-Based k-NN Collaborative Filtering outperforms other recommender systems in terms of overall performance when it comes to identifying software vulnerabilities. Furthermore, when compared to Item-Based k-NN Collaborative Filtering prior to active learning or with active learning and a random sampling technique, Item-Based k-NN Collaborative Filtering with active learning incorporating a 4- or 10-batch sampling technique with 20 or 40 items added yields a statistically significant improvement in the precision score. This indicates that a greater proportion of the predicted vulnerabilities are correct. Item-Based k-NN Collaborative Filtering with active learning and a single-batch sampling strategy only results in a statistically significant improvement in precision, compared to Item-Based k-NN Collaborative Filtering prior active learning or with active learning and a random sampling technique, when 20 items are added instead of 40.

Furthermore, only Item-Based k-NN Collaborative Filtering with a 10-batch sampling strategy adding 20 items demonstrated a statistically significant improvement in nDCG scores compared to Item-Based k-NN Collaborative Filtering prior to active learning. This implies a more accurate ranking of the vulnerabilities. However, this could potentially be a type I error.

From these findings, it can be concluded that introducing active learning in Item-Based k-NN Collaborative Filtering, using the approaches outlined, leads to significant improvement in precision score but not necessarily in nDCG score.

Considering this conclusion, it is advised to use Item-Based k-NN Collaborative Filtering with active learning to predict vulnerabilities in software products and enhance the reconnaissance phase of penetration testing. This can be achieved by incorporating a single-batch sampling technique with 20 items added or a 4- or 10-batch sampling technique with 20 or 40 added.

The insights gained from this research can help individuals, organizations, and governments strengthen

their cybersecurity defences and protect against potential cyber threats.

Acknowledgements

During this project, I received lots of support from different people. I would like to thank them in this section.

First, I would like to express my appreciation to my first examiner and supervisor, Dr. Krempl, for his supervision and guidance throughout the entire process of this research. His insightful ideas about the implementation of recommender systems on cybersecurity data enriched the quality of this work. Moreover, his expertise in active learning was a valuable asset during this research. I would also like to thank my second examiner, dr. Van Ommen, for his time and examination.

Secondly, A special appreciation goes to Tobias Vente and Lukas Wegmeth for their significant contributions and knowledge about recommender systems. Their assistance in the data preprocessing phase, sharing of knowledge during the implementation of recommender systems, and advising on the evaluation of the recommender systems played an enormous role during this research.

Finally, I would like to thank Ordina, and especially Amber Poppelier, for her guidance and support through this project. Her Insights about cybersecurity provided context and more depth to my work. And the offering of a laptop with robust computational resources improved the efficiency of the project and it allowed me to run implementations of active learning that are computationally demanding.

These efforts together with the support of family and friends have been the key in finishing this research. I am grateful for the help they provided and the opportunity to do this investigation.

1 Introduction

This research will investigate the possibilities to improve the reconnaissance phase in ethical hacking by predicting vulnerabilities in software products with the application of Item-Based k-NN Collaborative Filtering and active learning. First, the motivation for this research is elaborated and a problem state is defined. Followed by the research questions.

1.1 Motivation

Over the past two decades, humans have become increasingly reliant on information networks, social media, and other digital networks. With such heavy reliance on these systems, the consequences of a cyber attack could be severe, ranging from financial losses to reputational damages. This creates the necessity for cybersecurity to strengthen cyber defence mechanisms and prevent cybercrime. Cybercriminals can break into a computer system by exploiting vulnerabilities in software or networks. These vulnerabilities can be found in the National Vulnerability Database (NVD) [1] and could be caused by password management flaws, operating system design flaws, software bugs, and many more things. To protect devices or networks a penetration test can be performed where ethical hackers investigate potential vulnerabilities. As an IT-consulting Organization, Ordina could perform such penetration tests. Ordina has a team of ethical hackers named the 'Red Team' who try to break into systems by conducting a penetration test to find vulnerabilities. This hack is done according the steps in Figure 1 [2]. In the first two phases 'preparation' and 'information gathering' the ethical hacker will gather information about the system and this information is later used to examine the network. However, these phases are time-consuming, especially for larger systems with multiple applications, software, and operating systems. It requires a lot of time to find and review every vulnerability in the NVD. To make this process more efficient, it would be beneficial to consider common vulnerabilities of similar systems. Artificial Intelligence and particularly recommender systems can discern patterns in data rapidly. When applied to cybersecurity data, a recommender system can discover potential new vulnerabilities in products. This would also discover potential vulnerabilities which are not yet reported in the NVD.

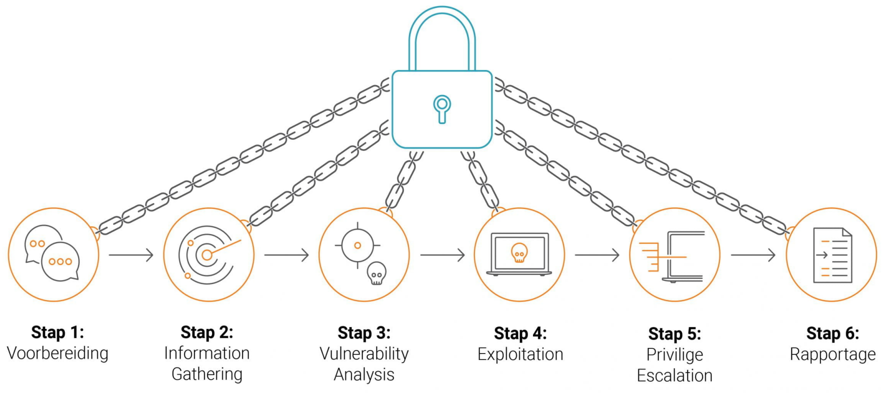


Figure 1: steps of penetration testing at Ordina

1.2 Problem space

This research will investigate whether the work efficiency during the reconnaissance phase of ethical hacking could be improved. Currently, there are a lot of tools to automate parts of this phase such as Metasploit, Nmap, Shodan, Wireshark etc. Most of them are information exploit tools and some are also capable of identifying vulnerabilities. However, this is based on reported vulnerabilities and there are no suggestions or predictions made about the existence of other possible vulnerabilities. Previous research has already explored machine learning for predicting vulnerabilities. For instance, in [3] nine machine learning techniques are used for the predictions of software vulnerabilities. Additionally, in [4] software vulnerability predictions are made using a machine learning algorithm trained on data from the NVD. However, the use of a recommender system, particularly Item-Based k-NN Collaborative Filtering, for predicting vulnerabilities has not yet been explored. This represents an opportunity for improvement which is investigated in this research. Specifically, this research explores the use of a recommender system in making predictions of vulnerabilities. However, a recommender system may encounter the cold-start problem where it is unable to suggest items to the user due to unavailable adequate information about them [5]. Active learning addresses this problem by actively querying a user to provide information about (new) items [6]. Hence, this research will also apply active learning to prevent the cold-start problem.

1.3 Research aims

Based on the problem represented in section 1.2 the main research question of this thesis is Formulated as:

How does active learning affect the performance of recommender systems to identify vulnerabilities in software products?

To answer this question, the following sub-questions need to be answered:

1. **Data collection and preprocessing**

   (a) *How to frame the prediction of vulnerabilities in software products as a problem suitable for a recommender system?*

   (b) How to collect and pre-process data for recommender systems with active learning?

2. **Recommender systems**

   (a) *What types of recommender systems are suitable for predicting vulnerabilities in software products?*

   (b) How can a recommender system be implemented for predicting vulnerabilities in software products?

3. **Active learning**

(a) *What types of strategies in active learning are suitable for implementing active learning in recommender systems to predict vulnerabilities in software products?*

(b) *How can active learning be implemented in a recommender system for predicting vulnerabilities in software products?*

4. **Evaluation**

   (a) *How to measure the performance of the recommender systems and active learning?*

   (b) *How to compare the performances of the recommender system before and after active learning?*

1.4 Thesis outline

To determine if active learning improves the performance of recommender systems for predicting vulnerabilities in software products, a contextualization will precede the investigation. This contains background information and related work. The methodology will be summarizing how the data is collected, pre-processed, and how the algorithms are implemented. Additionally, more details about the data can be found in the methodology. The subsequent section will contain all the results, which will be discussed in detail. Finally, the conclusion and discussion with implications of this research will be presented.

2 Background & related work

To enhance the understanding of the topic regarding the impact of active learning on the performance of recommender systems in predicting vulnerabilities, background information about the data, recommender systems, active learning, and evaluation methods is collected. The general concept of recommeder systems and active learning is explained, and multiple implementation methods are presented. Lastly, section 2.5 will on elaborate previous similar research. More information about vulnerabilities, penetration testing, and security can be found in the appendices.

2.1 Data

The data used in this research, is retrieved from the NVD which contains CPEs and CVEs. These concepts are described in the following sections.

2.1.1 NVD

The National Vulnerability Database (NVD) [1] provides an overview of information about vulnerabilities, and methods for technical assessment. The National Institute of Standards and Technology (NIST) maintains this database in the United States. The data is gathered from sources like software vendors, security researchers, or other organizations that detect vulnerabilities.

2.1.2 CPE

The NVD uses Common Platform Enumeration (CPE) as a standardized method for describing hardware, operating systems, and software components of a computer. The current version of CPE is 2.3 which is also used in the NVD. The format of a CPE entry is constructed like in listing 1 [7] [8].

Listing 1: CPE

```
cpe: 2.3 : part : vendor : product : version : update : edition : language :
    sw_edition: target_sw : target_hw : other
```

Breaking down Listing 1 into the variables to describe them:

1. **part**
   Can have three possible values: 'a', 'h' or 'o' which respectively represent 'Applications', 'Hardware', and 'Operating Systems'. It identifies the part of the product this vulnerability refers to.
2. **vendor**
   Supplier of the product in which this vulnerability is present.
3. **product**
   The product name in which this vulnerability is present.
4. **version**
   The version number of the product in which this vulnerability is present.
5. **update**
   The update or service pack information in which this vulnerability is present.

6. **edition**
   Further details describing the build of the product beyond the update.
7. **language**
   The language or localization of the product.
8. **sw_edition**
   Further details describing the build of the software.
9. **target_sw**
   Describing the target software environment that is compatible with this product.
10. **target_hw**
    Describing the target hardware environment that is compatible with this product.
11. **other**
    Any other information which was not included in preceding CPE details.

2.1.3 CVE and CWE

Another database containing vulnerabilities is the Common Vulnerabilities and Exposures (CVE) database maintained by MITRE [9]. The vulnerabilities in this database have an "identification number", description, public reference for publicly known cybersecurity vulnerabilities, vulnerability patch information, exploits information, CPE, Common Weakness Enumeration (CWE), and Common Vulnerability Scoring System (CVSS). The CWE is a resource that contains descriptions of software weaknesses. It serves as an evaluation tool for software vulnerability assessment [10]. And CVSS is a specification for measuring the severity of software vulnerabilities [11]. The NVD is fully synchronized with this MITRE CVE list, so the NVD contains all CVEs and provides additional information about them.

2.1.4 Bias in data

As previously mentioned, the NVD relies on the reporting of vulnerabilities from multiple sources like software vendors, security officers, researchers and users. The dependency on these sources could cause a potential bias. Examples of potential biases are described below.

1. **Report bias**
   When some vulnerabilities are more likely to be reported than others, this could create a bias in the NVD data.
2. **Location bias**
   The NVD is mainly focused on vulnerabilities affecting systems in the US which could cause a location bias.
3. **Severity bias**
   The NVD prioritizes the reporting of severe vulnerabilities. Table 1 represents the amount of CVEs in 2022 with various severity classes. Almost 40% of the reported CVEs have the severity class MEDIUM and 2% severity class LOW.

MEDIUM	9120
HIGH	8985
CRITICAL	3780
LOW	439

Table 1: Counts of severity levels in NVD data 2022

2.2 Recommender systems

Recommender systems are a type of information filtering technology that aims to predict the preferences or interests of users and recommend items that they may like. This technology has become increasingly important in recent years, especially in the context of online platforms and services that offer a variety of choices.

Formulating a recommender system problem involves defining the variables for users u, and items i. The variable r_{ui} is introduced to indicate the observations or ratings. For explicit datasets, where users provided direct feedback or ratings for items, r_{ui} indicates ratings that represent the preference of the user. For implicit datasets, where user preferences are not explicitly stated, r_{ui} indicates observations for user actions [12]. The goal of recommender systems is to predict missing or future ratings. This objective could be formulated like Formula 1. $f(u, i)$ represents a general function used to predict r_{ui} which differs among various recommender systems.

$$\hat{R}_{ui} = f(u, i) \tag{1}$$

Recommender systems could be built according to three different approaches: content-based, knowledge-based, and collaborative filtering. A content-based approach relies on the analysis of the attribute or features of items to make recommendations. It will look at the user's past behavior and recommend items that are similar in terms of their attributes or features. On the other hand, the knowledge-based approach relies on a knowledge base or expert system to make recommendations. This knowledge base contains information about the user's preferences as well as information about themselves and is used to reason about the user's preferences [13]. Collaborative filtering is described in section 2.2.1.

2.2.1 Collaborative filtering

Collaborative filtering is a type of recommender system that looks at patterns of item ratings or interactions across a group of users to make recommendations. This approach does not rely on explicit knowledge about the users or items but it uses other similar users or items to generate recommendations. It is based on the assumption that people who have similar preferences in the past will have similar preferences in the future [14].

Some techniques for collaborative filtering are memory-based methods, model-based methods, and deep learning-based methods [15]. Memory-based methods make recommendations based on similarity between users or items. This similarity is computed on the entire user-item interaction matrix using measures like cosine similarity or Pearson correlation. Memory-based methods make distinction between user-based collaborative filtering (UBCF) and item-based collaborative filtering (IBCF). UBCF recommends items that similar users have liked or interacted with. Conversely, IBCF recommends items that are similar to the items the user has liked or interacted with in the past [13]. These two types of collaborative filtering could be used together which is called Hybrid Collaborative Filtering. The most common way to combine UBCF and IBCF is to use a weighted combination of the recommendations from both techniques [14].

Model-based methods use machine learning algorithms to train a model on the user-item interaction matrix and additional information about the users and items. This model is used to predict ratings or preferences. Techniques such as Matrix Factorization (MF), Non-negative Matrix Factorization (NMF), and Probabilistic Matrix Factorization (PMF) are examples of model-based methods.

Deep learning-based methods learn the patterns in the user-item interaction matrix with neural networks by understanding the users and items beforehand. Deep Neural Networks (DNNs), and Convolution Neural Networks (CNNs) are examples of neural networks that could be used in deep learning-based methods [16].

1. **Memory-based algorithm**

   The general purpose of memory-based algorithms is to predict the votes of a particular user. To do so, they operate over the entire user or item database to make predictions [17]. Assume we have a user i, and an item j. To calculate the predicted rating of the active user on item j: $p_{a,j}$ we can use the following Formula:

   $$\bar{v_i} = \frac{1}{|I_i|} \sum_{j \in I_i} v_{i,j} \tag{2}$$

   Formula 2 represents the mean vote for user i. Where $|I_i|$ is the set of items on which user i have voted. And, $v_{i,j}$ is the vote of user i on item j. The mean vote for the user can now be used to predict the vote for item j of active user a: p_a, j:

   $$p_{a,j} = \bar{v_a} + \kappa \sum_{i=1}^{n} w(a,i)(v_{i,j} - \bar{v_i})(3)$$

   In Formula 3 n is the number of users in the database with nonzero weights. A normalized factor, denoted as κ ensures that the absolute values of the weights sum to unity. Two formulations for the weight calculation $w(a,i)$ are listed below.

   (a) **correlation**

   $$w(a,i) = \frac{\sum_j (v_{a,j} - \overline{v_a})(v_{i,j} - \overline{v_i})}{\sqrt{\sum_j (v_{a,j} - \overline{v_a})^2} \sqrt{\sum_j (v_{i,j} - \overline{v_i})^2}} \tag{4}$$

   Formula 4 represents the correlation between active user a and i. The j represents the items for which user a and i both have voted.

(b) **Vector similarity**

$$w(a,i) = \sum_j \frac{v_{a,j}}{\sqrt{\sum_{k \in I_a} v_{a,k}^2}} \frac{v_{i,j}}{\sum_{k \in I_a} v_{i,k}^2} \tag{5}$$

In equation 5 the $v_{a,k}^2$ and $v_{i,k}^2$ serve to normalize votes so that users who vote on a large number of items are not considered more similar to other users. This is achieved by using a normalization technique that takes into account the number of items that a user has voted on.

1.1 User- and item-based collaborative filtering

User- and item-based collaborative filtering are memory-based methods that both consist of three steps: similarity computation, neighborhood selection, and rating prediction. Correlation-based similarity and vector cosine-based similarity are two popular similarity computation methods which are elaborated below. Two popular neighborhood selection methods are: Max number (topk) and correlation threshold. These techniques either select the top K users and items with the highest similarity scores or retain the users and items whose similarity values exceed a threshold. Finally, the weighted sum method for the ratings prediction is described in this section [18] [19].

i. **Correlation-Based Similarity**

The Pearson correlation between users u and v is presented in Formula 6. The Pearson correlation between items i and j is presented in Formula 7

$$w_{u,v} = \frac{\sum_{i \in I}(r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in I}(r_{u,i} - \bar{r}_u)^2}\sqrt{\sum_{i \in I}(r_{v,i} - \bar{r}_v)^2}} \tag{6}$$

$$w_{i,j} = \frac{\sum_{u \in U}(r_{u,i} - \bar{r}_i)(r_{u,j} - \bar{r}_j)}{\sqrt{\sum_{u \in U}(r_{u,i} - \bar{r}_i)^2}\sqrt{\sum_{u \in U}(r_{u,j} - \bar{r}_j)^2}} \tag{7}$$

Formulas 6 and 7 are the user- and item-based implementations for Formula 4. In Formulas 6 and 7, R represents rating whereas v represents vote in Formula 4. However, these variables are equivalent. In addition, $\sum_j$ in Formula 4 corresponds with $\sum_{u \in U}$ and $\sum_{i \in I}$ in Formulas 6 and 7. In both formulas it represents the items that both users u and v have rated or the users that have both rated items i and j.

ii. **Vector Cosine-Based Similarity**

The Vector Cosine-Based Similarity treats a user as a vector of ratings rated by himself. An item is treated as vector of ratings rated by the set of users. A Vector Cosine-Based Similarity close to 1 indicates a strong correlation between each other [19]. Formulas 8 and 9 represent the Vector Cosine measure for users and items respectively. These formulas are derived from Formula 5. In these formulas v and r both represent the ratings.

$$\cos(u,v) = \frac{\sum_{i \in I_{uv}} r_{ui} r_{vi}}{\sqrt{\sum_{u \in I_u} r_{ui}^2}\sqrt{\sum_{u \in I_v} r_{vi}^2}} \tag{8}$$

$$\cos(i,j) = \frac{\sum_{u \in U_{ij}} r_{ui} r_{uj}}{\sqrt{\sum_{u \in U_i} r_{ui}^2} \sqrt{\sum_{u \in U_j} r_{uj}^2}} \tag{9}$$

iii. **Weighted sum method**
The weighted sum method predicts the rating of an active user based on the set of K neighbors that are most similar to user u which is formulated as N_u^i. The number of neighbors, K, can be adjusted. Formulas 10 and 11 represent the weighted sum method. In Formula 10, Sim_{uv} is the similarity value between user u and v. This similarity value could be computed with formulas 6 till 9. In Formula 11, $\sum_{j \in N_i^u}$ is the set of neighbors (top K items) that are most similar to item i and have been rated by the user u. Sim_{ij} is the similarity value between items i and j which could also be computed with formulas 6 till 9.

$$\tilde{r}_{ui} = \frac{\sum_{v \in N_u^i} \text{Sim}_{uv} \times r_{vi}}{\sum_{v \in N_u^i} |\text{Sim}_{uv}|} \tag{10}$$

$$\tilde{r}_{ui} = \frac{\sum_{j \in N_i^u} \text{Sim}_{ij} \times r_{uj}}{\sum_{j \in N_i^u} |\text{Sim}_{ij}|} \tag{11}$$

2. **Model-based algorithm**

The collaborative filtering task can be viewed as retrieving the expected value of a vote, given what we know about a user. In other words, calculating the probability that the active user will have a certain vote value for item j given the historical observed votes: $p_{a,j}$ and a trained model. This is captured in Formula 12.

$$p_{a,j} = \mathbb{E}(v_{a,j}) = \sum_{i=0}^{m} \text{Pr}(v_{a,j} = i | v_{a,k}, k \in I_a) i \tag{12}$$

Formula 12 includes I_a which represents the set of items on which user a has voted.

Two alternative probabilistic models for the model-based methods are cluster models and Bayesian networks which are elaborated below.

(a) **Cluster Model**
For Cluster models, with a Bayesian classifier, the probability of votes is assumed to be conditionally independent given membership in an unobserved class variable C. Users or items can be grouped into certain classes that capture commonalities in preferences or features. The model assumes that the features are conditionally independent given the class, so the joint probability of the feature vector and the class can be decomposed as a product of the individual conditional probabilities. This will be represented with the standard naive Bayes Formulation in Formula 13. The joint probability, $Pr(C = c, v_1, \dots, v_n)$, represents the user that belongs to class C and provides a set of votes denoted by $v_1, \dots, v_n$.

$$Pr(C = c, v_1, \ldots, v_n) = Pr(C = c) \prod_{i=1}^{n} Pr(v_i | C = c) \tag{13}$$

In Formula 13, $Pr(C = c)$ is the probability of a membership for class c, and $Pr(v_i|C = c)$ is the conditional probability of feature v_i given class c. The model then uses Bayes' theorem to compute the posterior probability of the class given the feature vector [17].

(b) **Bayesian Network Model**
The Bayesian Network Model is a probabilistic graphical model that represents variables and their conditional dependencies using a directed acyclic graph (DAG). It has each item in the domain represented as a node with possible vote values as states, including a state for "no vote". The training data is used to learn the Bayesian Network. The resulting model is a network with parent items that are the best predictors of its votes. A decision tree encoding the conditional probabilities for that node is used to represent each conditional probability table.

2.1 Matrix Factorization

Matrix Factorization is a model-based algorithm and one of the most popular techniques for collaborative filtering. Matrix Factorization uses a user-item matrix R, where each entry $R_{i,j}$ represents the rating given by user i to item j. The goal is to predict the missing ratings and make personalized recommendations to users based on their past interactions with items [20].

Matrix Factorization characterizes the items with vector q_i and the users with the vector p_u [21]. These vectors are inferred from item rating patterns. The elements in q_i represent the extent to which an item holds those characteristics and the elements in p_u represent the extent of interest a user has in items that correspond to those characteristics [22]. Funk [23] proposed the SVD algorithm where the rating matrix is denoted by $R \in \mathbb{R}^{m\times n}$ and it consists of m users, and n items. The regularized SVD algorithm decomposes the rating matrix R into the products of two lower rank matrices $U \in \mathbb{R}^{k\times m}$ and $V \in \mathbb{R}^{k\times n}$ as the feature matrices of users and items respectively where k represents the number of latent factors or dimensions. Formula 14 represents this decomposition. $\widetilde{R}$ illustrates the matrix decomposed by U^T and V.

$$\widetilde{R} = U^T V \tag{14}$$

Formula 15 represents the predicted rating of item i by user u based on the SVD algorithm. q_i represents the embedding vector for item i and p_u for user u.

$$\hat{r}_{ui} = q_i^T p_u \tag{15}$$

Formula 16 represents the prediction of item i by user u based on the biased version of the SVD algorithm

$$\hat{r}_{ui} = \mu + b_u + b_i + p_u^T q_i \tag{16}$$

In Formula 16, μ represents the global mean, b_u represents the bias for user u, and b_i represents the bias for item i.

The main objective of Matrix Factorization is to learn the embedding vectors q_i and p_u. These are learned by solving the formula for minimizing the regularized squared error on the known ratings which is represented by formula 17. Formula 18 represents the minimization of the regularized squared error with bias.

$$\min_{q*,p*} \sum_{(u,i)\in\kappa} (r_{ui} - q_i^T p_u)^2 + \lambda \left(\|q_i\|^2 + \|p_u\|^2 \right) \tag{17}$$

$$\min_{u,i\in K} \sum_{(u,i)\in\kappa} \big(r_{ui} - \mu - b_u - b_i - p_u^T q_i)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2 + b_u^2 + b_i^2\big) \tag{18}$$

In formulas 17 and 18, κ is the set of all the user-item pairs for which r_{ui} is known in the training set. And λ is the constant that controls the regularization to avoid overfitting. Additionally, r_{ui} represent the actual rating and $q_i^T p_u$ the predicted rating: $\hat{r}_{ui}$.

Approaches to minimize equations 17 and 18 are stochastic gradient descent, and alternating least squared (ALS) [24].

Stochastic gradient descent

The Stochastic Gradient Descent Algorithm is a general iterative optimization algorithm and loops through all ratings in the training set where it predicts r_{ui} for every training case. It starts at a random point on a function and travels down its slope in steps until it reaches the minimum point of that function.

It computes the related prediction error according to Formula 19.

$$e_{ui} =^{def} r_{ui} - q_i^T p_u \tag{19}$$

Afterwards q_i and p_u are modified like in formulas 20 and 21.

$$q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda \cdot q_i) \tag{20}$$

$$p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda \cdot p_u) \tag{21}$$

The γ in formulas 20 and 21 refers to the learning rate and proportional to this rate q_i and p_u are modified in opposite direction of the gradient. For the biased SVD algorithm, the bias terms are updated in line with update rules 22 and 23 and they are often initialized to 0.

$$b_i \leftarrow b_i + \gamma(e_{ui} - \lambda b_i), \tag{22}$$

$$b_u \leftarrow b_u + \gamma(e_{ui} - \lambda b_u). \tag{23}$$

Alternating Least Squares

Generally, ALS works by iteratively alternating between fixing one of the matrices q_i or p_u and solving the other using least squares till convergence. Convergence is attained when the difference between the actual ratings and the predicted ratings is minimized. It first holds the user matrix fixed and runs an optimizing algorithm (like gradient descent) with the item matrix. Then it holds the item matrix fixed and runs the optimizing algorithm with the user matrix.

2.2 Probabilistic Matrix Factorization

Probabilistic Matrix Factorization is also a model-based collaborative filtering technique. The difference with normal Matrix Factorization is the assumption about the user-item interaction matrix. This assumption implies that the user-item interaction matrix follows a Gaussian distribution and the latent factors are modeled as Gaussian distributions with unknown mean and variance [25]. This conditional distribution is captured in Formula 24.

$$p(R \mid U, V, \sigma^2) \ = \prod_{i=1}^{N}\prod_{j=1}^{M}[\mathcal{N}(r_{ij} \mid \mathbf{u}i^T\mathbf{v}j, \sigma^2)]^{\mathbb{I}_{ij}} \tag{24}$$

Here R is the rating matrix with entries r_{ij} for user i and item j. U and V are the latent factor matrices. σ^2 is the variance of the noise term. $\mathcal{N}(x \mid \boldsymbol{\mu}, \sigma^2)$ is the probability density function of the Gaussian distribution with mean $\boldsymbol{\mu}$ and variance σ^2.

The objective of probabilistic Matrix Factorization is to minimize the negative log-likelihood of the observed data which Formula 25 represents:

$$E = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}(r_{ij} - \mathbf{u}_i^T\mathbf{v}j)^2 + \frac{\lambda_U}{2}\sum_{i=1}^{N}||\mathbf{U}i||_{Fro}^2 + \frac{\lambda_V}{2}\sum_{j=1}^{M}||\mathbf{V}_j||_{Fro}^2 \tag{25}$$

In Formula 25, $\lambda_U = \sigma^2/\sigma_U^2$ and $\lambda_v = \sigma^2/\sigma_V^2$. Additionally, $|| \cdot ||_{Fro}^2$ indicates the Frobenius norm. The local minimum of this function can be found with stochastic gradient descent in U and V like in Formulas 19, 20, and 21.

2.3 Active learning

Recommender systems could face the cold start problem. This problem arises when the system has not yet acquired enough ratings to generate reliable recommendations. When a new user or item is introduced in the recommender system the cold start problem typically arises. Additionally, the cold start problem is often introduced with recommder systems trained on very sparse datasets. Active learning tackles this problem by focusing on obtaining more information that better represents the user's preferences.

The general concept of active learning is improving the performance of a machine learning algorithm with fewer labeled training instances by allowing it to choose the data from which it learns. The active learning algorithm can be illustrated by the following generic schema: Given a training set of N input-output pairs $(x_1, y_1), (x_2, y_2), ...(x_N, y_N)$ where $x_i \in X$ is an instance and $y_i \in Y$ is a label, we assume there is a model M that maps the input to an output $M : X \to Y$. A function $Loss(M)$ is defined that measures the error of the model. For every iteration j in the active learning process, the learner selects a query $q_i \in potentialQueries \subset X$, then requests the labels of them, and add these labels to the model. The new model including these new instances (q_i, y_i) is presented as M' [26].

Active learning can be implemented using online learning, where new data is available for the model while learning. On the other hand, active learning can be implemented using offline learning. With offline learning the model is implemented on a pre-collected dataset and does not consider new data instances while operating [27]. For both implementations of active learning, a learner will seek assistance from an oracle (e.g. human annotator) by asking queries in the form of unlabeled data that require labeling by this oracle. With this approach the learner gains a deeper understanding of the underlying patterns to make more accurate predictions. An example of active learning is represented in figure 2. Graph 'a' in figure 2 represents a dataset of 400 instances, drawn from two Gaussian's classes. Graph 'b' represents a logistic regression model trained with 30 labeled data points randomly picked from the problem domain in graph 'a'. The line in this graph represents the decision boundary of the classifier. This model has an accuracy of 0.7. Graph 'c' represents a logistic regression model trained with 30 actively queried data points using uncertainty sampling. this graph has an accuracy of 0.9 which is higher than graph of the model without active learning. More about this method is explained in section 2.3.2 [28].

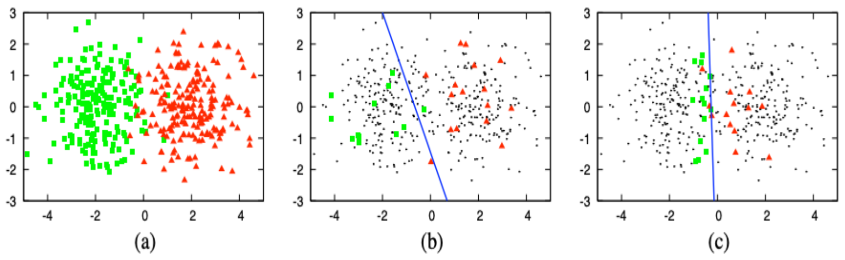
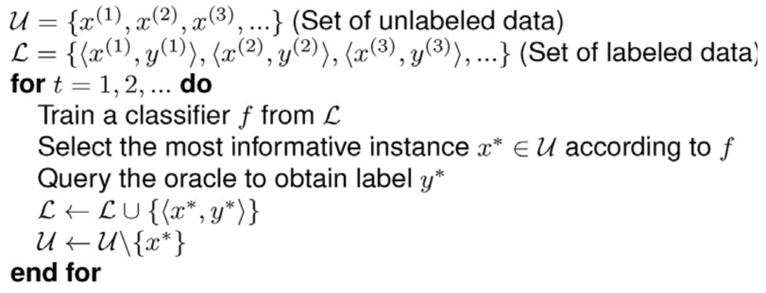


Figure 2: Example active learning

2.3.1 Algorithm

Figure 3 presents the pseudocode for a general active learning algorithm. The '$\mathcal{U}$' represents the set of unlabeled data and '$\mathcal{L}$' the set of labeled data. A classifier is trained to select instances to query. When these instances got labeled by the oracle, the set $\mathcal{L}$ is updated with the labeled instances and these instances are deleted from the set $\mathcal{U}$ [29]. This classifier will select the most informative instance to query. Other methods for selecting instances are elaborated in section 2.3.2.



```
$\mathcal{U} = \{x^{(1)}, x^{(2)}, x^{(3)}, ...\}$ (Set of unlabeled data)
$\mathcal{L} = \{\langle x^{(1)}, y^{(1)}\rangle, \langle x^{(2)}, y^{(2)}\rangle, \langle x^{(3)}, y^{(3)}\rangle, ...\}$ (Set of labeled data)
for $t = 1, 2, ...$ do
    Train a classifier $f$ from $\mathcal{L}$
    Select the most informative instance $x^* \in \mathcal{U}$ according to $f$
    Query the oracle to obtain label $y^*$
    $\mathcal{L} \leftarrow \mathcal{L} \cup \{\langle x^*, y^*\rangle\}$
    $\mathcal{U} \leftarrow \mathcal{U} \backslash \{x^*\}$
end for
```

Figure 3: Pseudocode active learning

2.3.2 Sampling strategies

Strategies for active learning in recommender systems can be classified into two main categories: personalized and non-personalized. Each of them can be further divided int single-heuristic and combined-heuristic [26] where combined-heuristic combines strategies. Some strategies are considered and elaborated.

1. **Personalized**
   (a) **Acquisition probability based**
   This strategy enhances the system's performance by increasing the likelihood that the user will recognize the queried item and will rate it. This is realized by personalizing the rating request with various algorithms.
      i. **Item-Item**
      Item-Item will select items that are most similar to the items the user already rated. The formulas in subsection 1.1 of the Collaborative Filtering section 2.2.1 present the item similarity computation used for Item-Item.
      ii. **Binary Prediction**
      Binary Prediction will transform the rating matrix into a new matrix where is indicated whether a user have rated an item. A factor model is then used to predict the item the user is most likely to rate.
      iii. **Personality-Based Binary Prediction**
      The matrix is transformed into a new matrix as with Binary Prediction. However, this matrix is now used to train an extended version of Matrix Factorization to profile users in terms of known attributes.

   (b) **Impact Based**
   With the Impact Based strategy, items will be selected that minimize the rating prediction uncertainty over all items.
      i. **Influence Based**
      This strategy calculates the effect of item ratings on the prediction of other items' ratings. It will choose the items that have the greatest influence on the others. To estimate this influence: first $\hat{r}_{ui}$ for user u and unrated item i is computed. Then

$\hat{r}'_{ui}$ is determined by $\hat{r}'_{ui} = \hat{r}_{ui} - 1$. Two models are trained where for one model the training data includes $\hat{r}_{ui}$ and the other includes $\hat{r}'_{ui}$. The absolute difference of these predictions for the ratings of all items different from i are then computed and summed up which results in the influence of i.

ii. **Impact Analysis**
Impact Analysis will select items whose ratings have the biggest impact on the prediction of other ratings.

(c) **Prediction Based**
Strategies that are prediction based use a rating prediction model to predict which item should be queried. Different models are considered.

i. **Aspect Model**
The probability of the user u giving a rating $r \in R$ to an item $i \in I$ is presented by Formula 26. In this formula every user u has a probability-based membership to multiple aspects $z \in Z$. Users that are in the same group have similar rating patterns. $p(r|u,i)$ represents how likely membership to group z for user u is. Additionally, $p(r|z,i)$ represents how likely it is for the users of group z to give rating r item i.

$$p(r|u,i) = \sum_{z \in Z} p(r|z,i)p(z|u) \tag{26}$$

ii. **Highest or lowest Predicted**
Highest Predicted strategy scores items based on their predicted rating and queries the items with the highest scores. These items are estimated to have the highest probability of being rated by the user. In contrast to, Lowest Predicted which uses the opposite heuristic of the Highest Predicted strategy.

iii. **MinNorm**
For the strategy MinNorm, Matrix Factorization is used to compute the latent factors for each item. It will select items with vectors that represent the minimum Euclidean Norm.

(d) **User Partitioning**
For User Partitioning, the users are divided into clusters based on similar taste. It will query the items that will reveal to which cluster the user belongs.

i. **IGCN**
Information Gain Clustered Neighbours (IGCN) constructs a decision tree with each leaf node representing a users' cluster. Each internal node represents tests to test the users' preferences.

2. **Non-personalized**
Non-personalized strategies are strategies that do not take personal preferences into consideration. Some strategies are elaborated.

(a) **Uncertainty-reduction**
This strategy selects items to reduce uncertainty about ratings by querying controversial or diverse ratings. Two strategies that can be involved in uncertainty-reduction are considered.

i. **Variance**
The item with the highest variance will be queried. A high variance indicates diverse ratings for this item. Variance is represented in Formula 27, where U_i represents the set of users who rated item i, r_{ui} is the rating user u gives to item i, and $\bar{r}$ is the average rating for i.

$$\text{Variance}(i) = \frac{1}{|U_i|} \sum_{u \in U_i} (r_{ui} - \bar{r}_i)^2 \tag{27}$$

ii. **Entropy**
Entropy calculates the ratings' distribution for a given item. Formula 28 captures this where $p(r_i = r)$ is the probability that a user gives rating r to item i.

$$\text{Entropy}(i) = -\sum_{r=1}^{5} p(r_i = r) \log(p(r_i = r)) \tag{28}$$

(b) **Error-reduction**
This strategy will query items that will help to reduce the error in the system, and therefore improve the prediction accuracy.

i. **Greedy Extend**
Greedy Extend searches for items whose ratings, when added to the training set and elicited by users, produce the lowest system RMSE.

ii. **Representative-based**
Representative-based will try to determine a subset of items that best exemplify the entire catalog.

(c) **Attention-based**
Attention-based strategies focus on querying the items that are most popular to the users. Users can rate such items because they are likely to be familiar to them.

i. **Popularity**
This strategy simply queries the most popular items which is based on the highest ratings among all users.

ii. **Co-coverage**
Co-coverage is represented in Formula 29. In this formula m_{ij} indicates the number of users who have rated item i and item j both.

$$\text{Co-coverage}(i) = \sum_{j=1}^{n} m_{ij} \tag{29}$$

Some other strategies for sampling in active learning are considered and elaborated below [28] [30].

1. **Uncertainty Sampling**
Uncertainty sampling, queries the instances whose posterior probability of being positive is 0.5. These are the instances that are most uncertain and close to the decision boundary.

Another technique to implement uncertainty sampling uses entropy as an uncertainty measure provided in equation 30.

$$x^*_{ENT} = \underset{x}{\operatorname{argmax}} - \sum_i P(y_i|x:\theta) log P(y_i|x:\theta) \tag{30}$$

In equation 30 y_i varies over all possible labeling options. Theta θ denotes the vector of parameters that specify the posterior distribution over models given the available training data. Entropy is the measure of uncertainty or randomness and is used in uncertainty sampling to quantify the amount of information in an instance. Additionally, instances can also be sampled according 'confidence'. The instances with the 'least confident' labeling will be queried in line with equation 31

$$x^*_{LC} = \underset{x}{\operatorname{argmin}} P(y^*|x:\theta) \tag{31}$$

$y^* = argmax_y P(y|x:\theta)$ and this represents the class labeling which is most likely. In other words, Formula 31 represents the instance that is most uncertain about having the most likely label.

2. **Query-By-Committee**
With the Query-By-Committee strategy, a 'committee', $\mathcal{C} = \theta^{(1)}, ..., \theta^{(\mathcal{C})}$, is maintained. This committee consists of models that are trained on the current labeled set $\mathcal{L}$ but represent competing hypotheses. Each member of $\mathcal{C}$ can vote on the labeling options of the query candidates. The query they most disagree about is the most informative query. To decide the amount of disagreement Formula 32 is considered.

$$x^*_{VE} = \underset{x}{\operatorname{argmax}} - \sum_i \frac{V(y_i)}{C} log \frac{V(y_i)}{C} \tag{32}$$

In Formula 32 y_i ranges over all the possible labeling options and $V(y_i)$ is the total amount of votes a label receives from the committee's members.

3. **Expected Model Change**
Expected model change is a method where the instance, that would bring the greatest change to the current model, is queried when its label is known. To determine what instance would bring the greatest change, Formula 33 is employed.

$$x^*_{EGL} = \underset{x}{\operatorname{argmax}} \sum_i P(y_i|x:\theta) \|\nabla \ell(\mathcal{L} \cup \langle x_i, y_i \rangle : \theta)\| \tag{33}$$

In Formula 33, $\|\cdot\|$ is the Euclidean norm of each resulting gradient vector which points in the direction of the steepest increase of a function. The length is calculated as an expectation over the possible labeling options because the true label y is not known. $\nabla \ell(\mathcal{L} \cup \langle x_i, y_i \rangle : \theta)$ is a gradient of the function ℓ with respect to θ that would be obtained by adding the instance $\langle x_i, y_i \rangle$ to $\mathcal{L}$

4. **Exhaustive search**
   Exhaustive search is querying each instance until one is not discarded.

5. **Multiple-instance query**
   Instead of labeling individual instances, a set of instances is labeled when the multiple-instance query strategy is applied. This set of instances is named a bag. It selects multiple bags for labeling based on their uncertainty. Once a batch of bags is selected, the clustering algorithm could be used to group the instances within each bag. The most informative in each cluster are then labeled while the remaining instances are assumed to have the same label as the cluster [31].

2.4 Evaluation methods

Evaluation methods are critical to determine how well a model is performing. Selecting the correct evaluation method is essential to measure the relevant aspects of the model. Various evaluation metrics are considered for evaluating the performances of recommender systems and active learning. Additionally, these performance scores need to be compared in order to make statements about the differences in the performances of the models. These comparisons could be performed using statistical tests.

2.4.1 Cross-validation

Cross-validation is a popular data resampling technique used to evaluate the performances of models by estimating the true prediction error. It is implemented on the input data where it generates training set(s) and testing set(s) from data instances in the input. One critical aspect that needs to be considered is that the train and test data are split and the model is only trained with the train data. Some sampling methods are considered [32].

1. **K-fold cross-validation**
   K-fold cross-validation distributes data instances over K groups that are called folds. Consider dataset D, for each fold the data instances in this fold are split into a training set D_{train} and a testing set D_{test}. A different model per fold will be built using D_{train}. The resulting model $\hat{f}(x, D_{train})$ will be tested with D_{test}. For every fold a different subset is used as a testing set with no overlap between validation sets of different folds. The average of the performance scores that these models achieved is an estimate of the performance of the final model $f(x, D)$.

2. **Single hold-out validation**
   With the single hold-out validation method some data instances from the input data are sampled to use as a validation set while the remaining data instances are used for the learning set. Generally, the test set contains 10% to 30% of the data instances. A single model is trained on D_{train} and then tested with D_{test}.

3. **K-fold random subsampling**
   K-fold random subsampling is similar to K-fold cross-validation. However, the difference is in how the data instances are distributed over the folds. For K-fold random subsampling, the data instances are randomly distributed over the folds where any two training sets or two testing sets, may overlap. However, as with K-fold cross-validation: any pair of the training and testing set is disjoint $D_{\text{train},j} \cap D_{\text{test},j} = \emptyset$.

4. **Leave-one-out cross-validation**
   Leave-one-out cross-validation is a special case of K-fold cross-validation where the testing set D_{test} consists of one single data point and the model is trained on the remaining data, D_{train}. This process is repeated for every data point in D.

2.4.2 Performance metrics

There is a wide range of performance metrics available, each intended to measure different aspects of a model's performance. In context of recommender systems, aspects such as ranking could be more important than accuracy. Elaborations on some performance metrics follow [33] [34] [35]. Table 2 contains the description of generally used terms in evaluation metrics.

Term	**Description**
True negative	Predicted negative, actual negative
False negative	Predicted negative, actual positive
True positive	Predicted positive, actual positive
False positive	Predicted positive, actual negative

Table 2: Terms and descriptions of concepts in evaluation

1. **MAE, MSE, and RMSE**
   The formulas for Mean Absolute Error (MAE), Mean Square Error (MSE), and Rooted Mean Square Error (RMSE) are presented in equations 34, 35, and 36. The MAE, MSE, and RMSE capture the accuracy of a model. They differ in quantifying the difference between the actual and the predicted rating. MAE evaluates the absolute differences. In contrast, MSE squares the differences which make all the results positive. RMSE, on the other hand, takes the square root of the differences for a more interpretable and directly comparable result by aligning its scale with the original data. In formulas 34, 35, 36, Q represents the test set of the model, r_{ui} the actual rating of item i by user u, and $\hat{r}_{ui}$ the predicted rating of item i by user u.

   (a) **MAE**

$$MAE = \frac{1}{|Q|} \sum_{(u,i)\in Q} |r_{ui} - \hat{r}_{ui}| \tag{34}$$

   (b) **Mean Square Error**

$$MSE = \frac{1}{|Q|} \sum_{(u,i)\in Q} (r_{ui} - \hat{r}_{ui})^2 \tag{35}$$

   (c) **Root Mean Square Error**

$$RMSE = \sqrt{\frac{1}{|Q|} \sum_{(u,i) \in Q} (r_{ui} - \hat{r}_{ui})^2} \tag{36}$$

2. **Precision, Recall, F-measure**

   (a) **Precision**
   Precision measures the accuracy of positive predictions. Positive predictions are data instances classified to be part of a specific class. It is calculated as the ratio of true positive predictions to the total of positive prediction (the incorrect and the correct ones). This is captured in Formula 37. Table 2 contains the explanation of the terms.

   $$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{37}$$

   (b) **Recall**
   Recall measures the ability of the model to identify all instances of the positive class. It is calculated as the ratio of true positive predictions to the total number of actual positive instances. False negatives are positive instances classified as negative by the model.

   $$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{38}$$

   (c) **F-measure**
   The F-measure combines Precision and Recall into a single evaluation metrics. It incorporates the aspects of both Precision and Recall.

   $$\text{F-Measure} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{39}$$

3. **ROC Curve**
   The Receiver Operating Characteristic (ROC) Curve is represented as a two-dimensional graph with the false positive rates (FPR) on the X-axis and the true positive rate (TPR) on the Y-axis. It visualizes the trade-off between FPR and TPR.

4. **Mean Average Precision (MAP) and MAP@K**
   MAP is a performance metric that considers the ranking of items. It captures the accuracy and order of the items which is shown in formulas 40 and 41. In formulas 40 and 41, Q is the number of recommendations, k is the rank in a list of recommendations, $rel(k)$ is the relative function given rank k which determines the relevance of the item at rank k, and $p(k)$ represents the precision given rank k. MAP@K focuses on a subset (of K) of the recommendations in stead of considering all recommendations.

$$MAP = \frac{1}{Q}\sum_{q=1}^{Q} AveP(q) \tag{40}$$

$$AveP(q) = \frac{\sum_{k=1}^{n} p(k) \times rel(k)}{n} \tag{41}$$

5. **Mean Reciprocal Rank**
   The Reciprocal Rank evaluates whether the model places the most relevant items in front. This is captured by Formula 42 where Q is the number of recommendations and $rank_i$ is the rank of the first relevant item in the ranked list of recommendations. So, the Reciprocal rank is calculated as the inverse of the rank of the highest-ranked relevant item.

$$MRR = \sum_{q=1}^{Q} \frac{\frac{1}{rank_i}}{Q} \tag{42}$$

6. **Normalized Discounted Cumulative Gain (nDCG)**
   nDCG is an evaluation metric that takes into account the relevance of the item and its position in the ranked recommendation list. It measures how well the model performs in presenting relevant items at the top of the ranked list. Formula 44 represents the DCG which gives weights to relevant items that are at the top of the ranked list. In this formula $disc(r(i))$ is a discount function based on the ranking, making the rating of the preceding items more important and $u(i)$ is the relevance of the items in the list, a higher value for $u(i)$ corresponds with more relevance. In Formula 43, $DCG(r_{\text{perfect}})$ represents the outcome of Formula 44 for a perfectly ranked list [36].

$$nDCG = \frac{DCG(r)}{DCG(r_{\text{perfect}})} \tag{43}$$

$$DCG(r) = \sum_{i=1}^{n} disc(r(i))u(i) \tag{44}$$

7. **Hit**
   Hit-rate is presented in Formula 45. The hit-rate refers to the number of items in the test set that also appear in the list with predicted items. In this formula n represents the amount of users. A Hit-rate of 1.0 indicates that the model always recommends the correct items.

$$\text{HR} = \frac{\text{Number of Hits}}{n} \tag{45}$$

8. **Coverage**
   Coverage refers to the proportion of items recommended, relative to the total items as Formula 46 presents. In Formula 46 $I(u)$ is the number of items recommended for user u and I represent the total number of items.

$$\text{Coverage} = \frac{\bigcup_{u \in U} I(u)}{I} \tag{46}$$

9. **Personalization**
   Personalization computes recommendation similarity across users. A high personalization score indicates a good personalization where the lists of recommendations differ among users. Cosine similarity can be used to determine the similarity between the lists of recommendations for users which is captured in Formula 47. In this formula, x and x' are the lists with recommendations being compared for two users where d represents the dimensions (features) of the input space. Personalization score is the dissimilarity (1-cosine similarity) between x and x' [37] [38].

$$\cos(\theta_h) = \frac{\sum_{i=1}^{d} x_i \times x'_i}{\sqrt{\sum_{i=1}^{d} x_i^2} \times \sqrt{\sum_{i=1}^{d} x'^2_i}} \tag{47}$$

10. **Novelty**
    The novelty of a model is its ability to suggest items to users that are unfamiliar. Formula 48 captures the general idea of novelty where θ is a contextual variable and represents any element that influences item discovery. Novelty is defined as the difference between an item and "items that have already been observed" in some context. Additionally, $p(seen|i, \theta)$ reflects a factor of item popularity where high novelty scores correspond to items that few users have interacted with [39].

$$\text{nov}(i|\theta) = 1 - p(seen|i, \theta) \tag{48}$$

2.4.3 Statistical hypothesis testing

Statistical hypothesis testing is a method to verify the truth of two hypotheses: the null hypothesis and the alternative hypothesis. The null hypothesis states "there is no difference between the variables", and the alternative hypothesis contradicts the null hypothesis where it states "there is a significant difference between the variables". This method can be implemented using following approaches.

2.4.3.1 Paired t-test

The paired t-test, or dependent sample t-test, compares the means of two sets of observations taken from the same object. By doing this test, it is assumed that the population of difference scores is normally distributed [40]. Normality in the distribution of the population can be tested with a Shapiro-Wilk test.

Conducting a paired t-test for testing hypotheses about the population mean difference μ_d is done according to Formula 49 [41]. In this formula $\bar{d}$ is the sample mean difference, μ_{d0} is the hypothesized population mean difference assumed for the null hypothesis, s_d is the standard deviation of the sample differences, and n is sample size.

Null hypothesis for paired t-testing is represented in Formula 50, it indicates no significant differences between the means of the paired populations.

After determining the critical value from the t-distribution [42] with the chosen significance level and degrees of freedom (which is sample size -1) the critical value is compared against the absolute outcome of the t-statistic in Formula 49. The null hypothesis is rejected when $|t| \geq$ critical value.

$$t = \frac{\bar{d} - \mu_{d0}}{\frac{s_d}{\sqrt{n}}} \tag{49}$$

$$H_0 : \mu_d = 0 \tag{50}$$

However, another key assumption is that data instances in each sample are independent. When using cross-validation as described in section 2.4.1, this assumption is violated [43]. As part of the K-fold cross-validation, a given data instance will be used in the training data $k - 1$ times. This results in dependent performance scores because the models are trained on the (partially) same data. Advised is to use a nonparametric test with fewer assumptions like the Wilcoxon signed-rank test which is elaborated in section 2.4.3.2 [44].

Shapiro-Wilk test

A Shapiro-Wilk test can help with determining whether a population is normally distributed [45] [46]. It is based on the statistic in Formula 51 where $X_{(1)} \leq X_{(2)} \leq \ldots \leq X_{(n)}$ represent the ordered values of a sample $X_1, X_2, \ldots, X_n$, and a_i denotes the predefined constants. The null hypothesis is represented in Formula 52. This hypothesis states that distribution X follows a normal distribution with mean μ and variance σ^2. Using Formula 51 for testing hypothesis 52 we got Formula 53. When $W_0 < W_0(\alpha, n)$ holds, where $W_0(\alpha, n)$ is the critical value at significance level α, the null hypothesis in Formula 52 is rejected. This indicates strong evidence against normality.

$$W = \frac{\left(\sum_{i=1}^{n} a_i X_{(i)}\right)^2}{\sum_{i=1}^{n} (X_i - \bar{X})^2} \tag{51}$$

$$H_0 : X \sim \mathcal{N}(\mu, \sigma^2) \tag{52}$$

$$W_0 = \frac{\left(\sum_{i=1}^{n} a_i X(i)\right)^2}{\sum_{i=1}^{n} (X_i - \mu_0)^2} \tag{53}$$

2.4.3.2 Wilcoxon signed-rank test

A Wilcoxon signed-rank test is known to perform well on small datasets. It is a nonparametric statistical procedure that uses the magnitudes of differences between paired observations rather than just the signs of the differences [41] [47]. This test can either be one-sided or two-sided. In a one-sided Wilcoxon signed-rank test, it tests whether the median difference in the population is greater or less than than the median difference for the other population. For a two-sided Wilcoxon signed-rank test, it tests whether or not the median differences equal zero.

The null hypothesis for a two-sided Wilcoxon signed-rank test is presented in Formula 54 it indicates that the median difference of the population is equal to m_0 which is the median difference of the other population. Its alternative hypothesis in Formula 55 indicates they are not equal.

$$H_0 : \mu = \mu_0 \tag{54}$$

$$H_A : \mu \neq \mu_0 \tag{55}$$

The null and alternative hypotheses for the one-sided Wilcoxon signed-rank test in both directions are presented in formulas 56 to 59.

$$H_0 : \mu \geq \mu_0 \tag{56}$$

$$H_A : \mu < \mu_0 \tag{57}$$

$$H_0 : \mu \leq \mu_0 \tag{58}$$

$$H_A : \mu > \mu_0 \tag{59}$$

The following calculations need to be performed to decide whether to reject the null hypothesis:

1. For each observation in the population compute d_i which is the difference score between the paired observations. A paired observation refers to a set of data instances where each observation is matched with a specific observation from another population.
2. Rank the differences scores where rank 1 is assigned to the smallest difference score.
3. Find T_+ and T_- which are the sum of the ranks with positive signs and the sum of ranks with negative signs respectively.

For the two-sided Wilcoxon signed-rank test, the null hypothesis is true when no big differences in the values of T_+ and T_- are expected. However, when the null hypothesis is rejected, either a small value of T_+ or T_- is observed.
For the one-sided Wilcoxon signed-rank test the null hypothesis in Formula 56 is rejected at significance level α when T_+ is less than or equal to the tabulated T for n and preselected α. The tabulated T can be found in the Table with critical values [48]. Finally, for the one-sided Wilcoxon signed-rank test the null hypothesis in Formula 58 is rejected at significance level α when T_- is less than or equal to the tabulated T for n and preselected α.

2.5 Related work

This section elaborates the related literature at the intersection of the topics introduced above. However, there is a gap in the literature concerning the implementation of recommender systems and active learning on cybersecurity data for predicting vulnerabilities. This gap emphasizes the absence of research where recommender systems are used to predict vulnerabilities and are improved with active learning. Existing literature mainly focuses on automatically finding vulnerabilities in source code, predicting the likelihood of a vulnerability to be exploited, or generating attack graphs.

In [49] a machine learning-based system is proposed that gathers artifacts such as social media posts, write-ups and proof-of-concepts with information related to vulnerabilities. These artifacts are utilized to predict the likelihood of exploits being developed against these vulnerabilities. In [50] M. Edkrantz also predicts exploit likelihood by employing multiple machine learning algorithms trained on data from the National Vulnerability Database (NVD) and the Exploit Database (EDB). Among these machine learning algorithms, the linear time Support Vector Machine (SVM) algorithm outperforms the other algorithms in terms of performance metrics and execution time. This work also indicates the features: common words, references, and vendors as the most important features for the machine learning algorithms.

in [3] an empirical study is conducted where 9 machine learning techniques and 3 statistical techniques are used to predict software vulnerabilities. Important to note is that these machine learning techniques do not include recommender systems. This study concludes that the machine learning techniques achieved higher performance scores in the prediction task than the statistical techniques. Among the machine learning techniques, the adaptive-neuro fuzzy inference system outperformed the others. In [4] predictions are made for software vulnerabilities using historical patterns of the vulnerabilities from the National Vulnerability Database (NVD). N-grams are extracted from these historical patterns and used to make the predictions. It was found that the sequential patterns of vulnerability events follow a first-order Markov property. Meaning, the next vulnerability can be predicted by only using the previous vulnerability with a precious of around 90%.

Recommendation systems can also be utilized to make predictions about vulnerabilities. For example, N. Polatidis et al. [51] use a parameterized version of multi-level collaborative filtering method in combination with attack graph analysis, where they examine a network and predict how an attacker could move after access is gained. Similar to this, in [52] recommender systems are used to predict attacks within a network and to present a ranked list with cyber defence actions. The recommender system functioning as an attack predictor uses the collaborative filtering algorithm and the recommender system to rank defence actions uses a knowledge-based algorithm. Additionally, in [53] a recommender system is used to rank vulnerability-exploit. This recommender system is based on the TOPSIS algorithm and trained on data from the NVD and CNVD databases.

In the paper titled 'A Recommender System for Tracking Vulnerabilities', Philip Huff, Kylie McClanahan, Thao Le, and Qinghua Li present a recommender system to automatically identify a minimal candidate set of CPEs for software names [54]. A problem for matching vulnerability reports is identified and solved with a pipeline of natural language processing (NLP), fuzzy matching, and machine learning. Fuzzy matching is used to automatically match the target CPE (the concept 'CPE' is explained in section 2.1) against an enterprise hardware and software inventory. This is done during a three phase automation process. First, word vectors of the hardware and software inventory are extracted with Word2VEc which is a natural language processing technique.

Then, based on these word vectors, the software inventory package names are matched to a set of CPEs with fuzzy matching. This matching is based on the similarity between the software package inventory and the CPE. Similarity is calculated according to Formula 60.

$$\cos\theta = \frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}||||\mathbf{B}||} \tag{60}$$

Lastly, machine learning is used to order these CPEs and produce a small set of candidate target CPEs that match the hardware or software. It will recommend the most likely CPEs first.

In [26] a survey is conducted on active learning in collaborative filtering recommender systems where they classified different active learning strategies. They also identified some issues with implementing active learning in recommeder systems: the cold start problem, little user engagement in rating items, algorithmic complexity, context awareness for active learning, and continuous active learning.

Additionally, in [55] a study is presented where active learning is implemented in recommender systems, taking into account the characteristics of the aspect models which is a model used in active learning. This research shows better performances than with the Bayesian approaches in terms of accuracy. The proposed active learning implementation with the aspect model shows to be a promising solution for improving the performance of recommender systems.

Finally, in [56] a K-nearest neighbors algorithm is implemented to recommend movies. Multiple similarity measures like cosine, msd, pearson, and pearson baseline are implemented and compared on accurcay metrics. It has been observed that the error like RMSE, MSE, and MAE are stable after the neighborhood size of 40. The conclusion is that 40 is the optimized value of K number of nearest neighbor for movie recommendations.

Related research described above are all related to cybersecurity and some implement recommender systems. However, none of them uses recommender systems in combination with active learning to predict vulnerabilities in software products. Therefore, the related research is limited and the performance of a recommender system to predict vulnerabilities is unknown. Due to the lack of related work, there is no established consensus on the optimal method for implementation of a recommender system on cybersecurity data. As a result, further research is necessary.

3 Methods

To investigate how active learning affects the performance of a recommender system by identifying vulnerabilities in software, multiple implementations of recommender systems and active learning should be considered. In section 2.2 various implementations of the recommender systems are discussed: the classes 'memory-based' and 'model-based', and the approaches 'item-based', 'user-based' or 'hybrid'. In section 2.3 multiple methods and query strategies for active learning are elaborated. To select the best fitting choice we should formulate the problem. Section 2.2 describes how a recommender system problem should be formulated. First, users u and items i should be defined. In terms of predicting vulnerabilities in software products, the vulnerabilities are defined as items i and the software products as users u. Furthermore, the rating r_{ui} is binary where 1 indicates the presence of a vulnerability and 0 its absence. Additionally, these ratings are considered implicit feedback. To implement this recommender system problem, the data must first be considered.

3.1 Data

The data used in this research is fetched from the NVD [1] with a Python script that can be found in my GitHub repository [57]. This script also manipulates the data to ensure it conforms to the required format for the recommender systems. Specifically, it subtracts the software product names from the description and adds it as a column to the data. In cases where these names could not be subtracted, they will be included as 'unknown'. The cause for the inability to subtract some names is because some vulnerabilities are named differently in the database because they were rejected since they were incorrectly assigned.

Only data from 01-01-2005 till 31-07-2023 is considered to exclude outdated data. The total amount of CVEs is 234102. From this total, 13788 CVEs have an unknown product name. Investigation of the data shows that for all CVEs with unknown product names, CVEs were incorrectly assigned and could therefore be ignored and subtracted from the data. Figure 4 shows the long tail plot of the distribution of CVEs in the top 100 Vendor-Product combinations based on the amount of CVEs.

Figure 5 represents the occurrences of unique CVEs in the dataset. This plot reveals that the majority of CVEs occurs only once. Additionally, the CVEs that occurred twice are most of the time duplicates. So, the dataset is mainly occupied with unique CVEs. When defining items i as CVEs and users u as software products, every item in the dataset would only have one user. Such sparse data could not be handled in recommender systems. Therefore, the CWE is considered which is explained in section 2.1.3. The items i are now defined as CWE instead of CVE.

Finally an interaction sequence dataset is constructed with all users, items and their ratings. Each row in this dataset contains a user ID, item ID, and rating value. The rating value is always 1 which represents a CWE being present in the software product.

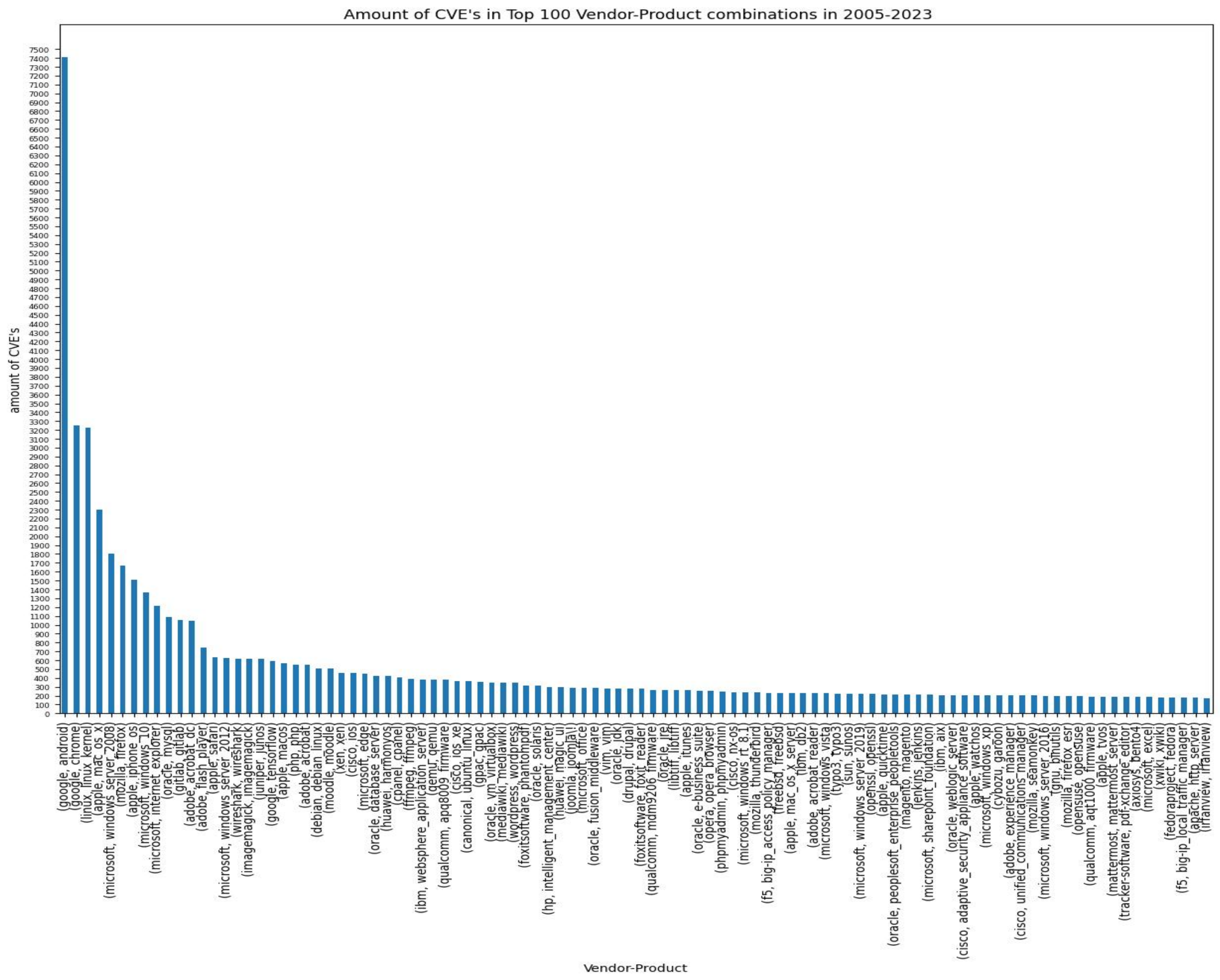


Figure 4: Long tail plot of CVEs

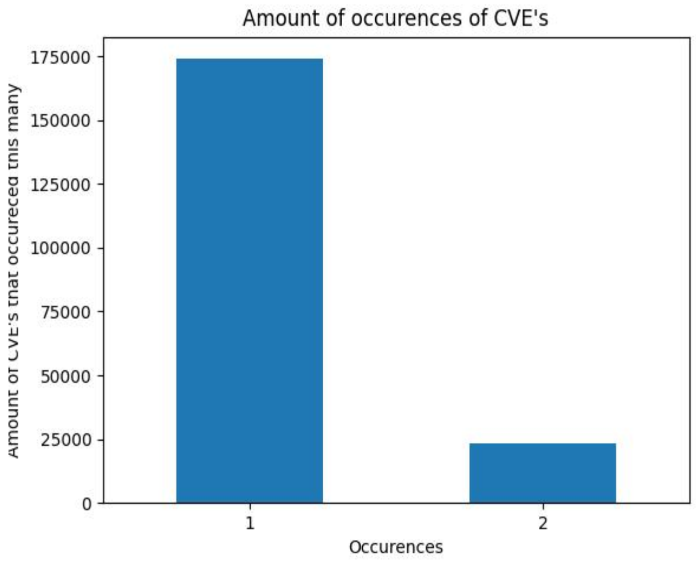


Figure 5: Amount of occurrences of CVEs in 2015-2023

Table 3 contains general information about the data where items represent CWEs and users the software products. The average number of users per item is 2, which could introduce a cold start problem especially when using K-fold cross-validation with $K > 2$. Since this could make test sets with users that are not provided in the training data. To prevent the item cold start problem, items with less than 5 users are subtracted from the dataset. However, pruning those items will result in some users having less than 5 items which could lead to user cold start problem. A loop is implemented that stops with pruning when every item and every user has more than or equal to 5 users and items respectively. A total of 42673 users and 215 items are subtracted. General information about this new dataset is presented in Table 4.

Total users	46187
Total items	394
Average number of users per item	238
Average number of items per user	2
User with most items	Android
Item with most users	CWE-79
items with less than 5 users	198
users with less than 5 items	42659

Table 3: General overview data before pruning

Total users	3514
Total items	179
Average number of users per item	186
Average number of items per user	9
User with most items	Android
Item with most users	CWE-70

Table 4: General overview data after pruning

Figures 6 and 7 represent the number of users per item, and number of items per user respectively. Both plots are based on the pruned data.

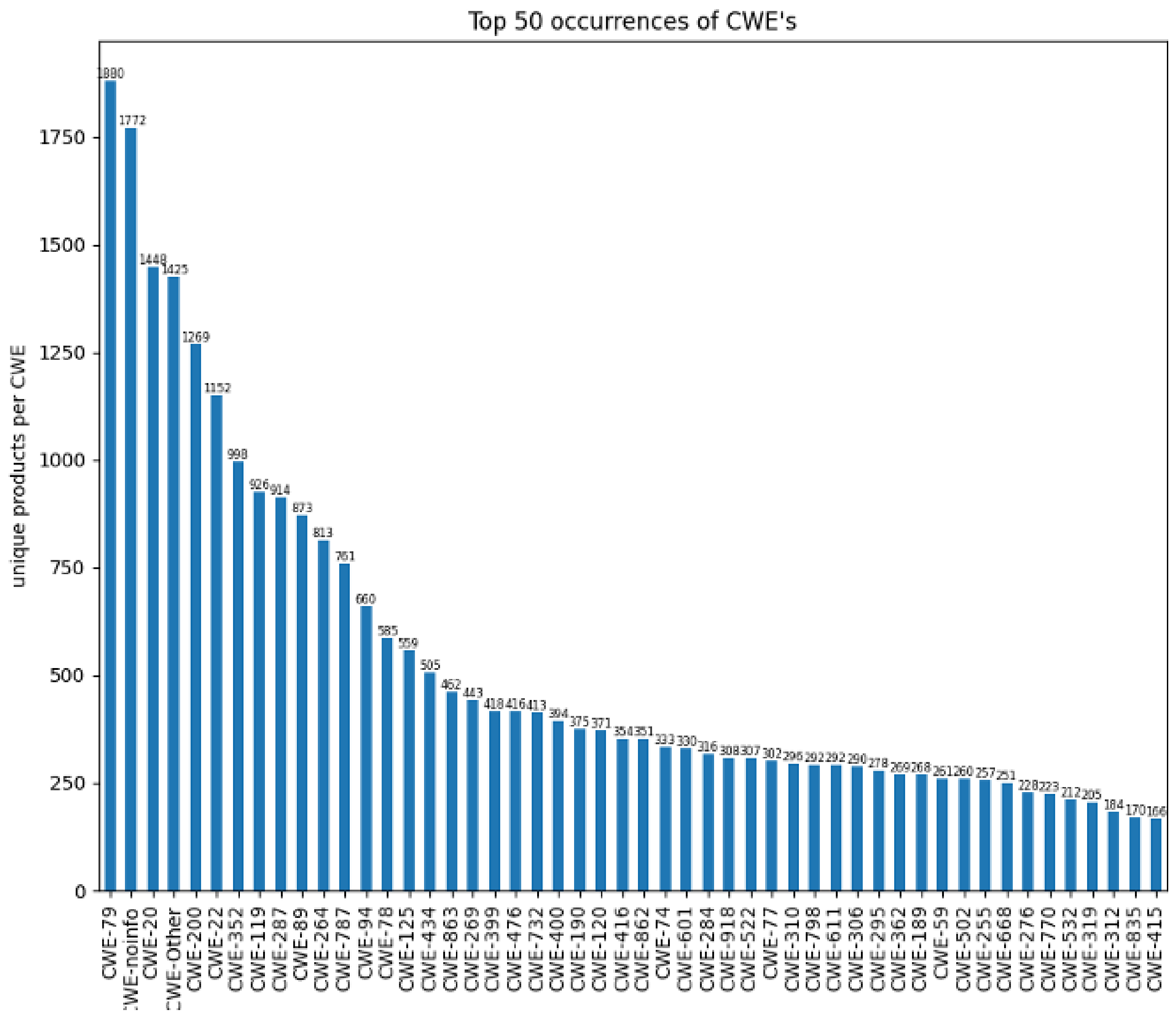


Figure 6: Number of users per item

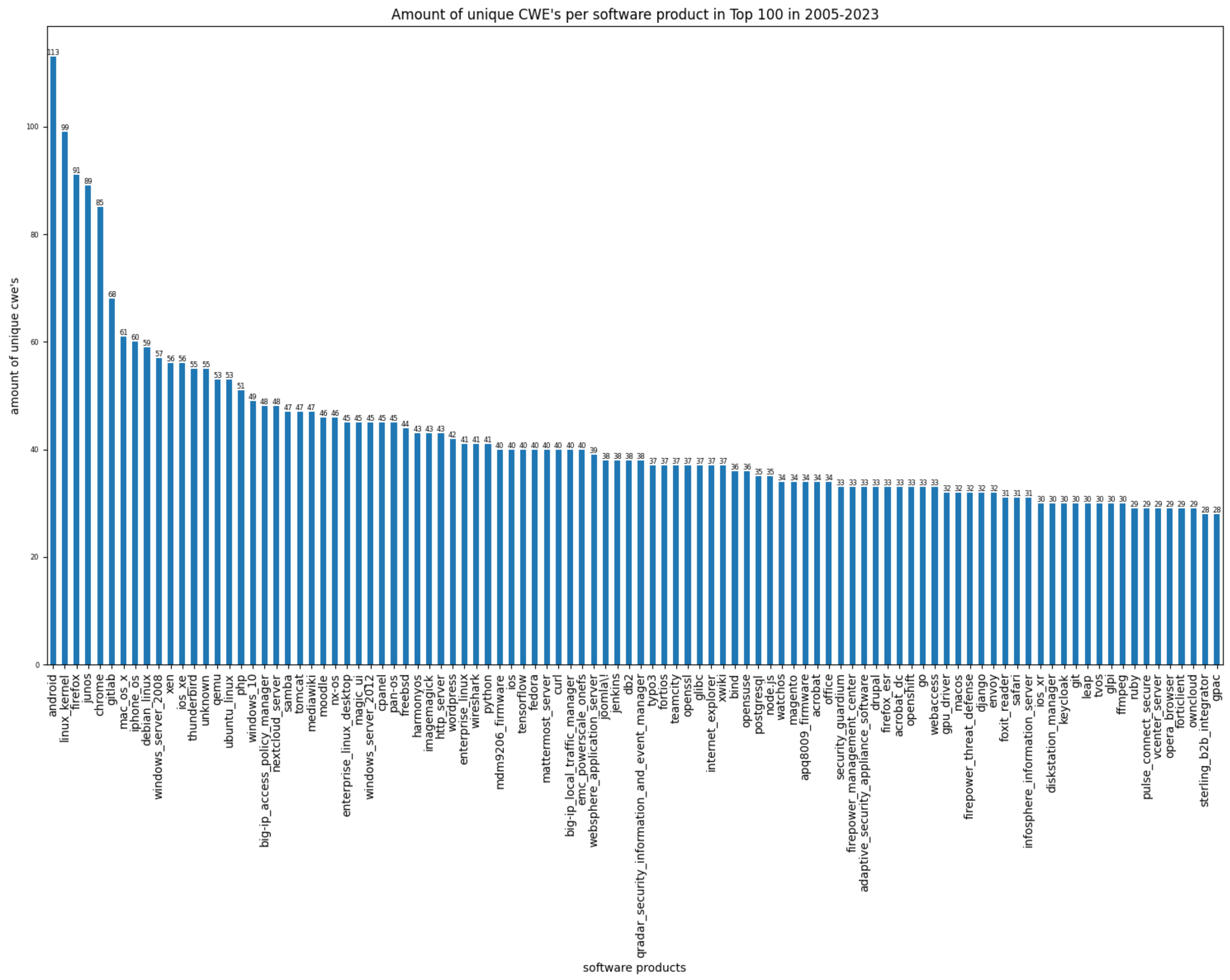


Figure 7: Number of items per user

After pruning, the data will be randomly divided into 5 folds according to a script available in my GitHub [57]. Every fold will once be used as test set where the remaining folds serve as training set. Every fold consists of 6653 data points, so for every iteration the model will be trained on a training set of 26612 data points and it will be tested on the test set of 6653 data points.

3.2 Recommender systems

Figure 8 shows the steps for implementing multiple recommender systems and selecting the top-performing one. All these steps are implemented as executable code that can be found in my GitHub repository [57]. A description of these steps is elaborated below:

1. From the library 'Lenskit.algorithms' [58] in Python, multiple recommender systems are implemented. The names of these recommender sytems are explained in Table 5. This table also contains the implemented parameter values which are chosen based on information and examples from the Lenskit documentation. For every recommender system the following steps will be performed:
   (a) As described in section 3.1, 5-fold cross-validation is implemented and for every fold the following steps will be performed:
      i. The training data in this fold is used to train the recommender system.
      ii. This trained recommender system is now used to make predictions. For every unique user in the test set, 5 recommendations are generated.
      iii. The test data in this fold is used to evaluate the performance of this recommender system. Evaluations are performed with the evaluation methods described in section 3.4.
2. The top-performing recommender system is selected based on the evaluation metrics from the previous step.

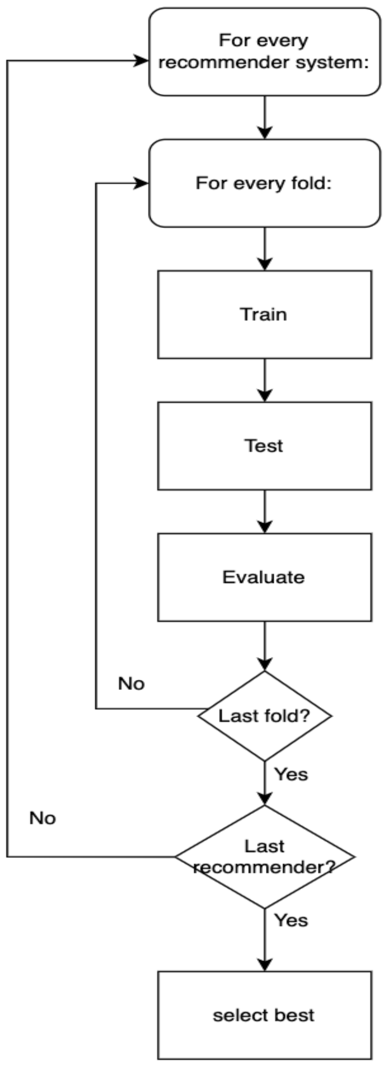


Figure 8: Diagram process recommender systems

Table 5: Names of algorithms from Lenskit library

Name algorithm	**Explanation**	**Parameters**
item_knn	Item-Based k-NN Collaborative Filtering	k=10 feedback=implicit
user_knn	User-based k-NN Collaborative Filtering	k=10 feedback=implicit
basic.Random	Random item recommender	
basic.Popular	Item recommender that recommends the most popular items the user has not already rated	
als.BiasedMF	Alternating least squares implementation of biased Matrix Factorization	features=50
als.ImplicitMF	Alternating least squares implementation of Matrix Factorization for implicit feedback	features=50
svd.BiasedSVD	Biased Matrix Factorization for implicit feedback using SciKit-Learn's SVD solver. It operates by first computing the bias, then computing the SVD of the bias residuals.	features=50
funksvd	FunkSVD Matrix Factorization. FunkSVD is a regularized biased Matrix Factorization technique trained with featurewise stochastic gradient descent	features=50
basic.Popular	Item recommender that recommends the most popular items the user has already rated or not	selector=AllItems-CandidateSelector

3.3 Active learning

Figure 9 shows the steps performed to implement active learning on the top-performing recommender system, which are again implemented as executable code [57]. Detailed explanation of these steps is elaborated:

1. Offline active learning is implemented with an adapted version of the strategy 'Expected Model Change' which is explained in section 2.3. Multiple implementations of learning with this strategy are considered and described later in this section. The implemented parameter values in the top-performing recommender system are the same as described in Table 5. For every implementation the following steps will be performed:

   (a) As described in section 3.1, 5-fold cross-validation is implemented where the same folds, with the same data are used as in section 3.2. For every fold the following steps will be performed:

      i. The training data in this fold is used to train the model which includes an implementation of active learning. From the training data, one user and all its items are removed. The removed data is used for an automatic oracle for active learning. When querying an item during active learning, the automatic oracle scans through the removed items and provides a label based on the label before the removal. The removal of a user is essential because it creates space for the introduction of new items with a correct label. Every fold has a different user removed which is hard coded in the executable code [57]. However, across the different implementations of active learning, the same user is consistently excluded for a given fold. This ensures that any difference in performance across different implementations of active learning is not caused by the difference in training data.

ii. The trained model is now used to make predictions. For every unique user in the test set, 5 recommendations are generated.

iii. The test data in this fold is used to evaluate the performance. Evaluations are performed with the evaluation methods described in section 3.4.

2. The outcomes of the evaluation metrics are compared using a Wilcoxon signed-rank test from the library 'Scipy.stat' in Python [59].

As already mentioned, active learning with an adapted version of the strategy 'Expected Model Change' is used. This strategy is coded by myself and can be found in the GitHub repository [57]. Generally, a sampling function will compute the performance score of the evaluation metric 'correct counts' for every distinct item. This calculation is performed under the assumption that the item, with label 1, is added to the training set on which the model is trained. This aims to capture the difference in performance by inclusion of this specific item. The sampling function will then return a sorted list with items arranged according to the greatest improvement in the 'correct counts' score. Various implementations for updating and adding items are considered:

1. **Random-20**
   This implementation will not use the strategy 'Expected Model Change'. In stead, it randomly selects 20 items and adds them to the training set with the correct label. When the item was already present in the training set it will not add it again. Random-20 serves as a benchmark to compare the performances of the active learning method.

2. **SingleBatch-20**
   SingleBatch-20 will make use of the strategy 'Expected Model Change'. It will add the top 20 items from the list returned by the sampling function all at once.

3. **SingleBatch-40**
   SingleBatch-40 has the same implementation as SingleBatch-20. However, instead of adding 20 items all at once it will add 40 items all at once.

4. **4Batch-20**
   4Batch-20 uses the sampling function 4 times with each iteration adding the top 5 items of the list returned by the sampling function to the training data. In the first iteration the sampling function is based on the initial model. For the subsequent iterations, the models trained on the updated training data, including the 5 items added in the previous iteration, are taken into consideration.

5. **4Batch-40**
   4Batch-40 employs the same strategy as 4Batch-20. However, instead of 5 items, 10 items are added in every iteration.

6. **10Batch-20**
   10Batch-20 is a variation of the 4Batch-20 strategy where instead of 4 batches, 10 batches are used. For every iteration 2 items are added.

7. **10Batch-40**
   10Batch-40 is the same as 10Batch-20 with for every iteration 4 items added instead of 2.

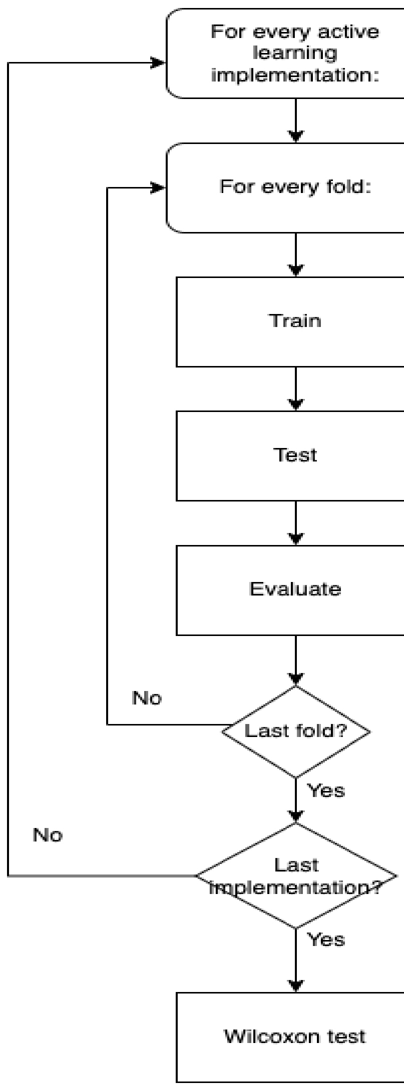


Figure 9: Diagram process active learning

3.4 Performance measure

Due to time constraints, a subset of the evaluation metrics, described in section 2.4 are employed to evaluate the performances of the recommender systems:

1. **Correct Counts**
   This evaluation metric is a simple counter that counts the amount of correct recommended items. Specifically, it determines whether the predicted items align with the items in the test

set.

2. **nDCG**
   Implementation of nDCG is based on nDCG from the library rank_metrics [60].
3. **Precision, Hit, Recip_rank**
   The evaluation metrics Precision, Hit, and Recip_rank are all imported from the library Lenskit.Metrics.topn [61].
4. **MAP@K**
   MAP@K is implemented using Metrics.Python.ml_metrics.average_precision in Python [62].
5. **Coverage, Novelty, Personalization**
   Coverage, Novelty, and Personalization are implemented by the library Recmetrics.metric [63].

4 Results

This section elaborates the outcomes of the analysis by evaluation metrics to assess the performance of multiple recommender systems. Subsequently, the performances of the best-performing recommender system with active learning using different sampling strategies is also evaluated using these evaluation metrics and the outcomes are presented in this section. Additionally, outcomes of a statistical test to validate a significant difference between the performances of active learning with various sampling strategies implemented are elaborated. An explanation of the used evaluation metrics can be found in the sections 3.4 and 2.4.

4.1 Recommender system

The outcomes of the evaluation metrics: precision and nDCG are presented in Figures 10 and 11. Results of the evaluation criteria: coverage, novelty, and personalization are presented in figures 12 till 14. In Appendix A.1 figures 19 till 22 represent the outcomes of evaluation metrics correct counts, hit, MAP@K, and recip_rank.

The graphs presented in this section serve as a visual illustration in understanding the differences between performances of multiple recommender systems. The names of the algorithms in the figures are explained in Table 14.

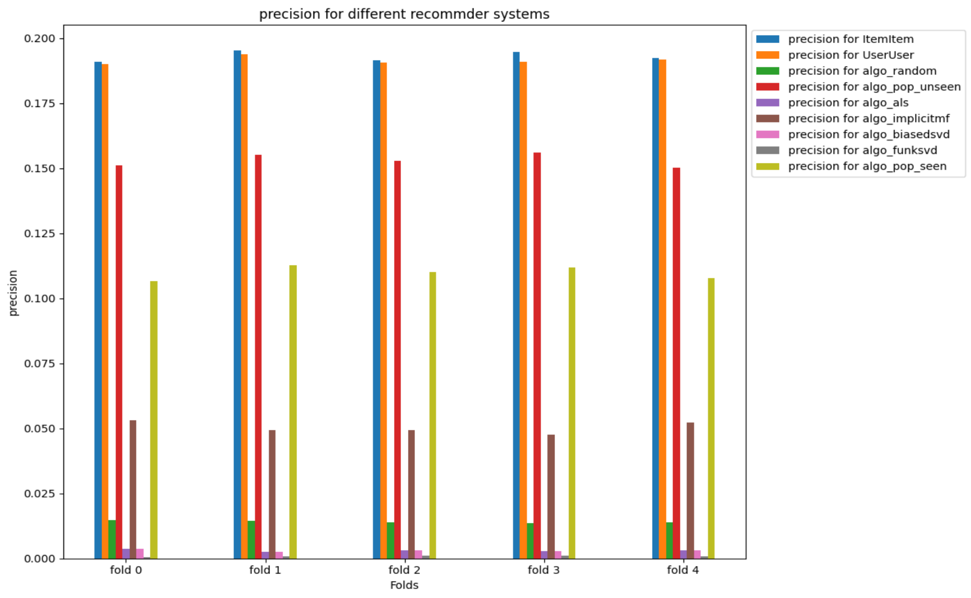


Figure 10: Precision per fold

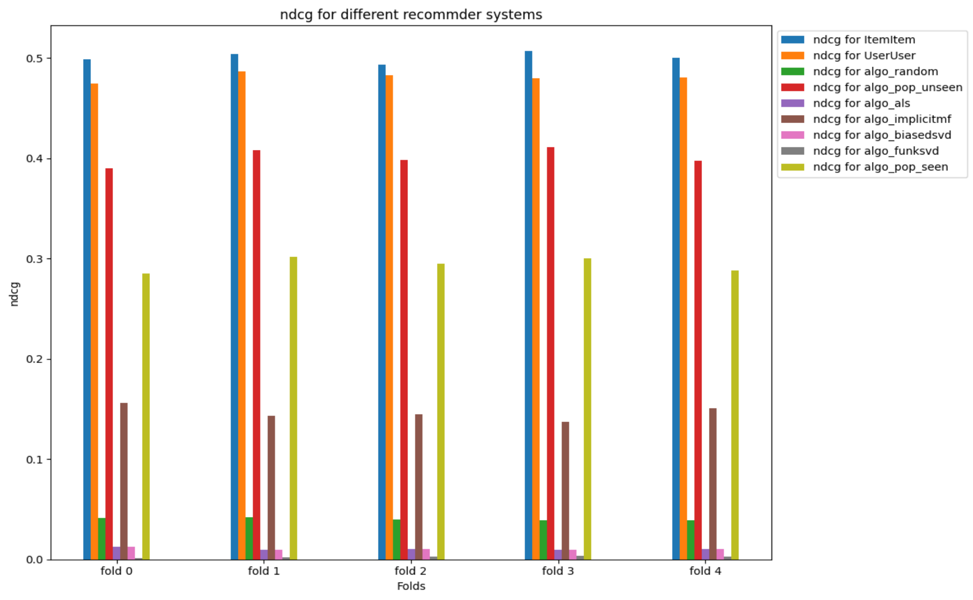


Figure 11: nDCG per fold

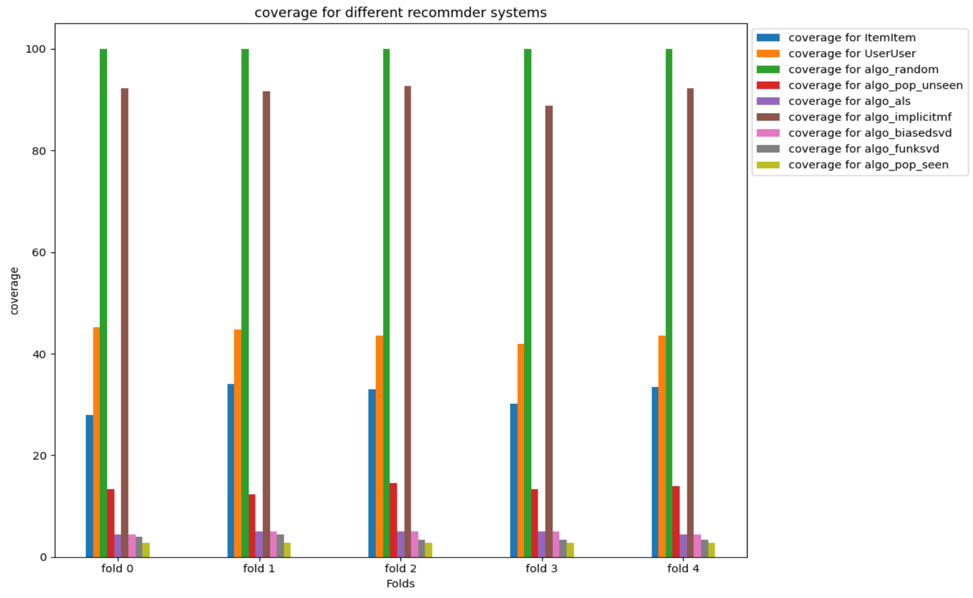


Figure 12: Coverage per fold

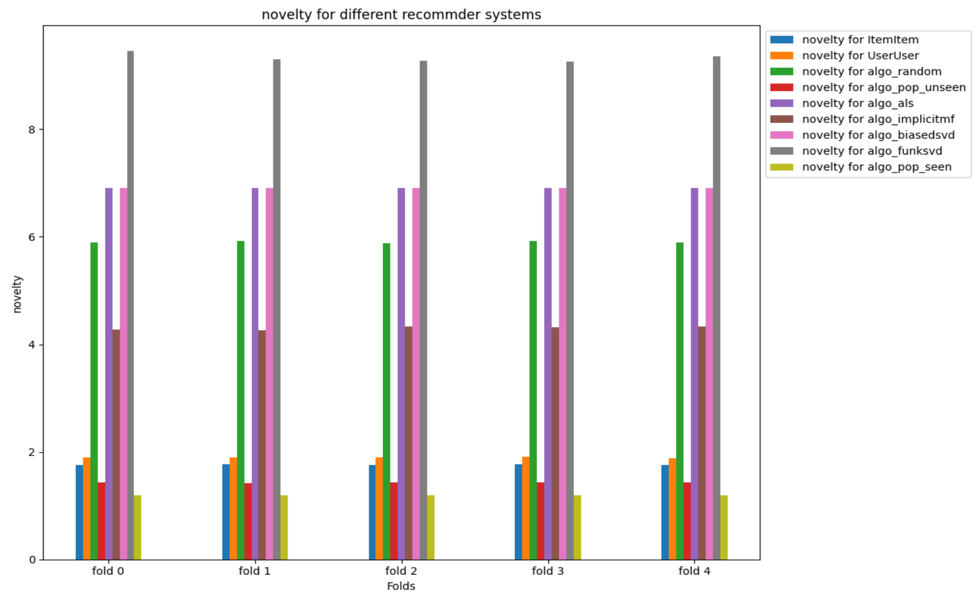


Figure 13: Novelty per fold

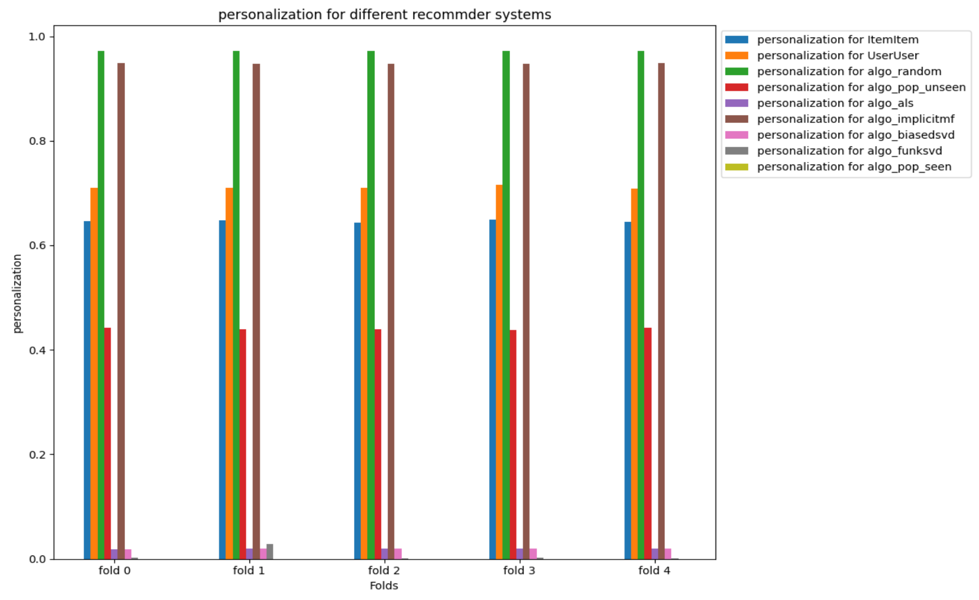


Figure 14: Personalization per fold

4.1.1 Discussion

Figure 10 and 11 indicate that the ItemItem has the best overall performance because it has the highest score on precision and nDCG. A high precision score, indicates a high percentage of correctly recommended items. Whereas, the nDCG score places importance on the overall precision and the order of the ranking. So, the ItemItem is the best in correctly recommending items and ranking them compared to the other recommender systems in figures 10 and 11.

Besides ItemItem, UserUser has high scores on precision, and nDCG. This indicates that k-NN Collaborative Filtering is well-suited for this type of recommender problems. A plausible explanation is that the data is very sparse and the k-NN Collaborative Filtering works well with sparse data. It can infer relationships between users and items by only looking at the user-item interaction data without any additional data. The ItemItem analyzes item similarity and can suggest items based on a user's previously rated items. Additionally, the UserUser analyzes user similarity and suggests items based on similar user preferences. This technique is beneficial in such sparse data. Furthermore, ItemItem and UserUser have higher precision and nDCG scores than the algo_random, algo_pop_unseen, and the algo_pop_seen. These recommenders use no intelligence in recommending as algo_random makes random recommendations and the algo_pop_unseen, and the algo_pop_seen make recommendations based on items that are popular. Therefore, it is anticipated that k-NN Collaborative Filtering, which uses a more intelligent approach, would outperform these recommenders.

As indicated in figures 10 and 11 the algorithm algo_als has low performance scores. This algorithm is suitable for explicit feedback data. Since the dataset used in this research is implicit feedback data the results align with the expectations.

Algorithms algo_biasedsvd, and algo_funksvd also have low performance scores. Matrix Factorization decomposes the user-item interaction data into two lower-dimension matrices that represent the latent features captured in the user-item interactions. With such sparse data the algorithm could have trouble inferring those two lower-dimensional matrices which makes it harder to make relevant recommendations. In addition, Matrix Factorization is more complex which could lead to overfitting or underfitting.

Figure 12 shows the predictions coverage per algorithm. As expected: the algo_random has a coverage of 100%, meaning this algorithm is able to generate recommendations for all user-items pairs in the dataset. In contrast, algo_pop_seen and algo_pop_unseen have a very low coverage percentage because they can only recommend the popular items. The coverage score for ItemItem and UserUser is higher than the coverage score of algo_pop_seen and algo_pop_unseen which indicates it considers more items to predict than only the popular items which could indicate the inferred relations are not solely based on popularity. In addition, having an extremely high coverage is not desirable since it might lead to general or less tailored recommendations.

The plots with the outcomes of the evaluation metric "correct counts" are placed in Appendix A. This evaluation metric solely counts the amount of accurate recommendations compared with the testing set. However, it should be noted that it does not consider the difference in amount of recommendations. Each recommender system is designed to generate 5 recommendations per unique user in the testing set. As the number of unique users varies per fold, the total number of recommendations is not equal. Thus, this evaluation metric may not produce an objective assessment.

Figure 12 provides the novelty scores of the recommender systems. It shows a very high novelty for the algo_funksvd. algo_funksvd uses stochastic gradient descent which minimizes the error between the recommended item and the real relevant items to the user. Doing so, it adjusts latent factors for users and items which could potentially reveal hidden patterns or connections between users and items that weren't visible in the data before and thus recommend items that before would not be considered. ItemItem and UserUser have relatively low novelty scores. This can be explained by the algorithms as both algorithms depend on user-item interactions which limit their ability to recommend items completely new to users/items. However, a low novelty score doesn't imply bad performance of the recommender system.

Lastly, Figure 14 shows that ItemItem and UserUser have personalization scores of more than 0.6 which indicates moderate levels of customization within the recommendation, meaning the recommendations are based on personal interests or behavior of the users. This is coherent since ItemItem and UserUser are based on user-item interactions. Also: algo_pop_seen, algo_random, algo_als, ItemItem, and UserUser score high on personalization. All these algorithms, except algo_random, also score high on the evaluation metrics for performance on recommender systems. It would be reasonable to expect a correlation between personalization score and the performance of a recommender system since recommendations are inherently individualized and depend on user's preferences. However, this wouldn't explain the high personalization score of the random recommender.

In conclusion, ItemItem is the top-performing recommender system based on the outcomes of the evaluation metric precision and nDCG. UserUser also has a high precision and nDCG score which indicates a good overall performance of k-NN Collaborative Filtering on such recommender system problems.

Table 6: Explanation of algorithms from Lenskit

Name algorithm	**Explanation**
ItemItem	Item-Based k-NN Collaborative Filtering
UserUser	User-based k-NN Collaborative Filtering
algo_random	Random item recommender
algo_pop_unseen	Item recommender that recommends the most popular items the user has not already rated
algo_als	Alternating least squares implementation of biased Matrix Factorization
algo_implicitmf	Alternating least squares implementation of Matrix Factorization for implicit feedback
algo_biasedsvd	Biased Matrix Factorization for implicit feedback using SciKit-Learn's SVD solver. It operates by first computing the bias, then computing the SVD of the bias residuals.
algo_funksvd	FunkSVD Matrix Factorization. FunkSVD is a regularized biased Matrix Factorization technique trained with featurewise stochastic gradient descent
algo_pop_seen	Item recommender that recommends the most popular items the user has already rated or not

4.2 Active learning

Active learning is applied on ItemItem. Multiple implementation of active learning are considered. Table 7 contains the naming convention used for every implementation of the algorithm.

Outcomes of the evaluation metrics: precision and nDCG for ItemItem are represented in Figures 15 and 16. These metrics are obtained by incorporating active learning with a sampling method that is either random, single-batch, 4-batch or 10 batch, and 20 or 40 items are added. The outcomes for evaluation metrics hit, correct counts and MAP@K, recip_rank are presented in Appendix A.2 in figures 23 till 26.

Figures 17 and 18 represent the learning curves where the performance scores of every fold are plotted against every additional item for every different implementation of active learning. For every different implementation of active learning, every fold has a different starting point since performance scores differ among different folds. No learning curve for ItemItem without active learning is visible since no learning is involved here. The learning curves for performance scores obtained with the evaluation metrics hit, correct counts and MAP@K, recip_rank are presented in Appendix A.2 in figures 27 till 30.

Table 8 contains the mean values of the performance scores from all evaluation metrics for ItemItem before active learning, or with Random-20, SingleBatch-20, SingleBatch-40, 4Batch-20, 4Batch-40, 10Batch-20, and 10Batch-40. In addition, tables 9 and 10 represent the mean values of precision and nDCG per different implementation of active learning. In Appendix A.2 tables 13 till 16 represent the mean values of the outcomes for each remaining evaluation method.

Finally, Appendix A includes more detailed graphs on the outcomes of all evaluation metrics on pairs of different implementations of active learning.

Table 7: Naming convention implementation algorithm

Naming convention	**detailed name**
Random-20	Item-based k-NN CF with random sampling active learning, adding 20 items.
SingleBatch-20	Item-based k-NN CF with single-batch active learning, adding 20 items.
SingleBatch-40	Item-based k-NN CF with single-batch active learning, adding 40 items.
4Batch-20	Item-based k-NN CF with 4-batch active learning, adding 20 items.
4Batch-40	Item-based k-NN CF with 4-batch active learning, adding 40 items.
10Batch-20	Item-based k-NN CF with 10-batch active learning, adding 20 items.
10Batch-40	Item-based k-NN CF with 10-batch active learning, adding 40 items.

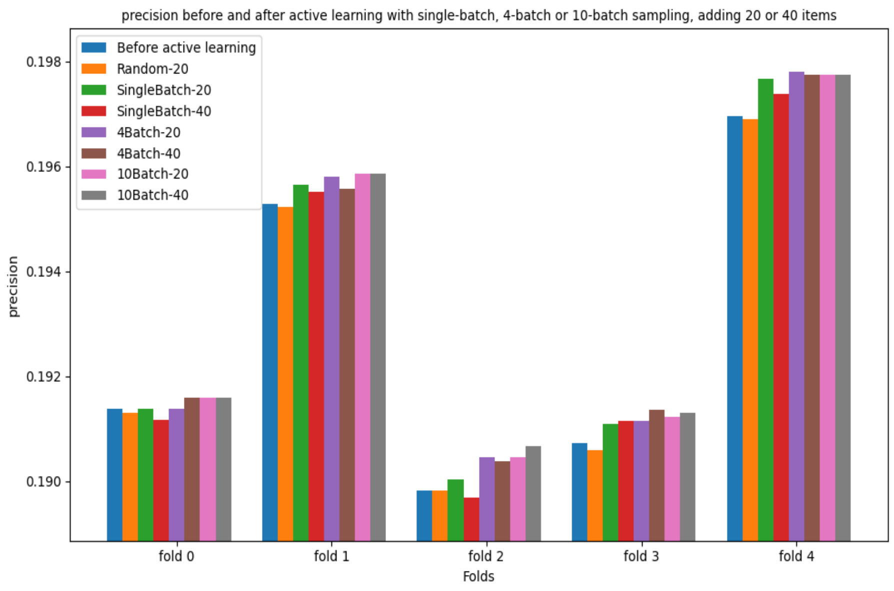


Figure 15: Precision per fold for every sampling method with 20 or 40 items added

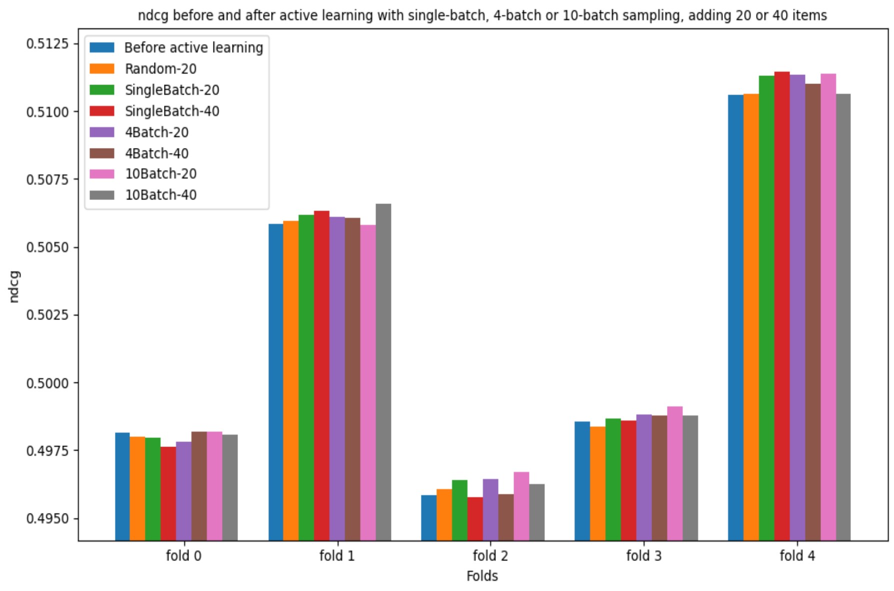


Figure 16: nDCG per fold for every sampling method with 20 or 40 items added

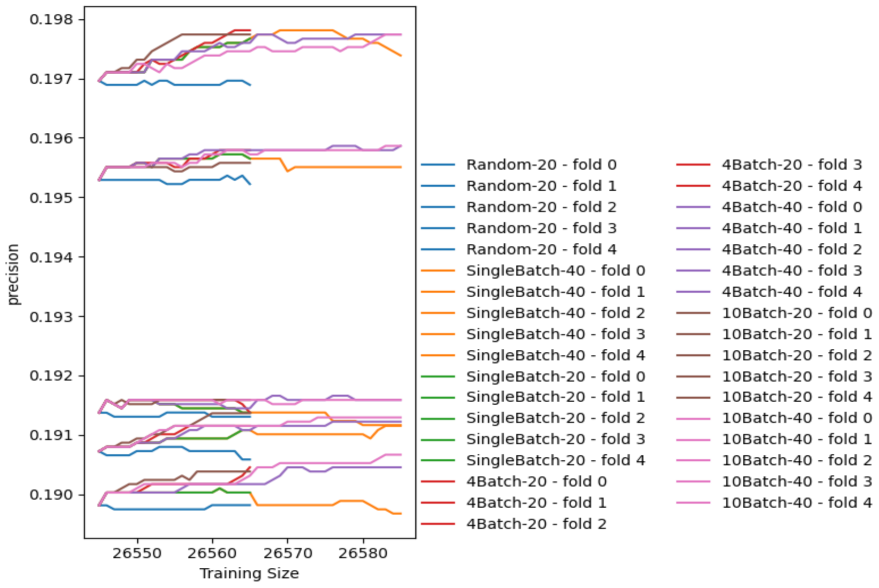


Figure 17: Learning curve precision per fold with all sampling methods, adding 20 or 40 items

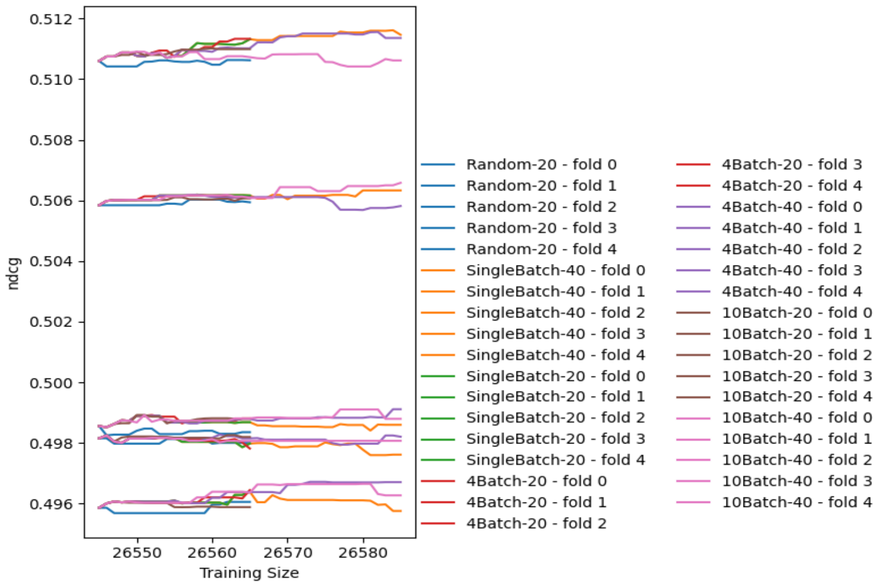


Figure 18: Learning curve nDCG per fold with all sampling methods, adding 20 or 40 items

Table 8: Mean values of the performance scores per sampling method

Algorithm	**Mean value performance score**
Random-20	453.852452
Before active learning	453.852480
SingleBatch-40	454.352581
SingleBatch-20	454.786093
4Batch-20	455.152813
10Batch-20	455.186071
4Batch-40	455.286249
10Batch-40	455.419503

Table 9: Mean values of precision per sampling method

Algorithm	**Mean value precision**
Random-20	0.192763
Before active learning	0.192832
SingleBatch-40	0.192976
SingleBatch-20	0.193160
4Batch-20	0.193316
10Batch-20	0.193330
4Batch-40	0.193373
10Batch-40	0.193429

Table 10: Mean values of nDCG per sampling method

Algorithm	**Mean value nDCG**
Random-20	0.501793
Before active learning	0.501801
SingleBatch-40	0.501951
10Batch-20	0.501985
10Batch-40	0.502065
4Batch-20	0.502098
SingleBatch-20	0.502106
4Batch-40	0.502238

4.2.0.1 Statistical Hypothesis Testing

A Wilcoxon singed-rank test is conducted to compare the precision and nDCG score of ItemItem with Before, Random-20, SingleBatch-20, SingleBatch-40, 4Batch-20, 4Batch-40, 10Batch-20, and 10Batch-40. The outcomes of the tests are listed in Tables 11 and 12. Outcomes that indicate a significant difference are marked.

Table 11: W, p-value for Wilcoxon signed-rank test on precision

Algorithm	Before	Random-20	SingleBatch-20	SingleBatch-40	4Batch-20	4Batch-40	10Batch-20	10Batch-40
Before	x	14.0, 0.0625	0.0, 1.0	3.0, 0.90625	0.0, 1.0	0.0, 1.0	0.0, 1.0	0.0, 1.0
Random-20	1.0, 0.96875	x	0.0, 1.0	3.0, 0.90625	0.0, 1.0	0.0, 1.0	0.0, 1.0	0.0, 1.0
SingleBatch-20	15.0, 0.03125	15.0, 0.03125	x	14.0, 0.0625	1.0, 0.96875	0.0, 1.0	2.0, 0.9375	0.0, 1.0
SingleBatch-40	12.0, 0.15625	12.0, 0.15625	1.0, 0.96875	x	0.0, 0.96606	0.0, 1.0	0.0, 1.0	0.0, 1.0
4Batch-20	15.0, 0.03125	15.0, 0.03125	14.0, 0.0625	10.0, 0.03394	x	1.0, 0.92794	8.0, 0.5	1.0, 0.96875
4Batch-40	15.0, 0.03125	15.0, 0.03125	15.0, 0.03125	15.0, 0.03125	9.0, 0.07206	x	4.0, 0.29649	0.0, 0.91014
10Batch-20	15.0, 0.03125	15.0, 0.03125	13.0, 0.09375	15.0, 0.03125	7.0, 0.59375	2.0, 0.70351	x	1.0, 0.85748
10Batch-40	15.0, 0.03125	15.0, 0.03125	15.0, 0.03125	15.0, 0.03125	14.0, 0.0625	3.0, 0.08986	5.0, 0.14252	x

Table 12: W, p-value for Wilcoxon signed-rank test on nDCG

Algorithm	**Before**	**Random-20**	**SingleBatch-20**	**SingleBatch-40**	**4Batch-20**	**4Batch-40**	**10Batch-20**	**10Batch-40**
Before	x	8.0, 0.5	2.0, 0.9375	6.0, 0.6875	3.0, 0.90625	1.0, 0.96875	0.0, 1.0	2.0, 0.9375
Random-20	7.0, 0.59375	x	1.0, 0.96875	5.0, 0.78125	2.0, 0.9375	1.0, 0.96875	2.0, 0.9375	1.0, 0.96875
SingleBatch-20	13.0, 0.09375	14.0, 0.0625	x	10.0, 0.3125	8.0, 0.5	4.0, 0.84375	10.0, 0.3125	8.0, 0.5
SingleBatch-40	9.0, 0.40625	10.0, 0.3125	5.0, 0.78125	x	5.0, 0.78125	4.0, 0.84375	7.0, 0.59375	5.0, 0.78125
4Batch-20	12.0, 0.15625	13.0, 0.09375	7.0, 0.59375	10.0, 0.3125	x	3.0, 0.90625	11.0, 0.21875	8.0, 0.5
4Batch-40	14.0, 0.0625	14.0, 0.0625	11.0, 0.21875	11.0, 0.21875	12.0, 0.15625	x	13.0, 0.09375	10.0, 0.3125
10Batch-20	15.0, 0.03125	13.0, 0.09375	5.0, 0.78125	8.0, 0.5	4.0, 0.84375	2.0, 0.9375	x	6.0, 0.6875
10Batch-40	13.0, 0.09375	14.0, 0.0625	7.0, 0.59375	10.0, 0.3125	7.0, 0.59375	5.0, 0.78125	9.0, 0.40625	x

4.2.0.2 Discussion

From Table 8 it can be concluded that ItemItem with 10Batch-40 has the best mean performance compared to the other sampling methods. ItemItem with 4Batch-40 is second best. On the third and fourth place respectively are 10Batch-20 and 4Batch-20. From this information, it is reasonable to draw the following conclusion: the influence of batch size in the active learning sampling method for ItemItem is less influential to the mean performance scores than the amount of items that are added to the dataset during active learning. Interestingly, when using active learning with a single-batch sampling method, the mean increment in performance of ItemItem is higher when 20 items are added in contrast to when 40 items are added. The underlying reason could be the interdependencies and effects among the items. With the single-batch sampling method the model considers the initial performance of the algorithm and finds the items that will improve this performance. With the batch sampling method, the model's state is considered after the insertion of the batch items.

The order of Table 9 corresponds with the order of Table 8. Here, ItemItem with a 10Batch-40 has the highest precision score. Additionally, ItemItem with Random-20 has the lowest precision score. However, this does not hold for Table 10 which represents the mean values of the nDCG scores per implemented sampling method. ItemItem with SingleBatch-20, 4Batch-20, 4Batch-40, 10Batch-20 and 10Batch-40 are at different ranks according highest nDCG score in comparison with the ranks for mean performance scores. In Table 10 the 4-batch-40 has the highest nDCG score. However, Table 12 indicates no significant difference between the performances of ItemItem with SingleBatch-20, 4Batch-20, 4Batch-40, 10Batch-20 and 10Batch-40. Specifically, the lack of statistical difference suggests that the ranking order may not be of paramount importance here.

Table 12 demonstrates that only for ItemItem Before and ItemItem 10Batch-20 the null hypothesis of the Wilcoxon Signed Ranked Test is rejected. This indicates a statistically significant improvement at $\alpha = 0.05$ in nDCG sore for ItemItem with 10Batch-20 compared to ItemItem with Before. However, ItemItem with 10Batch-20 doesn't demonstrate the biggest difference in mean value of the nDCG score against ItemItem with Before in Table 8. This can be attributed to the fact that the Wilcoxon signed-rank test examines the population median of the difference scores. Unlike the mean value which is obtained over all folds without focusing on the distribution of difference scores between paired samples.

Additionally, Table 11 highlights more instances with a statistically significant improvement at $\alpha = 0.05$ for the precision score. Here is revealed that SingleBatch-20, 4Batch-20, 4Batch-40, 10Batch-20 and 10-Batch 40 have a significant improvement in the precision score compared with ItemItem with Before and Random-20. 4Batch-40, 10Batch-40 have a significant improvement in precision score compared with SingleBatch-20. And, 4Batch-20, 4Batch-40, 10Batch-20, and 10Batch-40 have a significant improvement in precision score compared with SingleBatch-40. It therefore can be concluded that ItemItem with SingleBatch-20, 4Batch-20, 4Batch-40, 10Batch-20 and 10Batch-40 does statistically significantly improve the precision score of ItemItem with Before and Random-20. In addition, ItemItem with 4Batch-40 and 10Batch-40 has a significant improvement in precision score compared to ItemItem with SingleBatch-20. Lastly, ItemItem with 4Batch-20, 4Batch-40, 10Batch-20 and 10Batch-40 does significant improve the precision score of ItemItem with SingleBatch-40. The potential reasons are already detailed above.

Tables 11 and 12 differ in amount of instances that have a significant improvement in performance scores. Where table 11 has 16 instances with significant improvement compared to another instance, Table 12 only has 1 instance. Table 12 contains the outcomes of the Wilcoxon Singed Rank Test for the nDCG scores and Table 11 for the precision scores. Since in Table 12 0 active learning implementations show significant improvement compared to ItemItem with Random-20, it can be concluded that introducing active learning in ItemItem doesn't significantly improve the nDCG score. However, 10Batch-20 shows a significant improvement compared to Before. In contrast, there is no significant improvement with Random-20. This seems contradictory as there is a significant improvement in nDCG score for 10Batch-20 compared to Before, but no significant difference when comparing either 10Batch-20 to Random-20 or Random-20 to Before. This could potentially be a Type I error (false positive) which could be caused by the small sample size of only 5 performance scores in this Wilcoxon signed-rank test. Since, a small sample size reduces the statistical power of the Wilcoxon signed-rank test. Further investigation is needed to understand the underlying factors contributing to this consistency.

From the fact that there are more statistically significant improvements in precision score then in nDCG score when using multiple implementations of active learning in ItemItem, it could be concluded that active learning improves the ability of ItemItem to make correct recommendations. However, it does not necessarily improve its ability to rank those recommendations, as evidenced by the relatively stable nDCG score across various active learning implementations. This could be explained by the behavior of active learning with the query strategy 'Expected Model Change', where active learning will query the items that will result in the most correct recommendations without considering their rank. This does not give additional information about the rankings of the items which implies no improvement in nDCG score.

Figure 15 illustrates the precision score per fold per implementation of ItemItem and active learning. This figure serves as a visual representation of Table 9. Similarly, figure 16 illustrates the nDCG score per fold per implementation and serves as visual representation of the Table 10.

In addition, in figures 15 and 16, the differences in precision and nDCG scores among the different folds are visible. Where fold 4 holds the highest and fold 2 the lowest overall precision and nDCG. A potential explanation for the difference between the folds could be that fold 4 has the most favorable training data. Since, cross fold validation involves random splitting to divide the data into folds, one fold may contain data with more meaningful interaction the model can use to make recommendations. Also, the data in the test set of fold 4 could contain items that the model demonstrates more confidence in its recommendations. Difference in the performance scores of folds could also be caused by the different users and items being removed per fold. Some users and items could influence the user-item interactions more than others.

In figure 17 and 18, it is evident that, for each fold, the precision and nDCG score for the first batch of the 4Batch-20, 4Batch-40, 10Batch-20 and 10Batch-40 are the same. Each strategy has a different amount of items per batch: 4Batch-20 has 5 items, 4Batch-40 has 10 items, 10Batch-20 has 2 items and 10Batch-40 has 4 items. This is visible in the plots since for these strategies the precision and nDCG score of the amount of items in their first batches are the same. For SingleBatch-20 and SingleBatch-40 this also holds for all the 20 items of the SingleBatch-20. This aligns with the expectations since the first batch of every strategy anticipates on the first state of the model, which is for all strategies the same.

In summary, this discussion has explored the results of the research focusing on the impact of active learning on a recommender system to predict vulnerabilities in software products. The results reveal that ItemItem has the highest performance scores for predicting vulnerabilities in software products. In addition, ItemItem with 4Batch-20, 4Batch-40, 10Batch-20 and 10Batch-40 results in a statistically significant improvement in precision against ItemItem with Before and Random-20. For ItemItem with SingleBatch there was only a statistically significant improvement when 20 items were added instead of 40. For the nDCG scores, only ItemItem with 10Batch-20 showed a statistically significant improvement compared to ItemItem with Before. However, this could be a type I error. Overall, introducing active learning in ItemItem leads to significant improvement in precision score but not in nDCG score.

5 Conclusion

We now live in a time where we are very reliant on digital technologies, which elevates the importance of cybersecurity. To strengthen cybersecurity, it would be beneficial to discover new vulnerabilities in software products with Artificial Intelligence. This research introduces a novel approach where a recommender system is implemented to predict vulnerabilities in software products and its performance is improved with active learning. Multiple recommender systems are implemented on cybersecurity data from the NVD [1]. Notably, the Item-Based k-NN Collaborative Filtering achieves the highest performance scores on predicting vulnerabilities. To improve its performance, active learning with an adapted version of query strategy 'Expected Model Change' is incorporated with various update methods: a single-batch, a 4-batch, or a 10-batch and either 20 or 40 items are added. The evaluation metrics and statistical tests show a significant improvement on precision for the single-batch strategy adding 20 items, and the 4-batch and 10-batch adding 20 and 40 items compared to Item-Based k-NN Collaborative Filtering without active learning or with active learning and a random sampling strategy. For nDCG score only Item-Based k-NN Collaborative Filtering with active learning and the 10-batch sampling strategy, adding 20 items showed significant improvement compared to Item-Based k-NN Collaborative Filtering without active learning which could be a potential type I error. Therefore, it can be concluded that introducing active learning, with an adapted version of the query strategy 'Expected Model Change' that focuses on the number of correct recommendations rather than rankings, will improve the precision score of Item-Based k-NN Collaborative Filtering when a 4-batch or 10-batch update strategy is used or a single-batch strategy only with 20 items added instead of 40. It is evident these findings form the answer to the main research question: "How does active learning affect the performance of recommender systems to identify vulnerabilities in software products?". In conclusion, active learning significantly improves the precision of Item-Based k-NN Collaborative Filtering to predict vulnerabilities in software products but not necessarily the nDCG score and thus the ability to rank the vulnerabilities.

Furthermore, the investigation into sub-questions introduced in the introduction has contributed to answering the main research question. The findings of these sub-questions are elaborated below.

1. **Data collection and pre-processing**
   (a) *How to frame the prediction of vulnerabilities in software products as a problem suitable for a recommender system?*
   To formulate the prediction of vulnerabilities in software products as a recommender system problem we need to specify the users and items. As elaborated in section 2.2, the objects represented with "u" correspond to the software products that are the users in this problem. The objects represented with "i" are the vulnerabilities (the CWEs) which are the items. r_{ui} represents the rating user "u" gave to item "i". Where r_{ui} could be a 1 indicating the vulnerability "i" is present in software product "u" and a 0 otherwise.

   (b) *How to collect and pre-process data for recommender systems with active learning?*
   Data is collected from the NVD database and analyzed using visual graphs as presented in section 3.1. The CWE and software product name are subtracted and used to make an interaction sequence where each software product name is listed with the CWE it contains. To address the cold start problem, software products with less than 5 CWEs

are pruned from the dataset. Additionally, the data is randomly distributed over 5 folds where every fold is once used as test set.

2. **Recommender systems**

   (a) *What types of recommender systems are suitable for predicting vulnerabilities in software products?*
   Taking into account that the data is sparse with implicit feedback and the ratings are binary, suitable recommender system algorithms are item- and user-based k-NN Collaborative Filtering, Alternating least squares implementation of Matrix Factorization with implicit feedback, Biased Matrix Factorization for implicit feedback, and FunkSVD Matrix Factorization.

   (b) *How can a recommender system be implemented for predicting vulnerabilities in software products?*
   To implement the recommender systems, the Python library 'Lenskit' is used. These recommender system models are fitted on the training data and tested on the test data with cross fold validation.

3. **Active learning**

   (a) *What types of strategies in active learning are suitable for implementing active learning in recommender systems to predict vulnerabilities in software products?*
   Pool-based active learning with an adapted version of the query strategy 'Expected Model Change' is used to implement active learning which is explained in section 2.3.2. Section 2.3 contains more information about other suitable strategies.

   (b) *How can active learning be implemented in a recommender system for predicting vulnerabilities in software products?*
   Active learning is implemented by programming an adaption version of 'Expected Model Change' query strategy. For each item, the change in number of correct recommendations when this item is added with label 1 to the dataset, is calculated and the item that causes the highest number of correct recommendations is queried. The item, upon its return, will be assigned a label of 0 or 1 and will be added to the dataset.

4. **Evaluations**

   (a) *How to measure the performance of the recommender system and active learning?*
   To measure the performance of the recommender systems and active learning, the evaluation metrics: count of correct predicted items, HIT, MAP@K, nDCG, recip_rank, and precision are used. Evaluation of the performances with these metrics is done before active learning and after active learning with various sampling strategies. Additionally, the evalaution methods: novelty, personalization, and coverage are implemented to reveal characteristics of the recommender systems

   (b) *How to compare the performances of the recommender system before and after active learning?*
   The performance scores, of each evaluation metric on the recommender system before and after active learning with various sampling strategies, are compared using a statistical test named Wilcoxon signed-rank test.

5.1 limitations and future work

While this research provides valuable insights, it is essential to mention its limitations. A key limitation is the limitation of the Wilcoxon signed-rank test that is used to compare performance scores. The statistical power of the Wilcoxon signed-rank test is reduced when using the test on a small sample size. Since, 5-fold cross-validation is used every implementation of Item-Based k-NN Collaborative Filtering with or without active learning has 5 performance scores. This results in a small sample size in the Wilcoxon signed-rank test which contributes to less confidence in the outcomes. An additional limitation is the limitation of the nDCG score used on this type of data. nDCG is designed to handle the graded relevance of items and ranking values. However, the data used in this research is binary which causes the model to make no distinction between degrees of relevance and it treats all non-zero values equally. This could result in incorrect nDCG scores. Furthermore, the query strategy used in active learning focuses on the increment of correct recommendations rather than ranking. Therefore, it is hard to make a general conclusion about the improvement in nDCG score when using active learning. Additionally, the Item-Based k-NN Collaborative Filtering with active learning using a 10-batch sampling strategy and adding 40 items took approximately 50 hours to run. With limited resources and time for this research using a sampling method with more than 10 batches and 40 items was not feasible despite the potential for improved outcomes. Due to this limited time, no additional investigation on varying the K in K-fold cross-validation, the amount of batches in active learning, the parameters used in the implementation of recommender systems, and the amount of items added in active learning could be performed. Lastly, an interesting observation is that the Item-Based k-NN Collaborative Filtering has the potential to recommend items that may not appear in the test set but do exist in real-life scenarios. So, it could predict a vulnerability in a software product that is classified as incorrect but is correct. This could eventually lead to inaccurate performance scores.

These limitations introduce some potential directions for future work. It would be interesting to investigate Item-Based k-NN Collaborative Filtering with other query strategies in active learning and how they contribute to the nDCG and precision score to investigate the influence of the query strategy on the performance scores. Also, adding more items during active learning to identify the optimal active learning implementation for this problem would be interesting. Finally, research where more features of the data like CVE score, vendor names, and version numbers are used in recommender systems could be beneficial.

References

[1] H. Booth, D. Rike, and G. Witte, "The national vulnerability database (nvd): Overview," 2013.

[2] "Ordina - cybersecurity and compliance." `https://www.ordina.com/what-we-do/focus-areas/cybersecurity-compliance/`.

[3] G. Jabeen, S. Rahim, W. Afzal, D. Khan, A. A. Khan, Z. Hussain, and T. Bibi, "Machine learning techniques for software vulnerability prediction: a comparative study," *Applied Intelligence*, pp. 1–22, 2022.

[4] S. S. Murtaza, W. Khreich, A. Hamou-Lhadj, and A. B. Bener, "Mining trends and patterns of software vulnerabilities," *Journal of Systems and Software*, vol. 117, pp. 218–228, 2016.

[5] J. Gope and S. K. Jain, "A survey on solving cold start problem in recommender systems," *International Conference on Computing*, pp. 133–138, 2017.

[6] M. Elahi, M. Braunhofer, F. Ricci, and M. Tkalcic, "Personality-based active learning for collaborative filtering recommender systems," in *AI* IA 2013: Advances in Artificial Intelligence: XIIIth International Conference of the Italian Association for Artificial Intelligence, Turin, Italy, December 4-6, 2013. Proceedings 13*, pp. 360–371, Springer, 2013.

[7] E. Wåreus and M. Hell, "Automated cpe labeling of cve summaries with machine learning," in *Detection of Intrusions and Malware, and Vulnerability Assessment: 17th International Conference, DIMVA 2020, Lisbon, Portugal, June 24–26, 2020, Proceedings 17*, pp. 3–22, Springer, 2020.

[8] "Nvd common platform enumeration (cpe)." Accessed: December 12, 2023.

[9] "CVE - common vulnerabilities and exposures." `https://cve.mitre.org/`, 4 2023. Accessed: April 3, 2023.

[10] Y. Wu, I. Bojanova, and Y. Yesha, "They know your weaknesses–do you?: Reintroducing common weakness enumeration," *CrossTalk*, vol. 45, 2015.

[11] K. Scarfone and P. Mell, "An analysis of cvss version 2 vulnerability scoring," in *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pp. 516–525, IEEE, 2009.

[12] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets," in *2008 Eighth IEEE international conference on data mining*, pp. 263–272, Ieee, 2008.

[13] D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich, *Recommender systems: an introduction.* Cambridge University Press, 2010.

[14] D. Chen, "Recommender system matrix factorization," July 2020. [Online; posted 8-July-2020].

[15] J. B. Schafer, J. A. Konstan, and J. Riedl, "Collaborative filtering recommender systems," *The adaptive web*, vol. 1, pp. 291–324, 2007.

[16] M. Fu, H. Qu, Z. Yi, L. Lu, and Y. Liu, "A novel deep learning-based collaborative filtering model for recommendation system," *IEEE transactions on cybernetics*, vol. 49, no. 3, pp. 1084–1096, 2018.

[17] J. S. Breese, D. Heckerman, and C. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," *arXiv preprint arXiv:1301.7363*, 2013.

[18] X. Su and T. M. Khoshgoftaar, "A survey of collaborative filtering techniques," *Advances in artificial intelligence*, vol. 2009, 2009.

[19] F. Fkih, "Similarity measures for collaborative filtering-based recommender systems: Review and experimental comparison," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 9, pp. 7645–7669, 2022.

[20] H. Kim, "Matrix factorization part i: Understanding all the processes and how stochastic gradient descent step is derived," 2022.

[21] D. Bokde, S. Girase, and D. Mukhopadhyay, "Matrix factorization model in collaborative filtering algorithms: A survey," *Procedia Computer Science*, vol. 49, pp. 136–146, 2015.

[22] Y. Koren, S. Rendle, and R. Bell, "Advances in collaborative filtering," *Recommender systems handbook*, pp. 91–142, 2021.

[23] S. Funk, "Netflix update: Try this at home," 2006.

[24] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[25] A. Mnih and R. R. Salakhutdinov, "Probabilistic matrix factorization," *Advances in neural information processing systems*, vol. 20, 2007.

[26] M. Elahi, F. Ricci, and N. Rubens, "A survey of active learning in collaborative filtering recommender systems," *Computer Science Review*, vol. 20, pp. 29–50, 2016.

[27] M. Shuaibi, S. Sivakumar, R. Q. Chen, and Z. W. Ulissi, "Enabling robust offline active learning for machine learning potentials using simple physics-based priors," *Machine Learning: Science and Technology*, vol. 2, no. 2, p. 025007, 2020.

[28] B. Settles, "Active learning literature survey," 2009.

[29] C. Deng, X. Ji, C. Rainey, J. Zhang, and W. Lu, "Integrating machine learning with human knowledge," *iScience*, vol. 23, p. 101656, 10 2020.

[30] D. Angluin, "Queries and concept learning," *Machine learning*, vol. 2, pp. 319–342, 1988.

[31] R. Wang, X.-Z. Wang, S. Kwong, and C. Xu, "Incorporating diversity and informativeness in multiple-instance active learning," *IEEE Transactions on Fuzzy Systems*, vol. 25, no. 6, pp. 1460–1475, 2017.

[32] D. Berrar *et al.*, "Cross-validation.," 2019.

[33] M. Chen and P. Liu, "Performance evaluation of recommender systems," *International Journal of Performability Engineering*, vol. 13, no. 8, p. 1246, 2017.

[34] Nillsf, "Confusion matrix: Accuracy, recall, precision, false positive rate, and f-scores explained," 2020. `https://blog.nillsf.com/index.php/2020/05/23/confusion-matrix-accuracy-recall-precision-false-positive-rate-and-f-scores-explained/`.

[35] M. Deshpande and G. Karypis, "Item-based top-n recommendation algorithms," *ACM Transactions on Information Systems (TOIS)*, vol. 22, no. 1, pp. 143–177, 2004.

[36] Y. Wang, L. Wang, Y. Li, D. He, and T.-Y. Liu, "A theoretical analysis of ndcg type ranking measures," in *Conference on learning theory*, pp. 25–54, PMLR, 2013.

[37] P. Xia, L. Zhang, and F. Li, "Learning similarity with cosine similarity ensemble," *Information sciences*, vol. 307, pp. 39–52, 2015.

[38] C. Longo, "Evaluation metrics for recommender systems," 2018.

[39] S. Vargas and P. Castells, "Rank and relevance in novelty and diversity metrics for recommender systems," in *Proceedings of the fifth ACM conference on Recommender systems*, pp. 109–116, 2011.

[40] R. S. Witte and J. S. Witte, *Statistics*. John Wiley & Sons, 2017.

[41] W. W. Daniel and C. L. Cross, *Biostatistics: a foundation for analysis in the health sciences*. Wiley, 2018.

[42] "Stattrek's online t-distribution calculator." `https://stattrek.com/online-calculator/t-distribution?utm_content=cmp-true`.

[43] J. Brownlee, "Statistical significance tests for comparing machine learning algorithms," August 8 2019.

[44] B. R. Sziklai, M. Baranyi, and K. Héberger, "Testing rankings with cross-validation," *arXiv preprint arXiv:2105.11939*, 2021.

[45] Z. Hanusz, J. Tarasinska, and W. Zielinski, "Shapiro–wilk test with known mean," *REVSTAT-Statistical Journal*, vol. 14, no. 1, pp. 89–100, 2016.

[46] A. King and R. Eckersley, "Chapter 7-inferential statistics iv: Choosing a hypothesis test, statistics for biomedical engineers and scientists," 2019.

[47] "Nonparametric statistics: The wilcoxon signed-rank test," 2023. Accessed: December 2, 2023.

[48] "Tables for the wilcoxon signed-rank test." PDF document. Retrieved from: `https://users.stat.ufl.edu/~winner/tables/wilcox_signrank.pdf`.

[49] O.-P. Suciu, *Data-Driven Techniques for Vulnerability Assessments*. PhD thesis, University of Maryland, College Park, 2021.

[50] M. Edkrantz and A. Said, "Predicting exploit likelihood for cyber vulnerabilities with machine learning," *Unpublished Master's Thesis, Chalmers Unıversıty of Technology Department of Computer Science and Engineering, Gothenburg, Sweden*, pp. 1–6, 2015.

[51] N. Polatidis, E. Pimenidis, M. Pavlidis, S. Papastergiou, and H. Mouratidis, "From product recommendation to cyber-attack prediction: Generating attack graphs and predicting future attacks," *Evolving Systems*, vol. 11, pp. 479–490, 2020.

[52] K. B. Lyons, "A recommender system in the cyber defense domain," 2014.

[53] I. Forain, R. de Oliveira Albuquerque, and R. T. de Sousa Júnior, "Revs: A vulnerability ranking tool for enterprise security.," in *ICEIS (2)*, pp. 126–133, 2022.

[54] P. Huff, K. McClanahan, T. Le, and Q. Li, "A recommender system for tracking vulnerabilities," in *Proceedings of the 16th International Conference on Availability, Reliability and Security*, pp. 1–7, 2021.

[55] R. Karimi, C. Freudenthaler, A. Nanopoulos, and L. Schmidt-Thieme, "Active learning for aspect model in recommender systems," in *2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, pp. 162–167, IEEE, 2011.

[56] S. Airen and J. Agrawal, "Movie recommender system using k-nearest neighbors variants," *National Academy Science Letters*, vol. 45, no. 1, pp. 75–82, 2022.

[57] E. Stijger, "Recsys_predicting_vulnerabilities." `https://github.com/elisestijger/Recsys_predicting_vulnerabilities`, 2023.

[58] M. D. Ekstrand, J. O'Donovan, S. M. McNee, and et al., "Lenskit." `https://github.com/lenskit/lkpy`, 2023. Accessed: December 2, 2023.

[59] SciPy Contributors, "scipy.stats.wilcoxon." `https://scipy.github.io/devdocs/reference/generated/scipy.stats.wilcoxon.html`.

[60] kmbnw, "Python script for ndcg calculation." `https://github.com/kmbnw/rank_metrics/blob/master/python/ndcg.py`.

[61] Lenskit Contributors, "Lenskit topn metrics module." `https://github.com/lenskit/lkpy/blob/main/lenskit/metrics/topn.py`.

[62] Wendykan, "Metrics average precision module." `https://github.com/benhamner/Metrics/blob/master/Python/ml_metrics/average_precision.py`.

[63] statisticianinstilettos, "Recmetrics metrics module." `https://github.com/statisticianinstilettos/recmetrics/blob/master/recmetrics/metrics.py`.

Appendices

A results

A.1 Recommender system

The outcomes of the evaluation metrics: correct counts, recip_rank, MAP@K and hit per different recommender system are depicted in the Figures 19 till 22.

The graphs presented in this section serve as visual illustration in understanding the differences between performances of multiple recommender systems. The names of the algorithms in the Figures 19 till 22 are explained in Table 14.

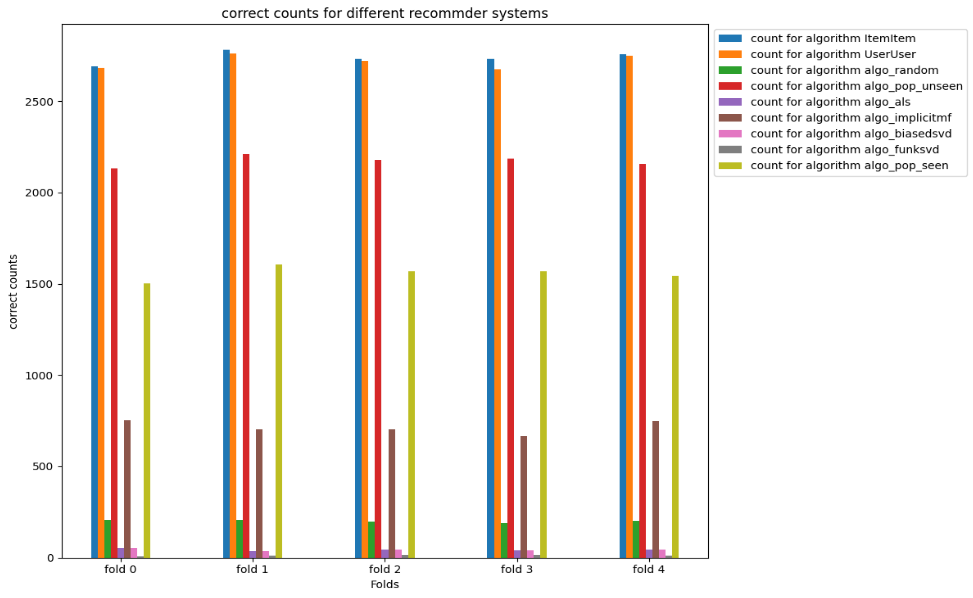


Figure 19: Count of correct predicted items per fold

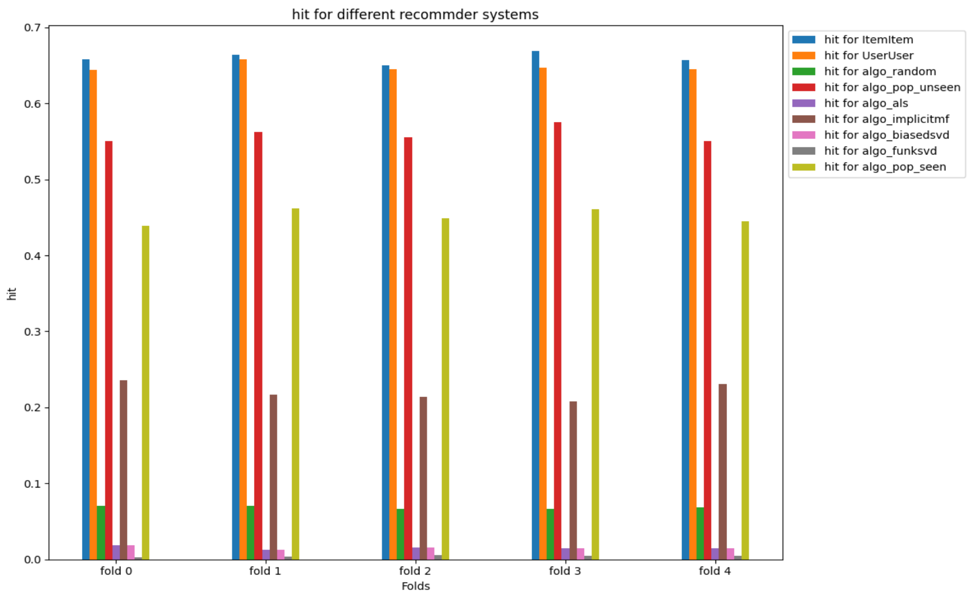


Figure 20: Hit per fold

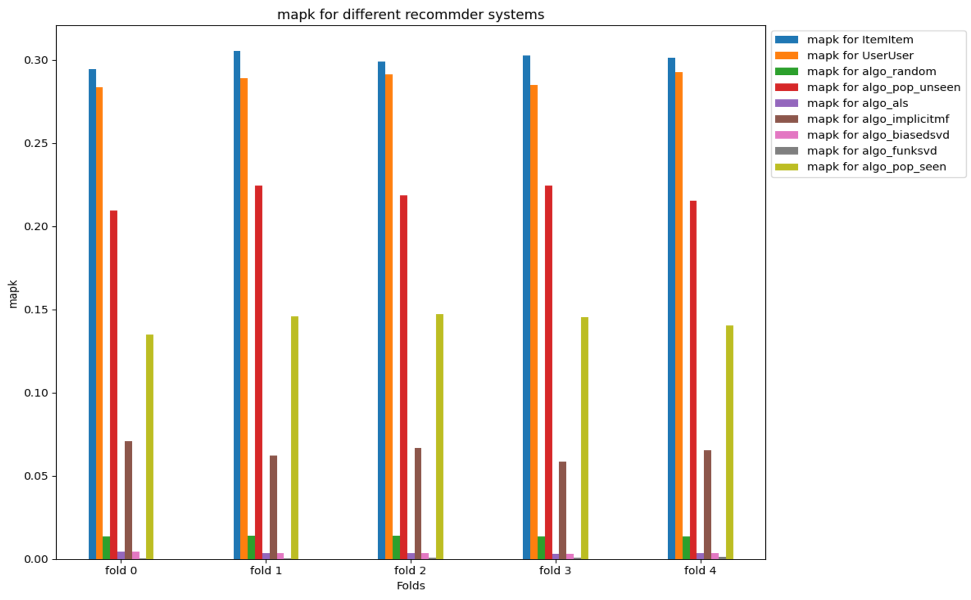


Figure 21: MAP@K per fold

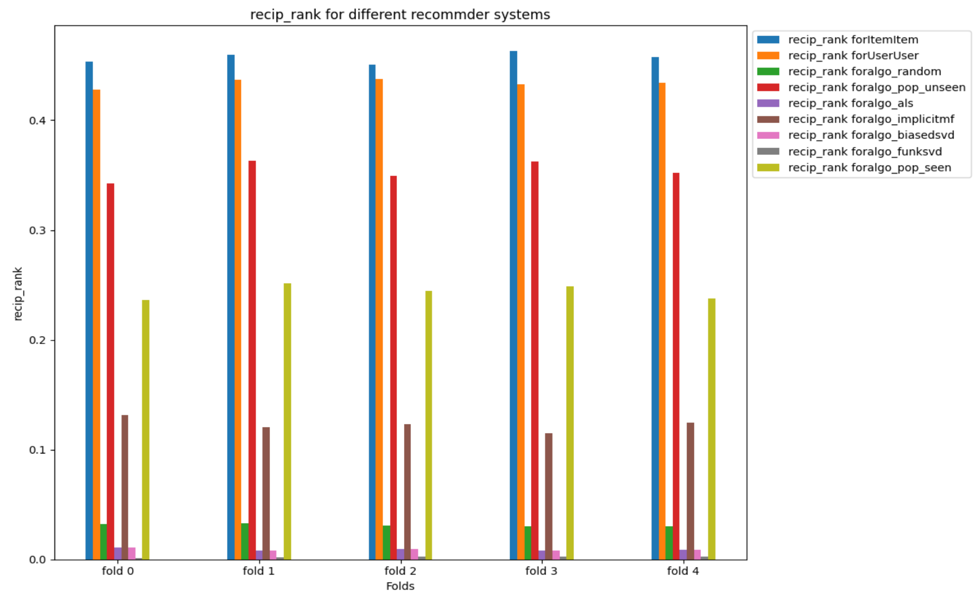


Figure 22: Recip_rank per fold

A.2 All sampling strategies adding 20 or 40 items

Outcomes of the evaluation metrics: count of correct predicted items, HIT, MAP@K, and recip_rank for Item-Based k-NN Collaborative Filtering are represented in Figures 23 till 26. These metrics are obtained by incorporating active learning with a sampling method that is either random, single-batch, 4-batch or 10 batch, and 20 or 40 items are added.

Figures 27 till 30 represent the learning curves where the performance scores of every fold is plotted against every additional item for every sampling method. For every sampling method, every fold has a different starting point since performance scores differ among different folds.

Tables 13 till 16 represent the mean values of all outcomes for each evaluation method seperate.

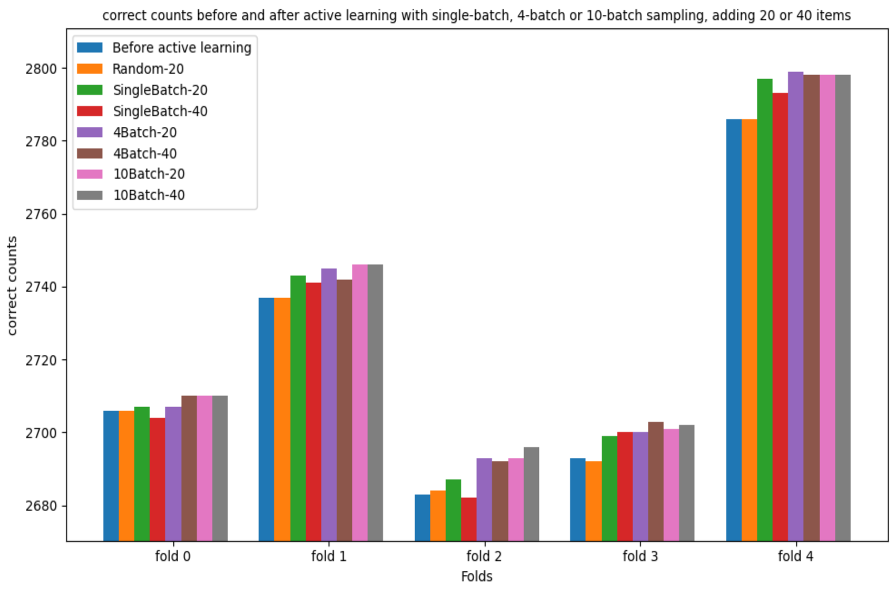


Figure 23: Count correct recommendations per fold for every sampling method with 20 or 40 items added

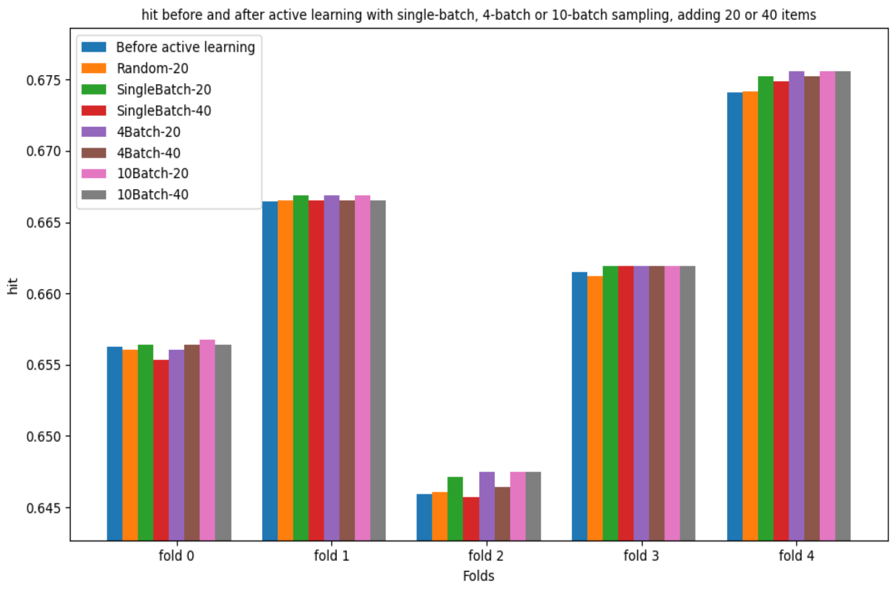


Figure 24: Hit per fold for every sampling method with 20 or 40 items added

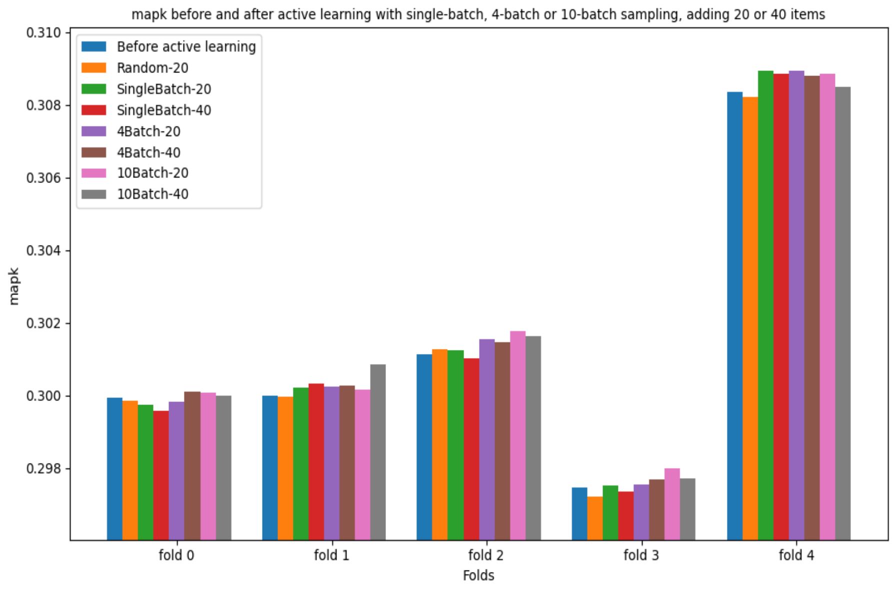


Figure 25: MAP@K per fold for every sampling method with 20 or 40 items added

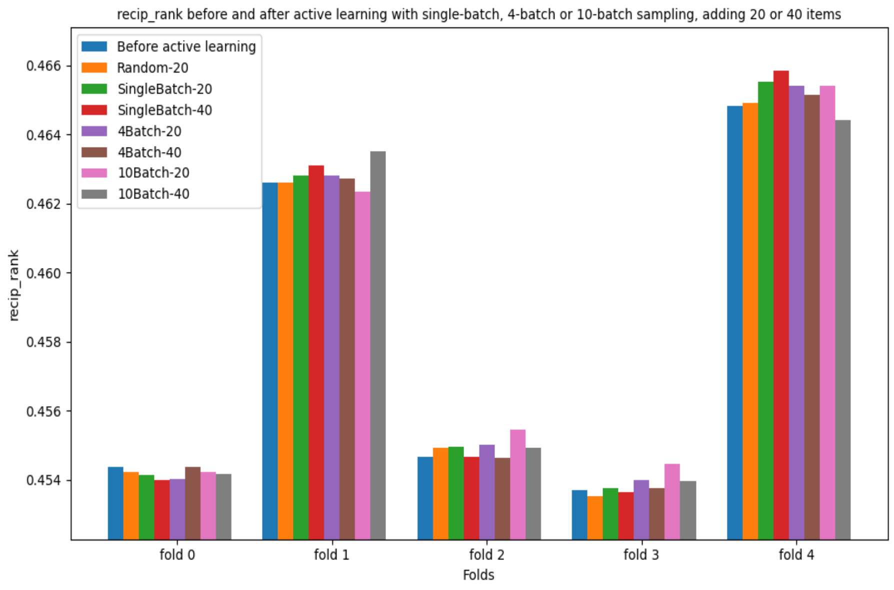


Figure 26: Recip_rank per fold for every sampling method with 20 or 40 items added

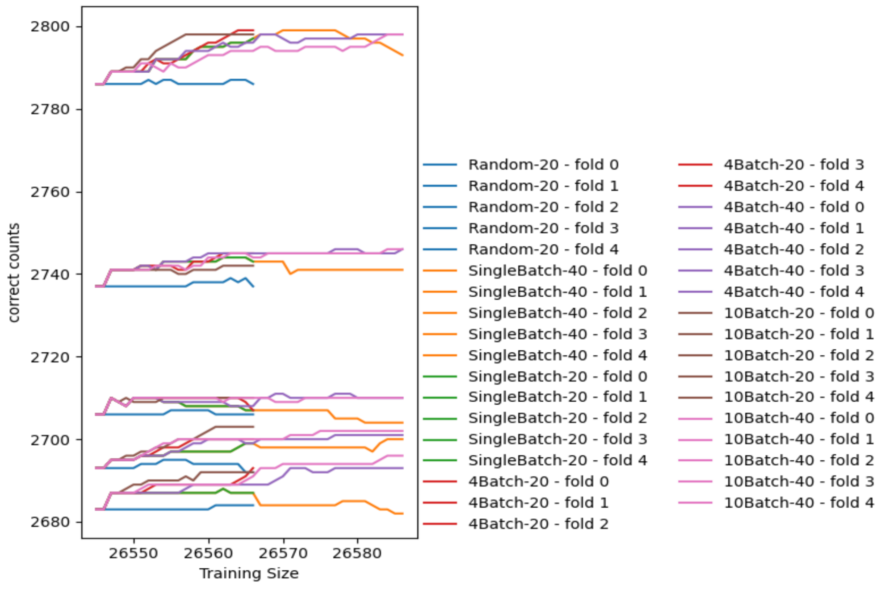


Figure 27: Learning curve count of correct recommendations per fold with all sampling methods, adding 20 or 40 items

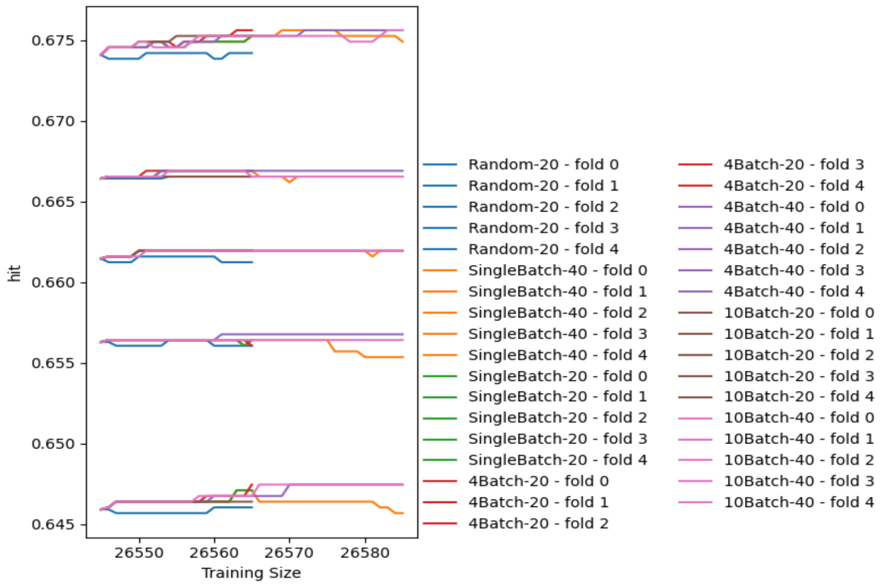


Figure 28: Learning curve hit per fold with all sampling methods, adding 20 or 40 items

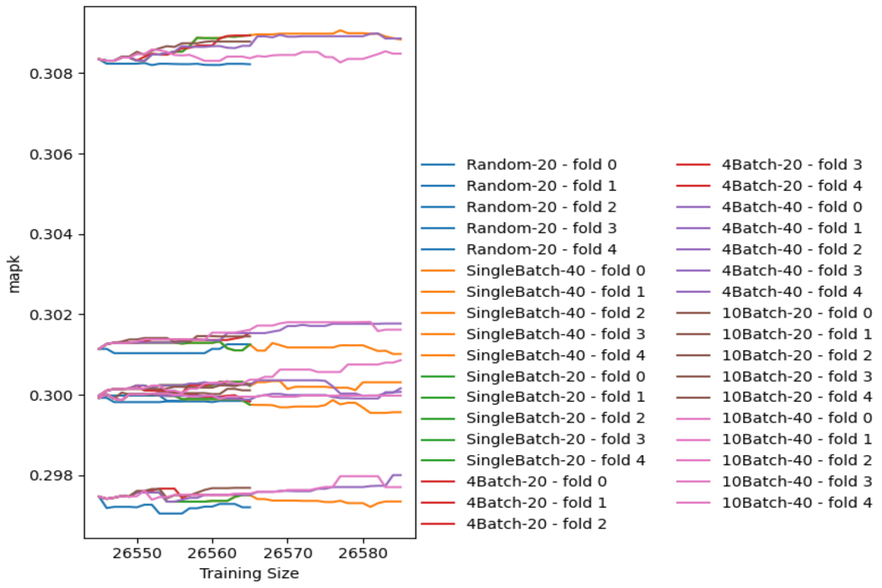


Figure 29: Learning curve MAP@K per fold with all sampling methods, adding 20 or 40 items

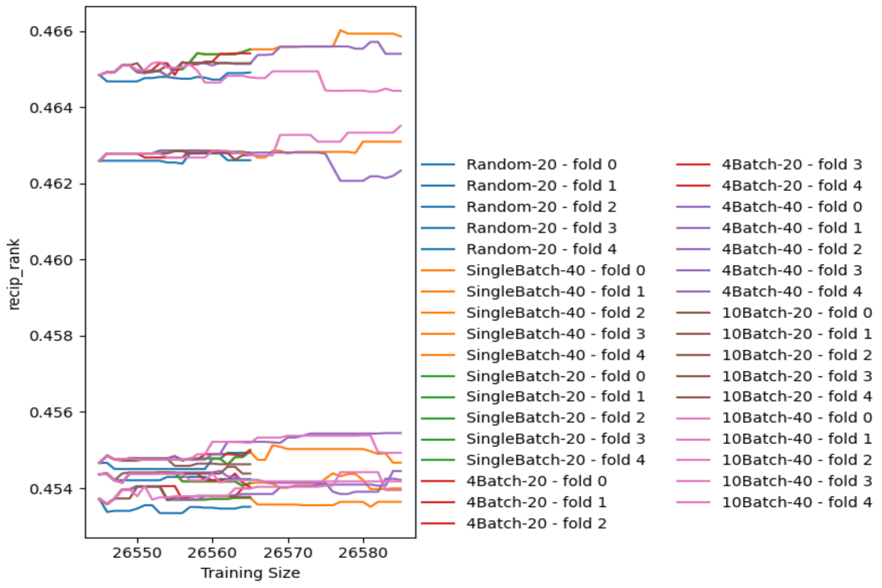


Figure 30: Learning curve recip_rank per fold with all sampling methods, adding 20 or 40 items

Table 13: Mean values of correct counts per sampling method

Algorithm	**Mean value**
Before active learning	2721.0
Random-20	2721.0
SingleBatch-40	2724.0
SingleBatch-20	2726.6
4Batch-20	2728.8
10Batch-20	2729.0
4Batch-40	2729.6
10Batch-40	2730.4

Table 14: Mean values of hit per sampling method

Algorithm	**Mean value**
Random-20	0.660819
Before active learning	0.660840
SingleBatch-40	0.660890
SingleBatch-20	0.661527
4Batch-20	0.661597
10Batch-20	0.661314
4Batch-40	0.661739
10Batch-40	0.661597

Table 15: Mean values of recip_rank per sampling method

Algorithm	**Mean value**
Before active learning	0.458035
Random-20	0.458038
10Batch-20	0.458134
10Batch-40	0.458198
SingleBatch-20	0.458231
4Batch-20	0.458247
SingleBatch-40	0.458251
4Batch-40	0.458369

Table 16: Mean values of MAP@K per sampling method

Algorithm	**Mean value**
Random-20	0.301296
Before active learning	0.301372
SingleBatch-40	0.301416
SingleBatch-20	0.301536
4Batch-20	0.301620
10Batch-20	0.301661
10Batch-40	0.301730
4Batch-40	0.301774

A.3 Single-batch and random sampling strategy adding 20 items

Three scenarios are made: Item-Based k-NN Collaborative Filtering algorithm with active learning using a specific sampling technique, Item-Based k-NN Collaborative Filtering algorithm with active learning using random sampling and just the Item-Based k-NN Collaborative Filtering algorithm. The performances of the three scenarios are compared using the evaluation metrics: count of correct predicted items, HIT, MAP@K, nDCG, recip_rank, and precision. The outcomes are depicted in Figures 31 till 36 .

Figures 37 till 42 represent the learning curves where the performance scores of every fold is plotted against every additional item for random and single-batch sampling adding 20 items.

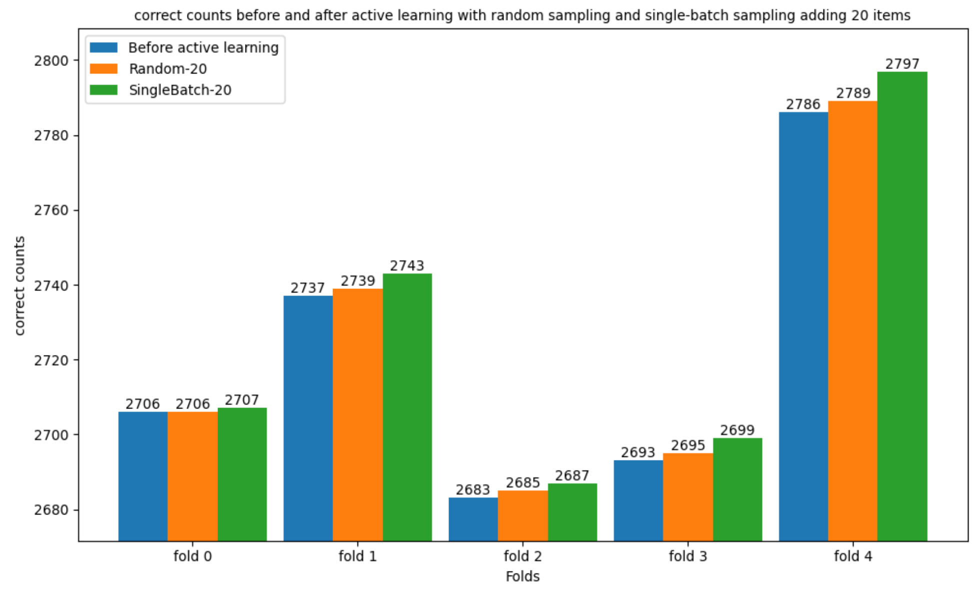


Figure 31: Count of correct recommendations per fold before and after active learning with random sampling and single-batch sampling, adding 20 items

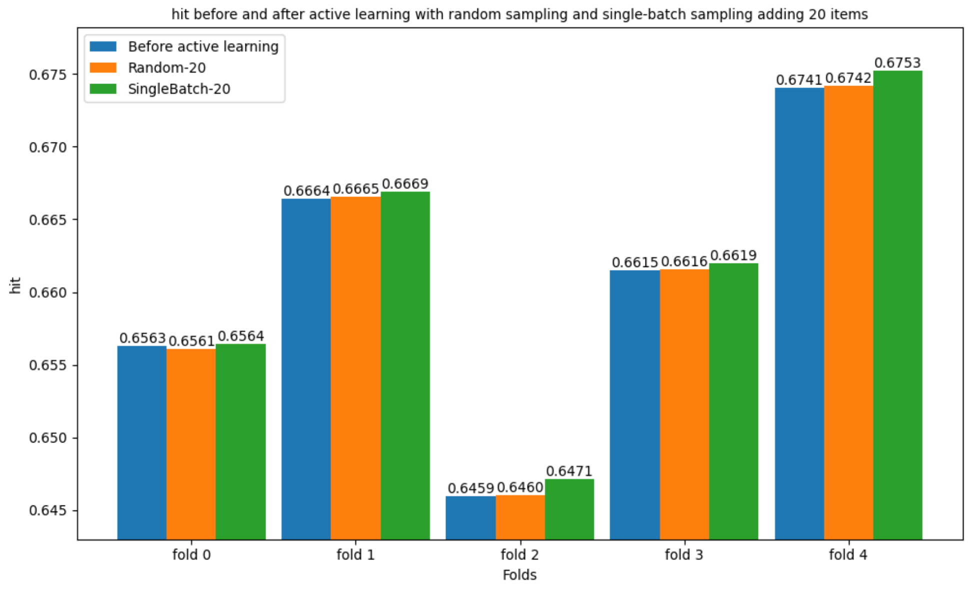


Figure 32: Hit per fold before and after active learning with random sampling and single-batch sampling, adding 20 items

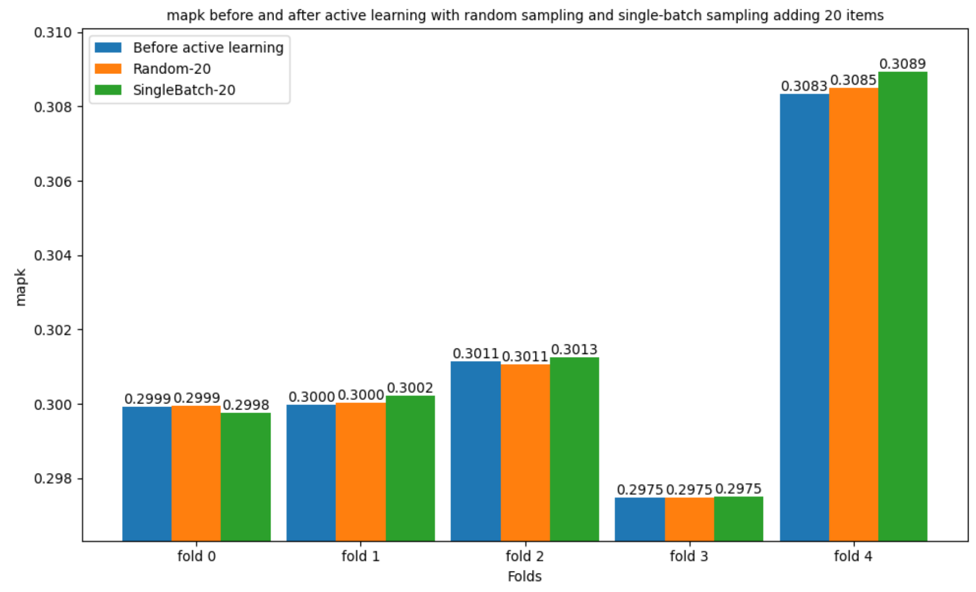


Figure 33: MAP@K per fold before and after active learning with random sampling and single-batch sampling, adding 20 items

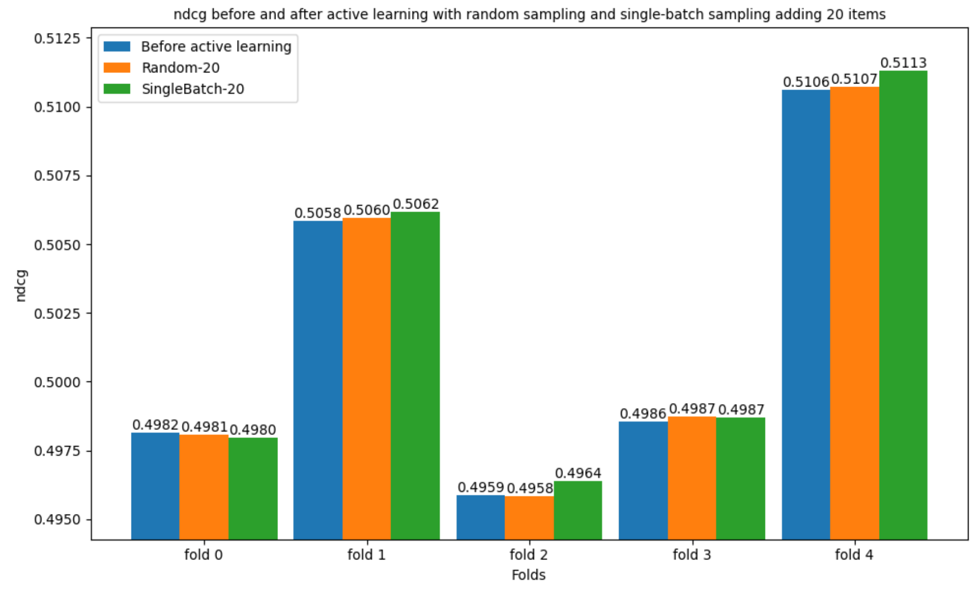


Figure 34: nDCG per fold before and after active learning with random sampling and single-batch sampling, adding 20 items

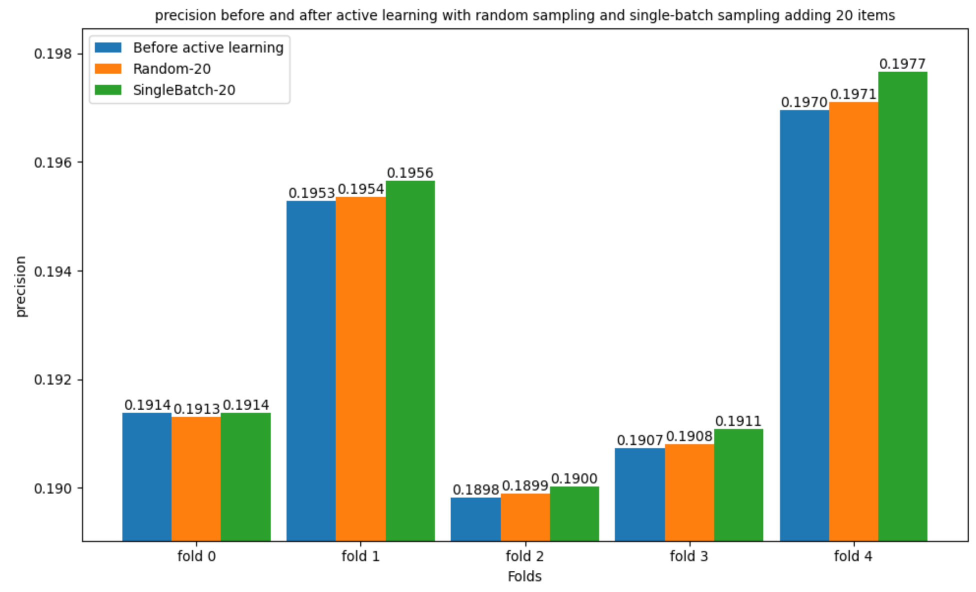


Figure 35: Precision per fold before and after active learning with random sampling and single-batch sampling, adding 20 items

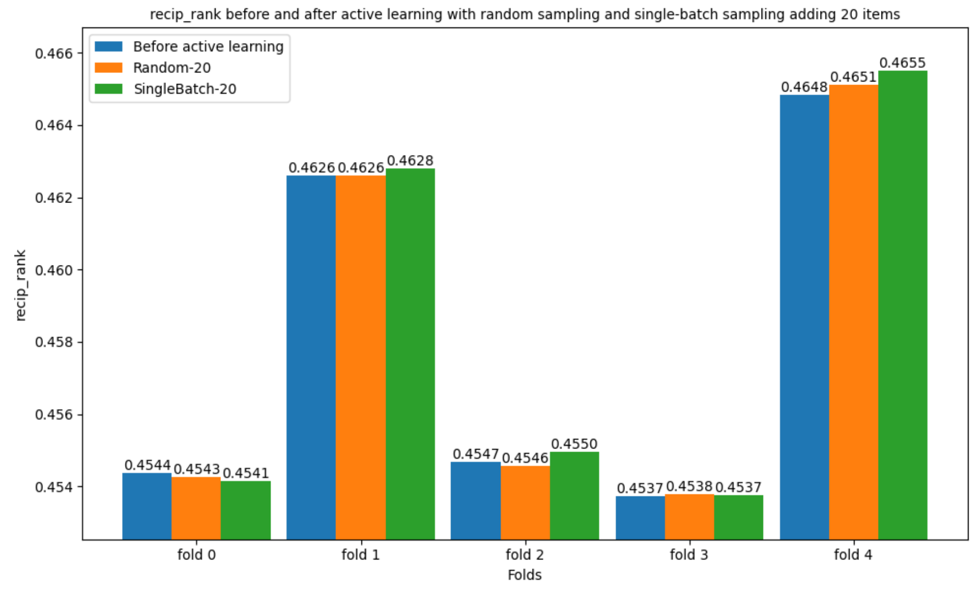


Figure 36: Recip_rank per fold before and after active learning with random sampling and single-batch sampling, adding 20 items

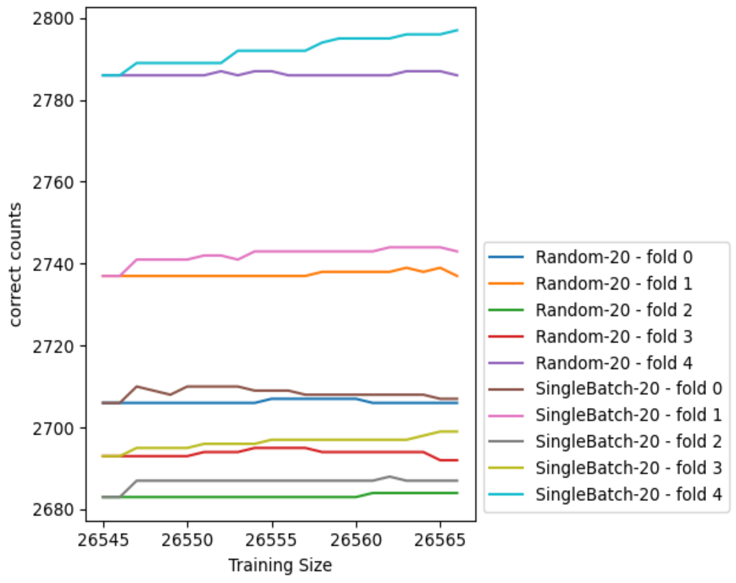


Figure 37: Learning curve count of correct recommendations per fold with random sampling and single-batch sampling, adding 20 items

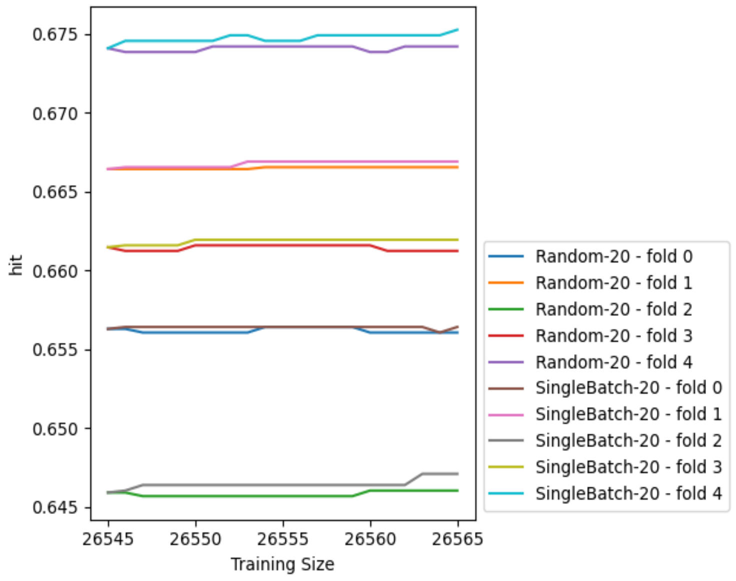


Figure 38: Learning curve hit per fold with random and sampling active learning

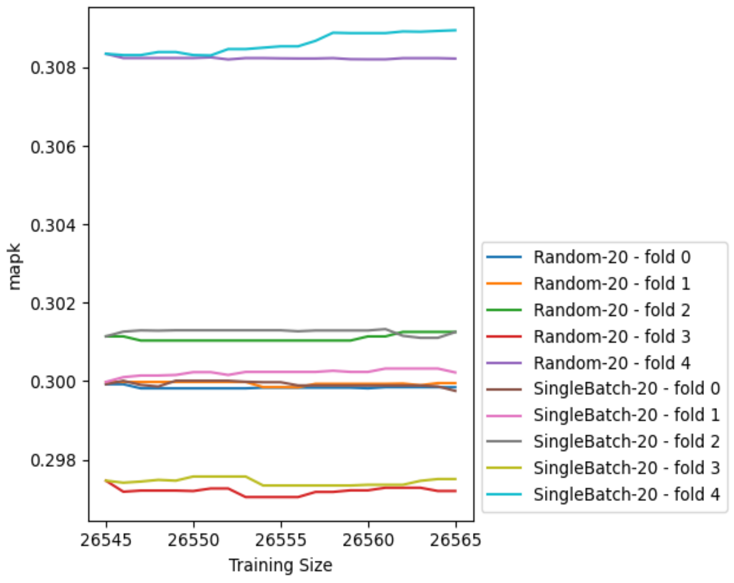


Figure 39: Learning curve MAP@K per fold with random sampling and single-batch sampling, adding 20 items

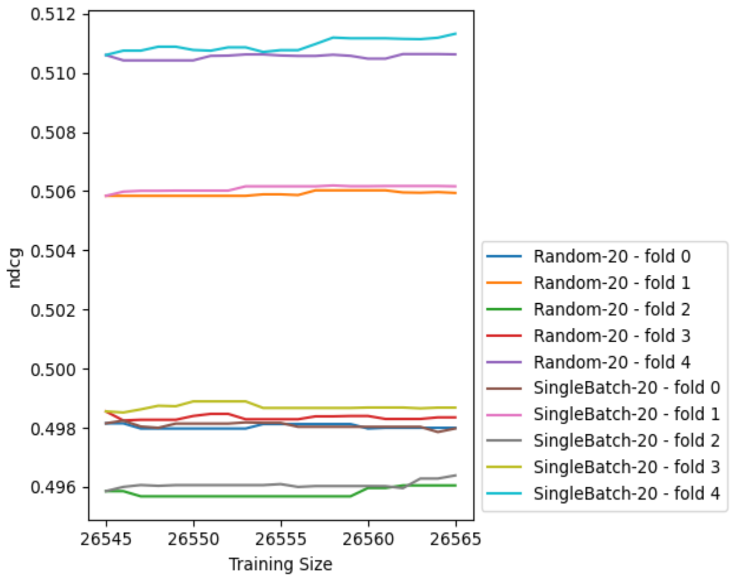


Figure 40: Learning curve nDCG per fold with random sampling and single-batch sampling, adding 20 items

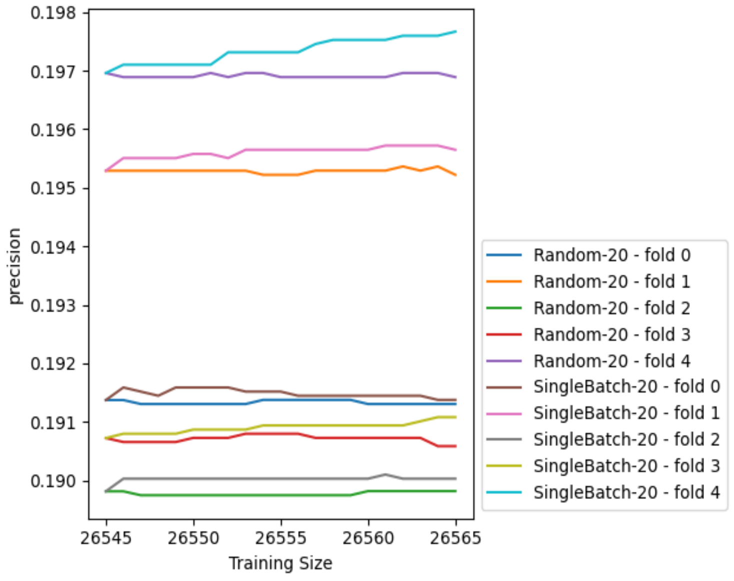


Figure 41: Learning curve precision per fold with random sampling and single-batch sampling, adding 20 items

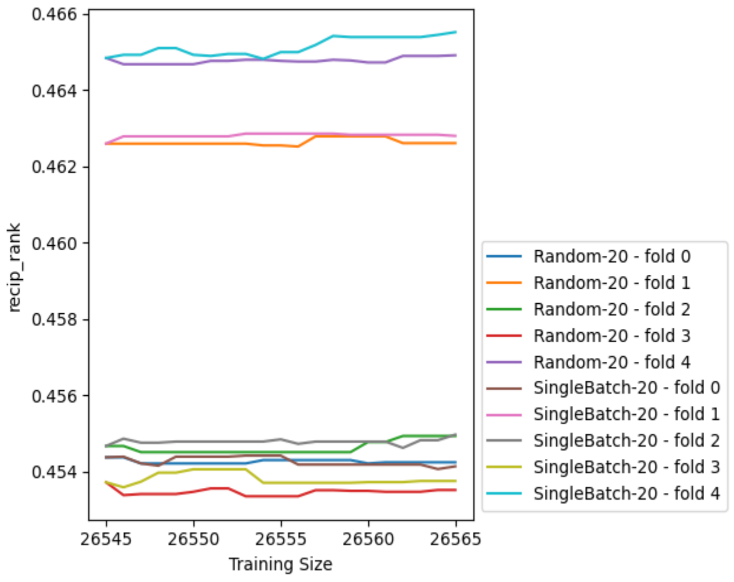


Figure 42: Learning curve recip_rank per fold with random sampling and single-batch sampling, adding 20 items

A.4 Single batch sampling strategy adding 20 or 40 items

Outcomes of the evaluation metrics: count of correct predicted items, HIT, MAP@K, nDCG, recip_rank, and precision for Item-Based k-NN Collaborative Filtering incorporating active learning with a single batch sampling strategy where 20 items or 40 items are added are depicted in Figures 43 till 48.

Figures 97 till 102 represent the learning curves where the performance score of every fold is plotted against every additional item for single batch active learning with 20 items added and single batch active learning with 40 items added.

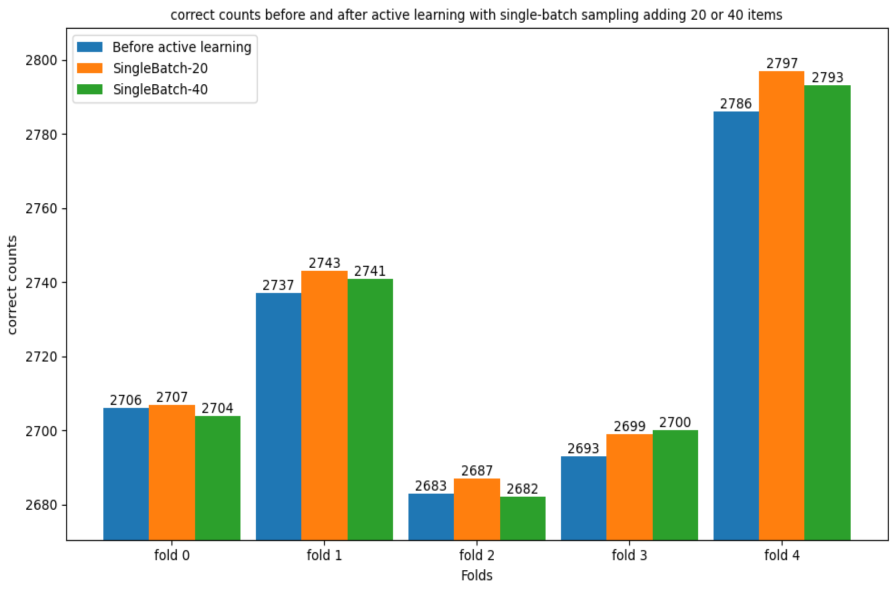


Figure 43: Count correct recommendations per fold for sampling active learning with 20 and 40 items added

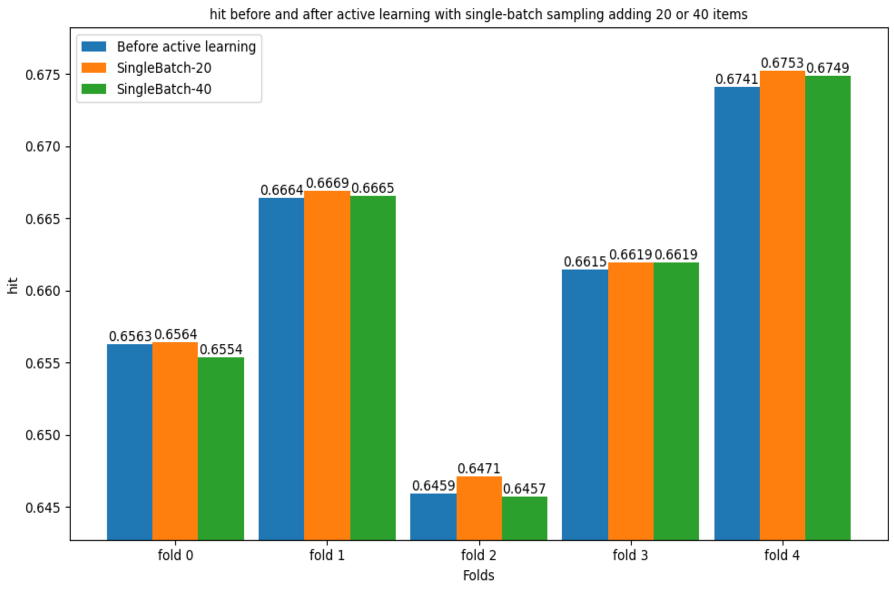


Figure 44: Hit per fold for sampling active learning with 20 and 40 items added

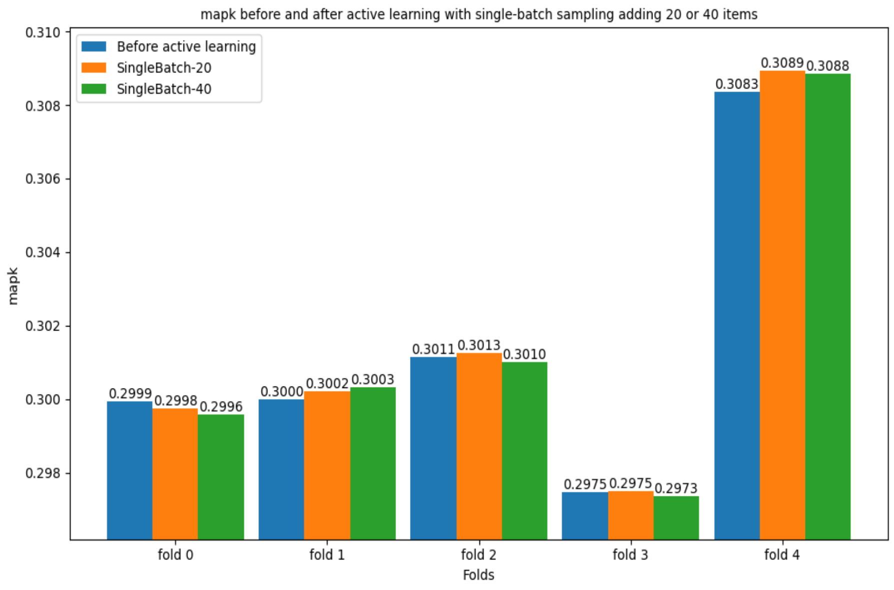


Figure 45: MAP@K per fold for sampling active learning with 20 and 40 items added

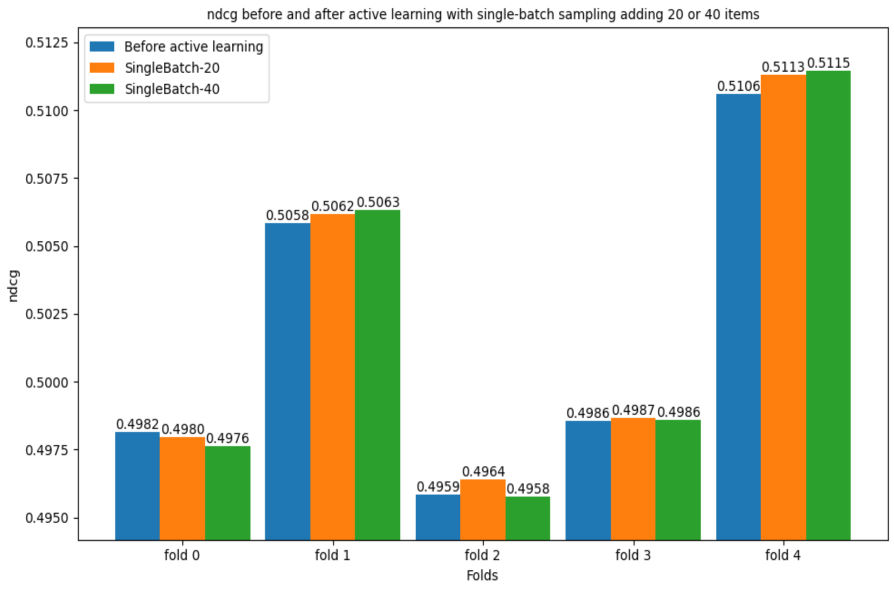


Figure 46: nDCG per fold for sampling active learning with 20 and 40 items added

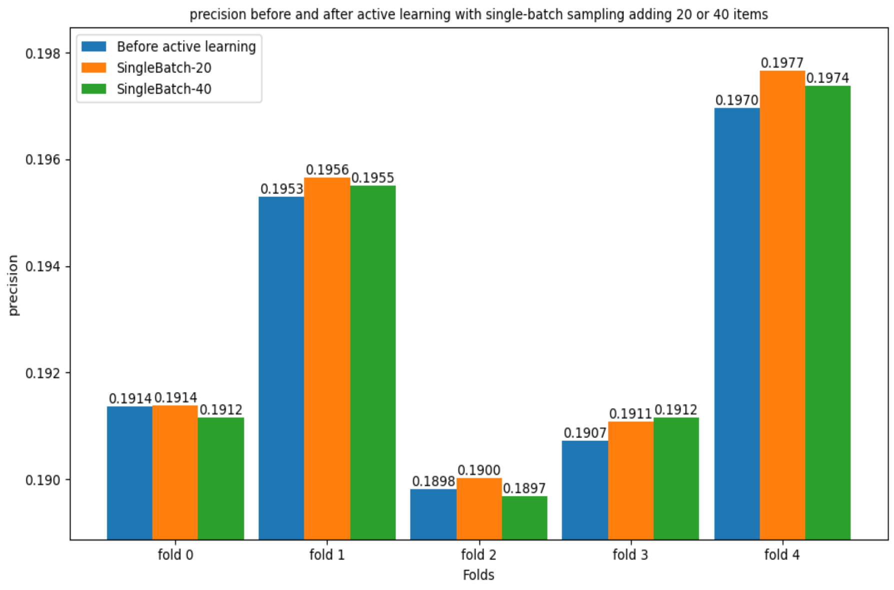


Figure 47: precision per fold for sampling active learning with 20 and 40 items added

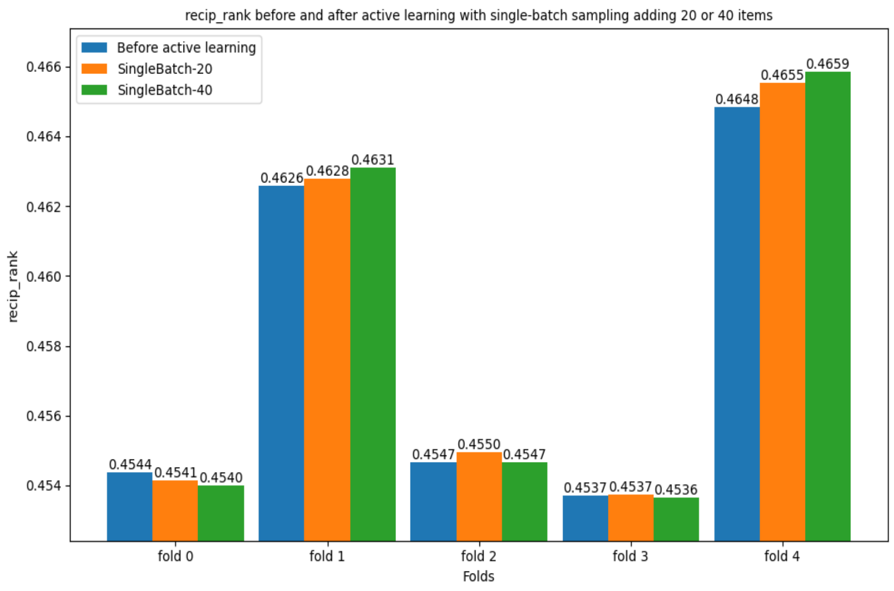


Figure 48: Recip_rank per fold for sampling active learning with 20 and 40 items added

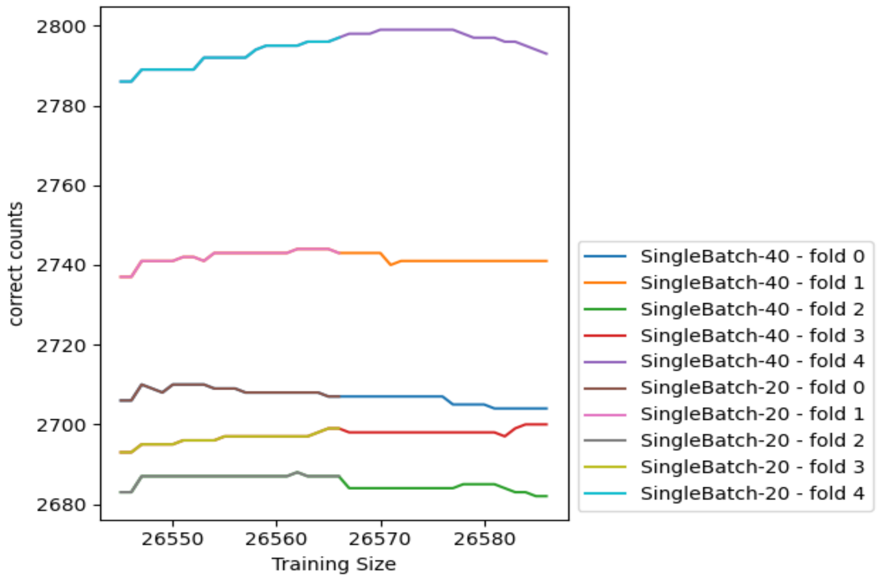


Figure 49: Learning curve count of correct recommendations per fold with single-batch sampling, adding 20 or 40 items

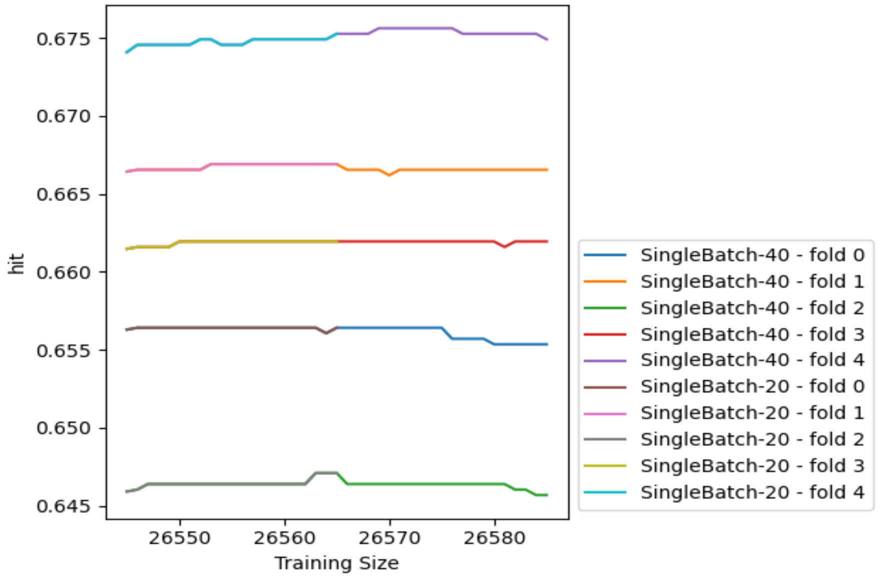


Figure 50: Learning curve hit per fold with single-batch sampling, adding 20 or 40 items

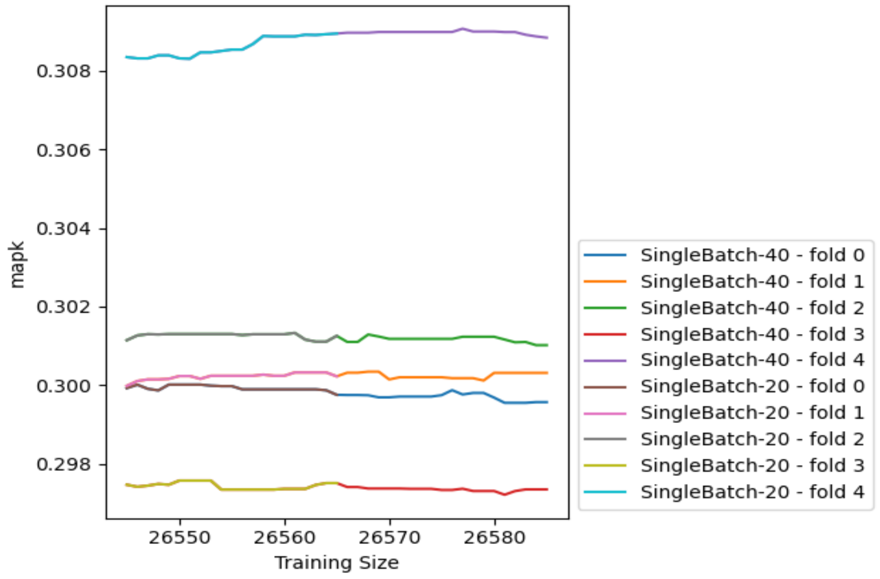


Figure 51: Learning curve MAP@K per fold with single-batch sampling, adding 20 or 40 items

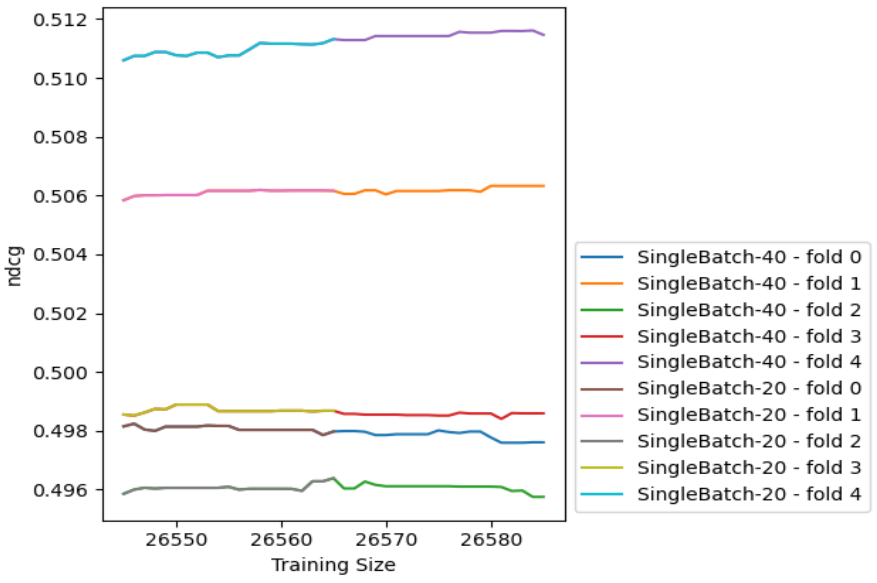


Figure 52: Learning curve nDCG per fold with single-batch sampling, adding 20 or 40 items

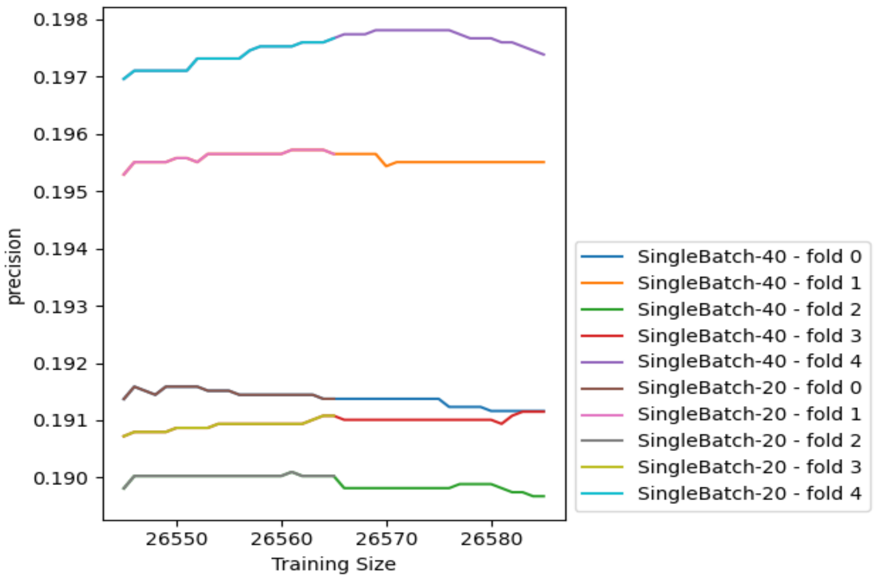


Figure 53: Learning curve precision per fold with single-batch sampling, adding 20 or 40 items

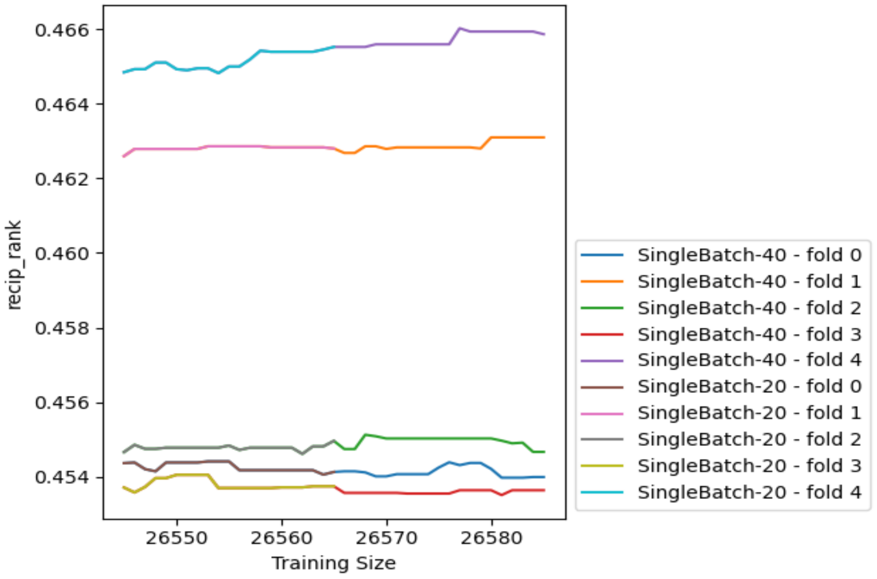


Figure 54: Learning curve recip_rank per fold with single-batch sampling, adding 20 or 40 items

A.5 4-Batch and single-batch sampling strategy adding 20 items

Outcomes of the evaluation metrics: count of correct predicted items, HIT, MAP@K, nDCG, recip_rank, and precision for Item-Based k-NN Collaborative Filtering are represented in Figures 55 till 60. These metrics are obtained by incorporating active learning with a single batch sampling strategy and a 4-batch sampling strategy where the strategy is updated 4 times. For both strategies, 20 items in total are added. In the 4-batch sampling strategy, 5 items are added per batch.

Figures 61 till 66 represent the learning curves where the performance scores of every fold is plotted against every additional item for active learning with single batch sampling strategy and 4 batches sampling strategy.

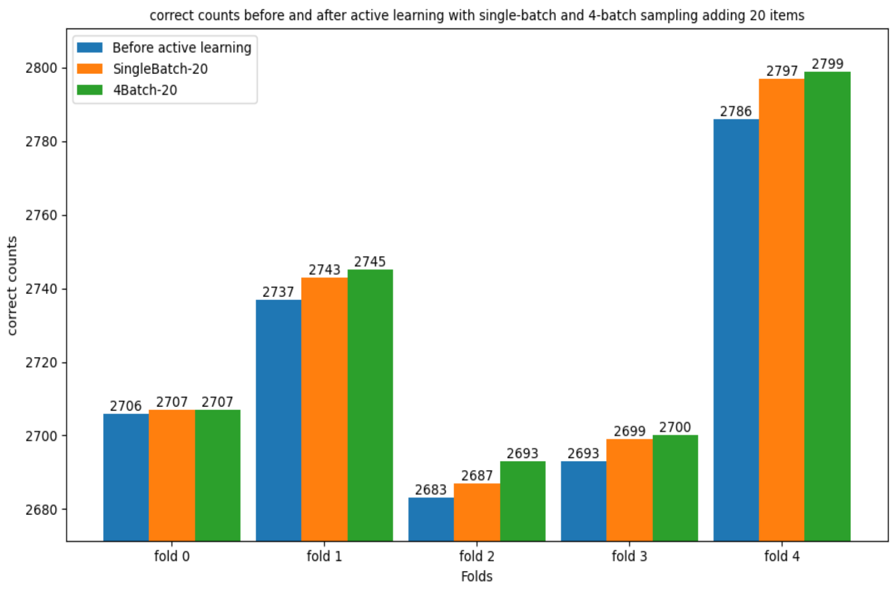


Figure 55: Count of correct recommendations per fold before and after active learning with single-batch and 4-batch sampling, adding 20 items

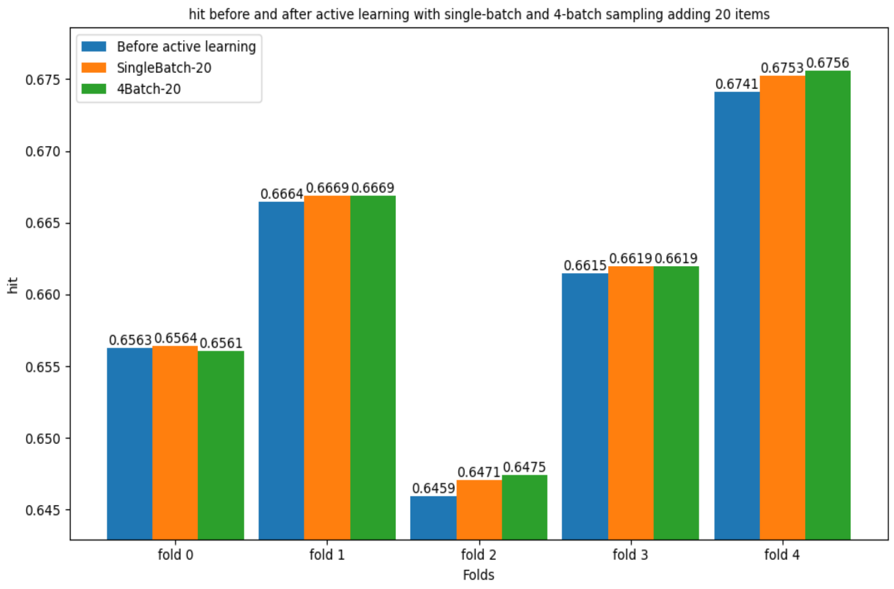


Figure 56: Hit per fold before and after active learning with single-batch and 4-batch sampling, adding 20 items

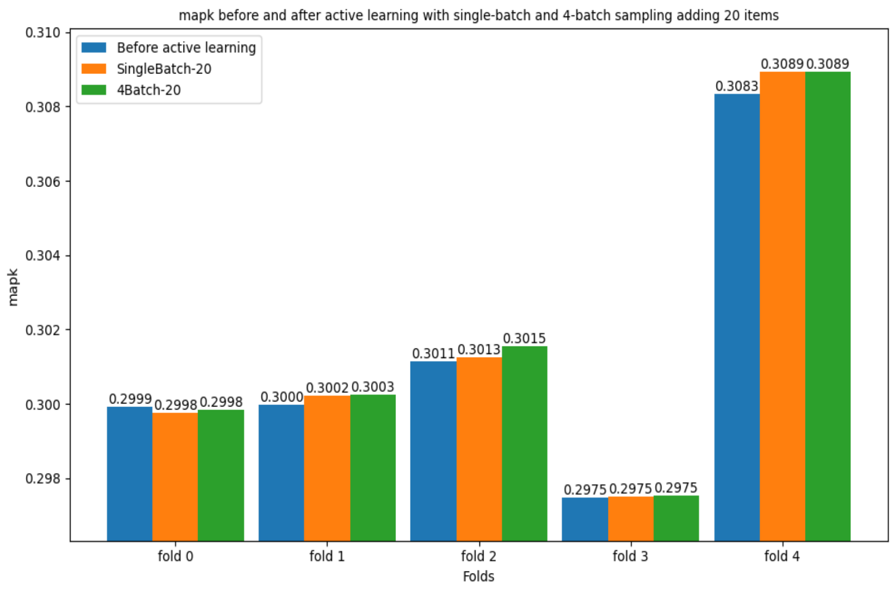


Figure 57: MAP@K per fold before and after active learning with single-batch and 4-batch sampling, adding 20 items

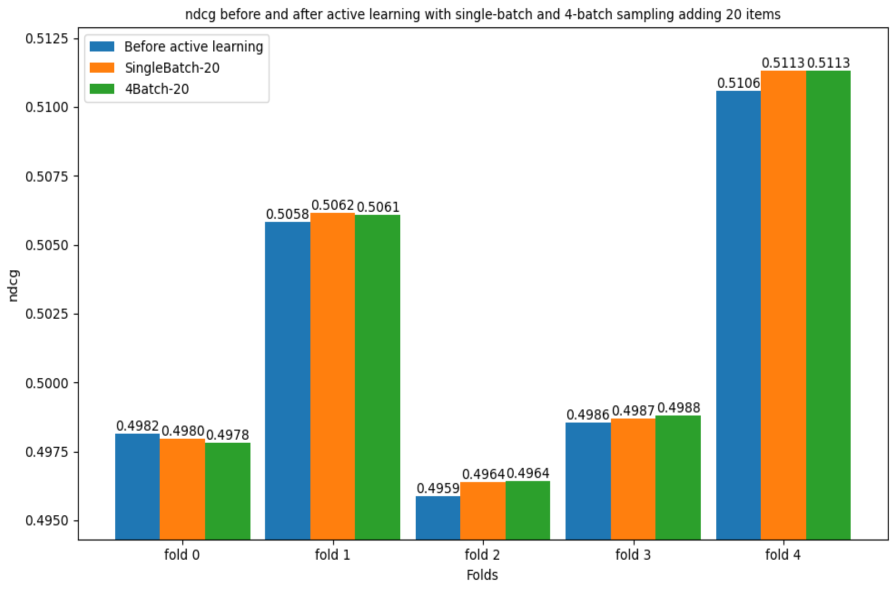


Figure 58: nDCG per fold before and after active learning with single-batch and 4-batch sampling, adding 20 items

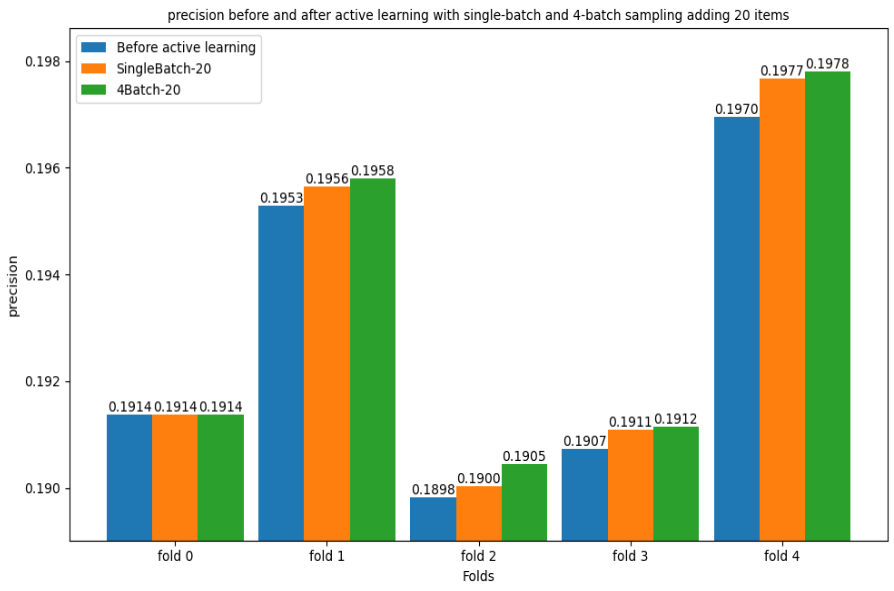


Figure 59: Precision per fold before and after active learning with single-batch and 4-batch sampling, adding 20 items

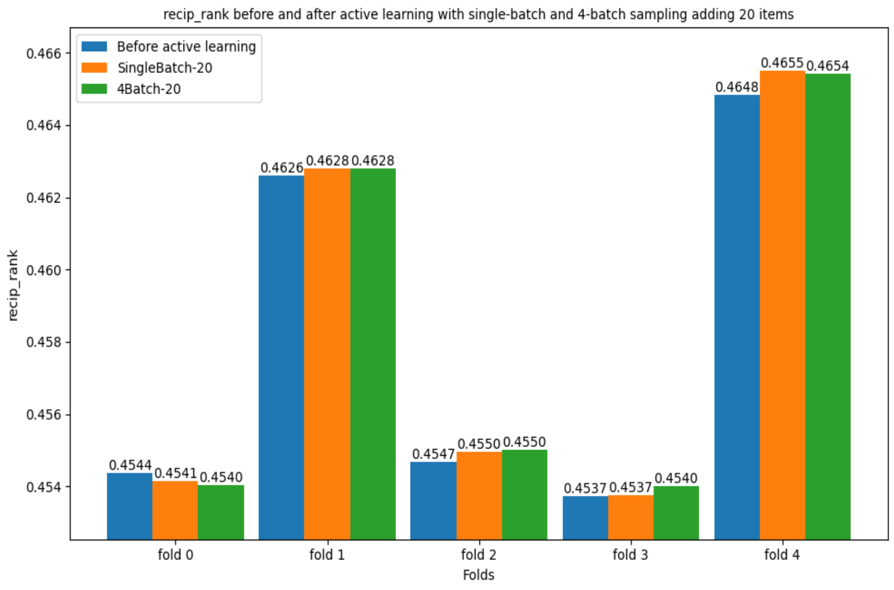


Figure 60: Recip_rank per fold before and after active learning with single-batch and 4-batch sampling, adding 20 items

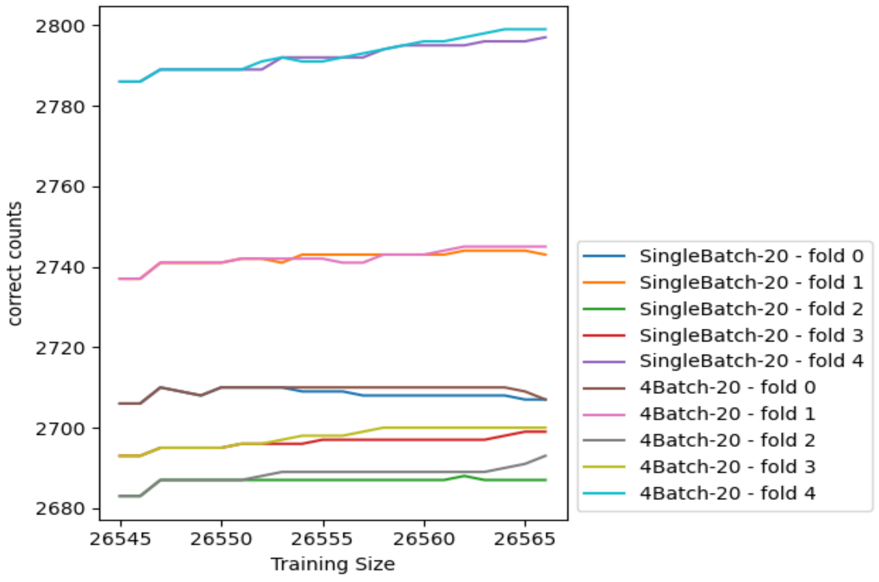


Figure 61: Learning curve count of correct recommendations per fold with single-batch and 4-batch sampling, adding 20 items

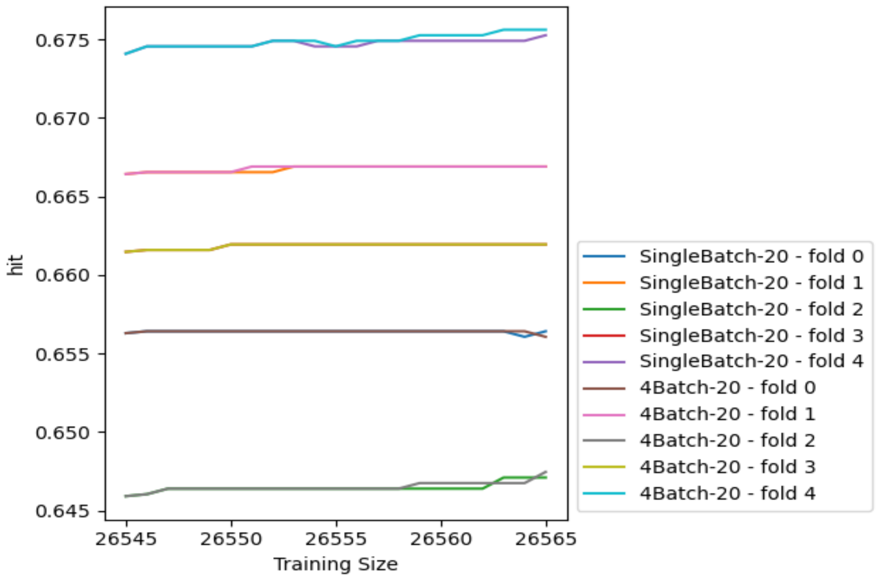


Figure 62: Learning curve hit per fold with single-batch and 4-batch sampling, adding 20 items

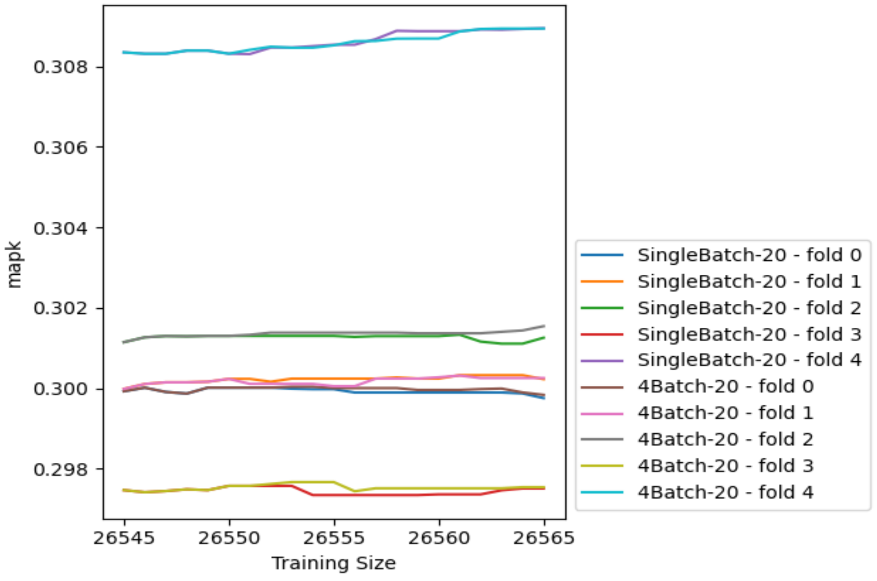


Figure 63: Learning curve MAP@K per fold with single-batch and 4-batch sampling, adding 20 items

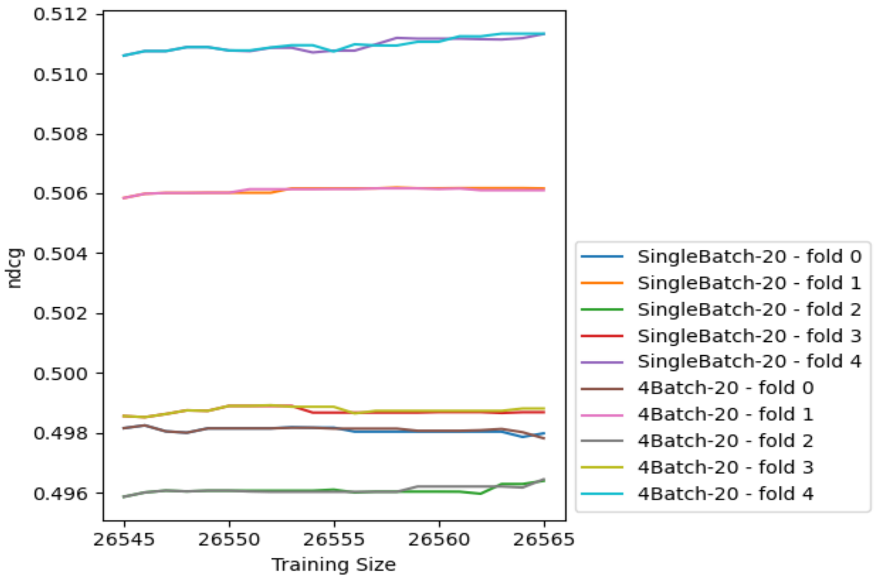


Figure 64: Learning curve nDCG per fold with single-batch and 4-batch sampling, adding 20 items

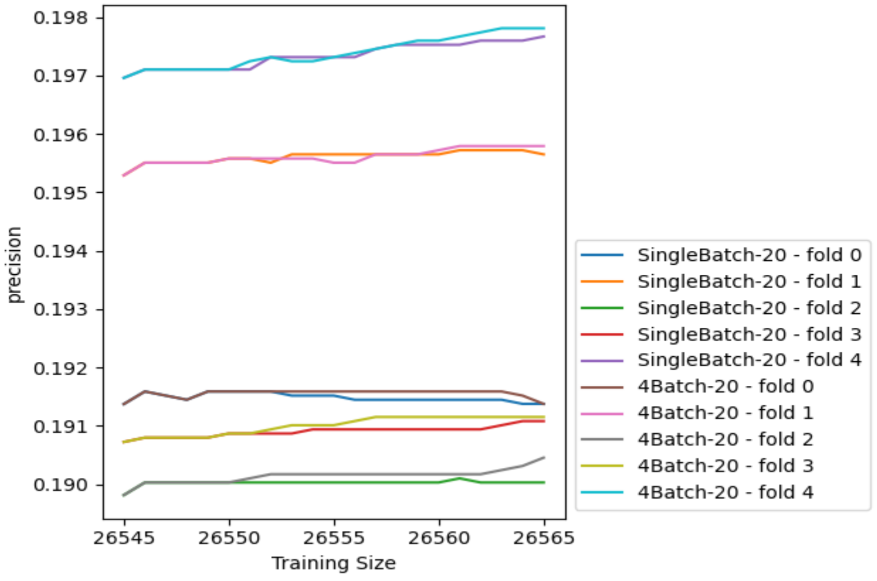


Figure 65: Learning curve precision per fold with single-batch and 4-batch sampling, adding 20 items

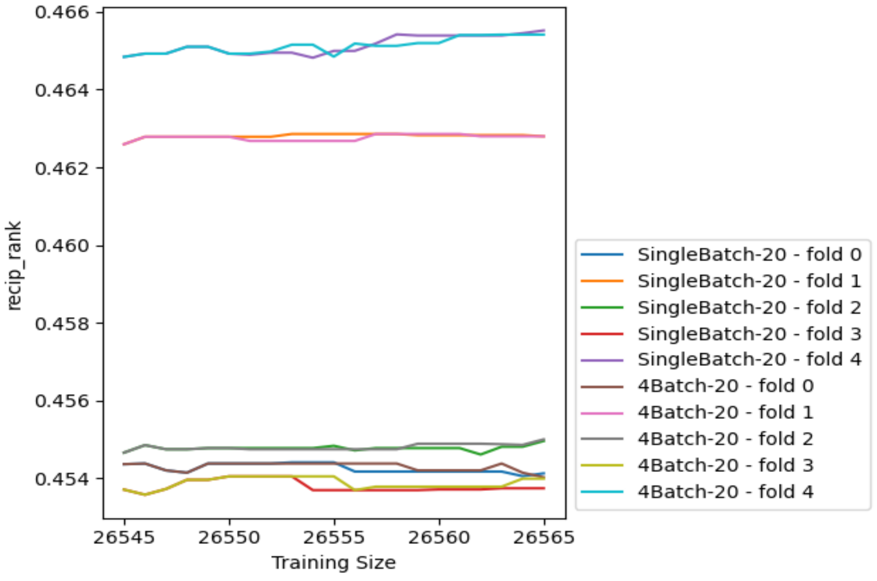


Figure 66: Learning curve recip_rank per fold with single-batch and 4-batch sampling, adding 20 items

A.6 10-batch and single-batch sampling strategy adding 20 items

Outcomes of the evaluation metrics: count of correct predicted items, HIT, MAP@K, nDCG, recip_rank, and precision for Item-Based k-NN Collaborative Filtering are represented in Figures 67 till 72. These metrics are obtained by incorporating active learning with a single batch sampling strategy and a 10-batch sampling strategy where the strategy is updated 10 times. For both strategies, 20 items in total are added. In the 10-batch sampling strategy, 2 items are added per batch.

Figures 73 till 78 represent the learning curves where the performance scores of every fold is plotted against every additional item for active learning with single batch sampling strategy and 10-batch sampling strategy.

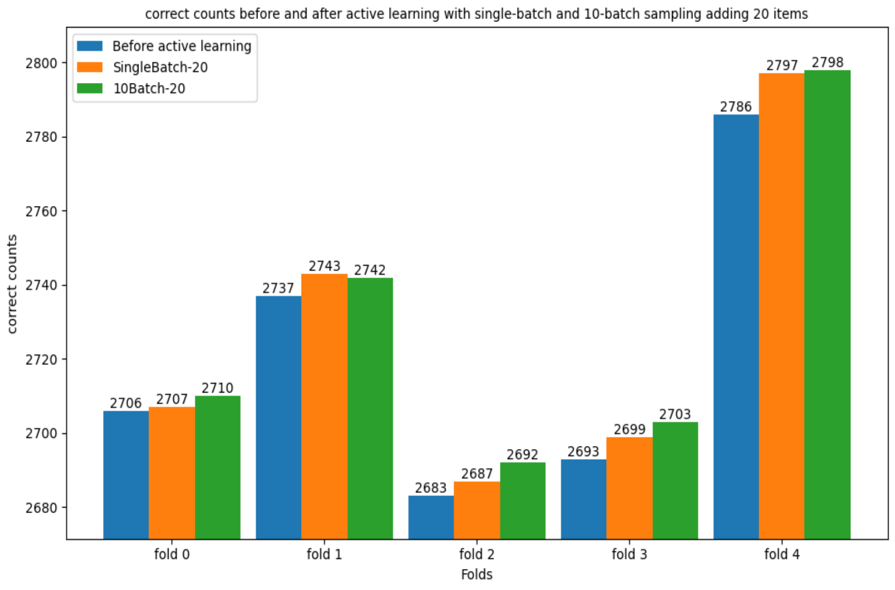


Figure 67: Count of correct recommendations per fold before and after active learning with single-batch and 10-batch sampling, adding 20 items

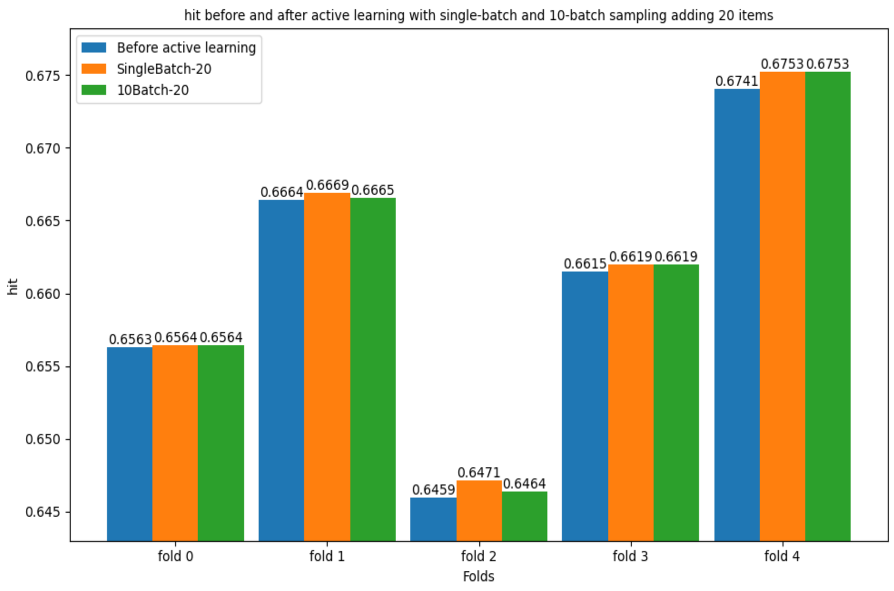


Figure 68: Hit per fold before and after active learning with single-batch and 10-batch sampling, adding 20 items

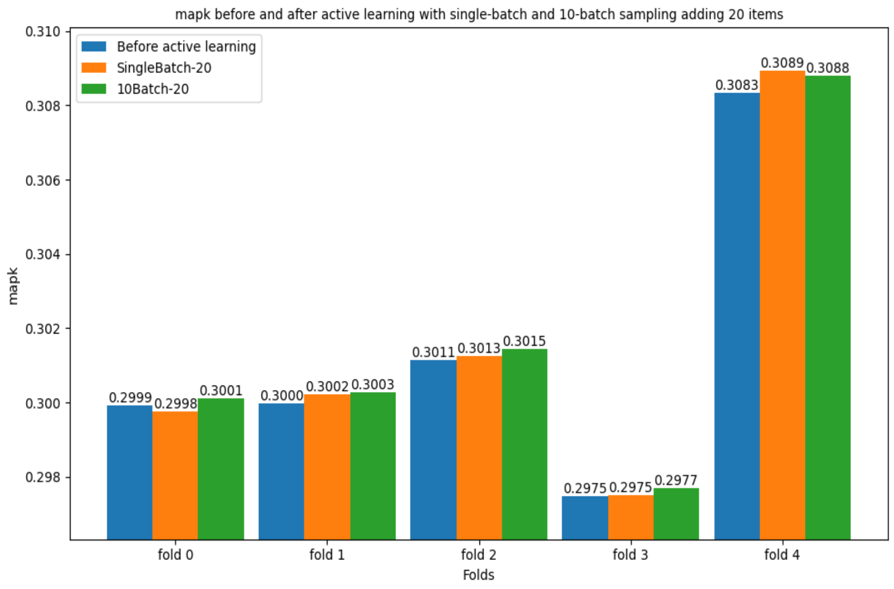


Figure 69: MAP@K per fold before and after active learning with single-batch and 10-batch sampling, adding 20 items

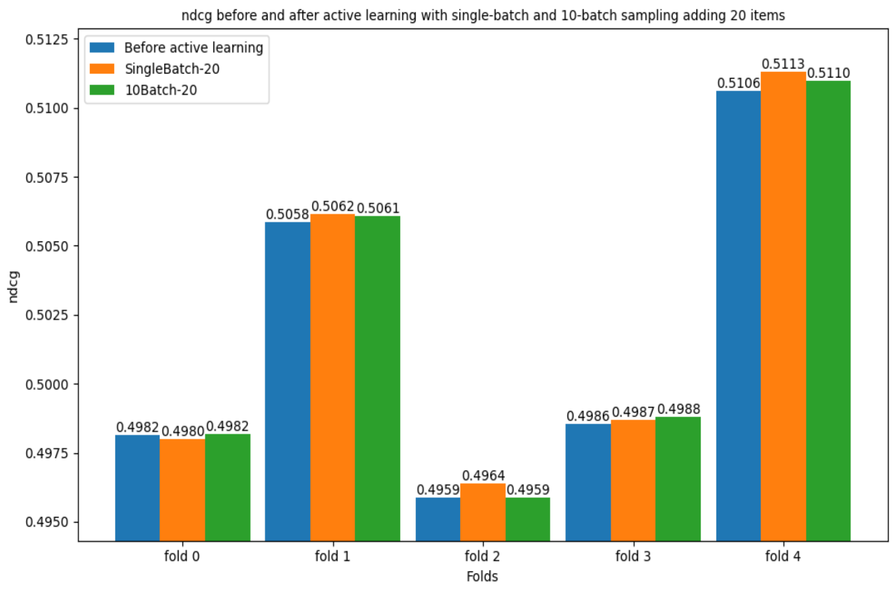


Figure 70: nDCG per fold before and after active learning with single-batch and 10-batch sampling, adding 20 items

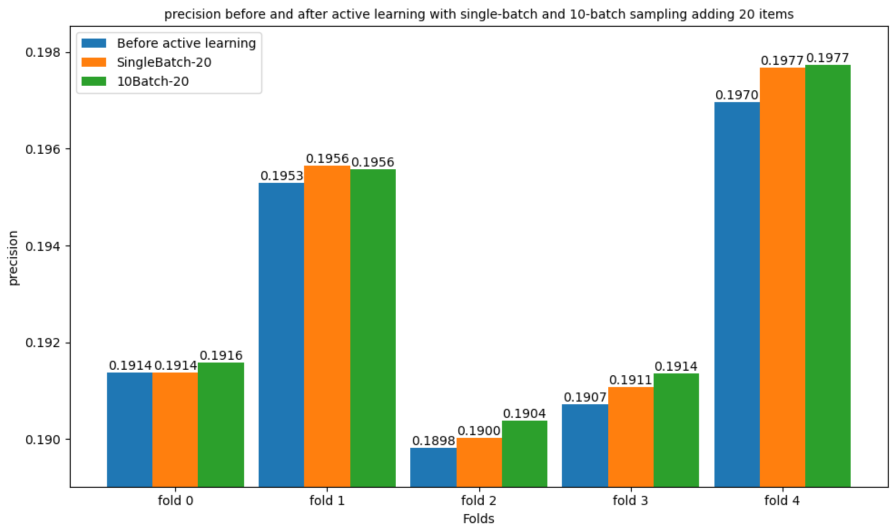


Figure 71: Precision per fold before and after active learning with single-batch and 10-batch sampling, adding 20 items

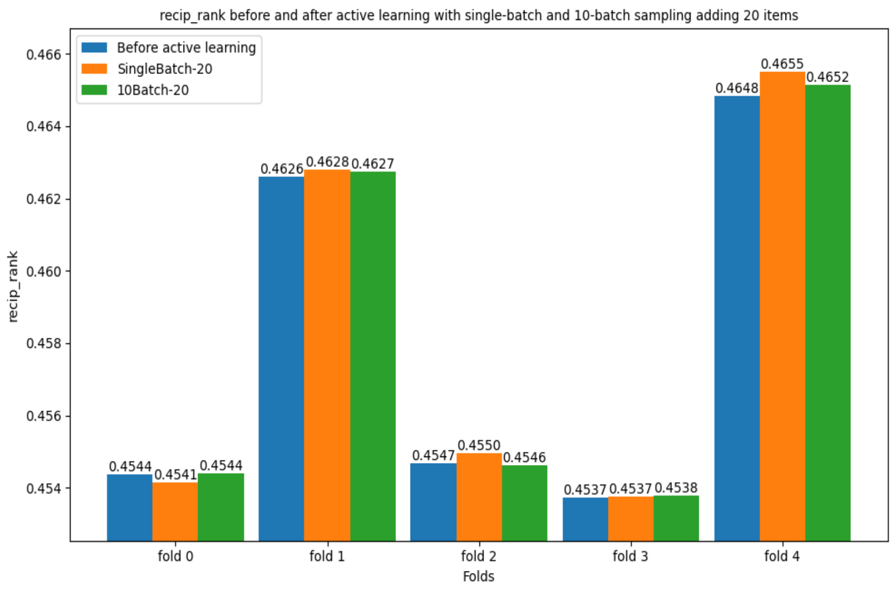


Figure 72: Recip_rank per fold before and after active learning with single-batch and 10-batch sampling, adding 20 items

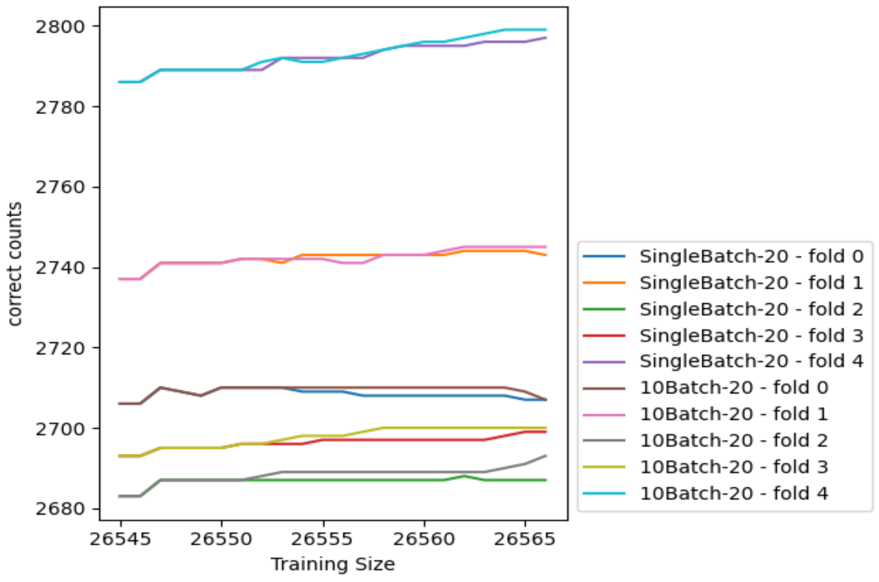


Figure 73: Learning curve count of correct recommendations per fold with single-batch and 10-batch sampling, adding 20 items

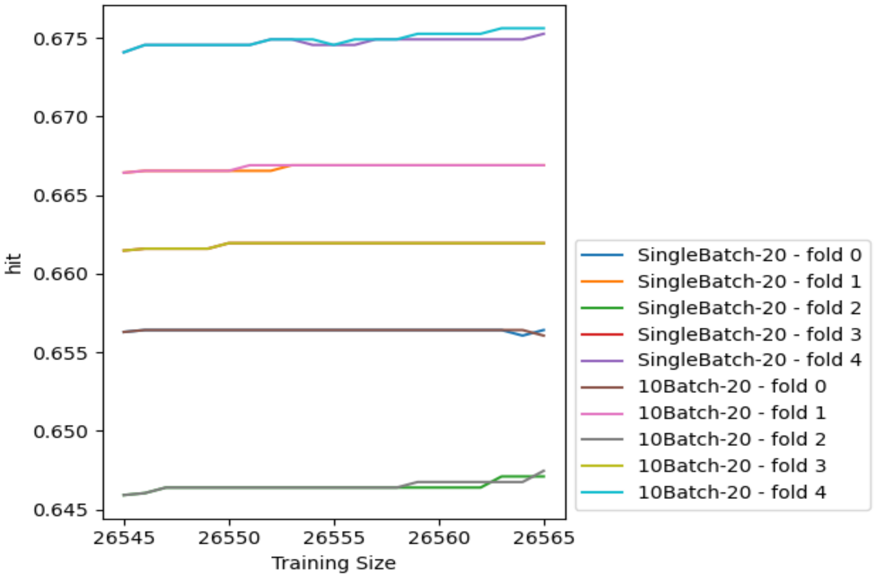


Figure 74: Learning curve hit per fold with single-batch and 10-batch sampling, adding 20 items

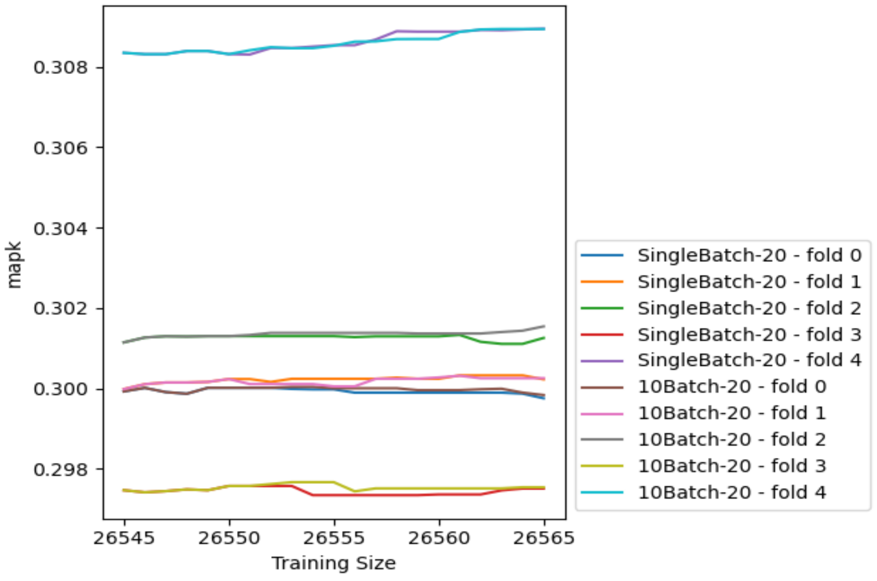


Figure 75: Learning curve MAP@K per fold with single-batch and 10-batch sampling, adding 20 items

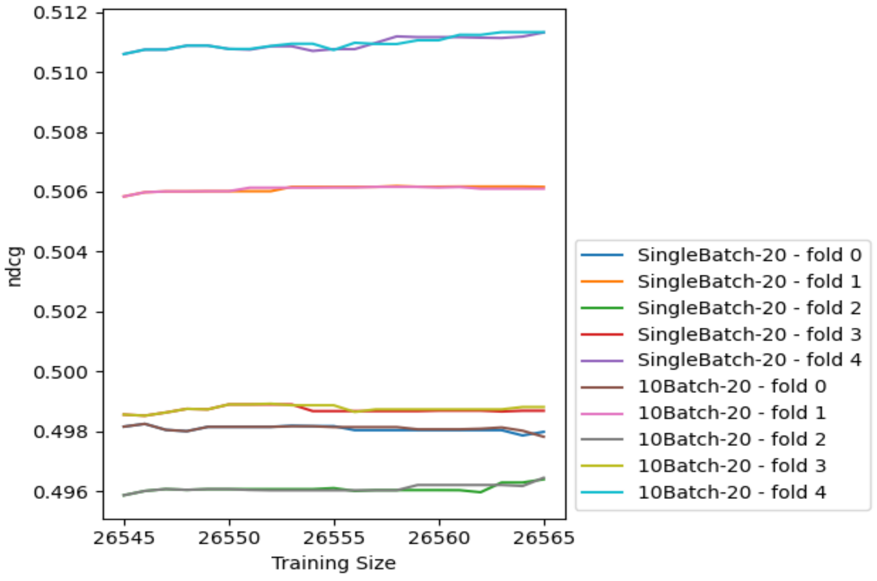


Figure 76: Learning curve nDCG per fold with single-batch and 10-batch sampling, adding 20 items

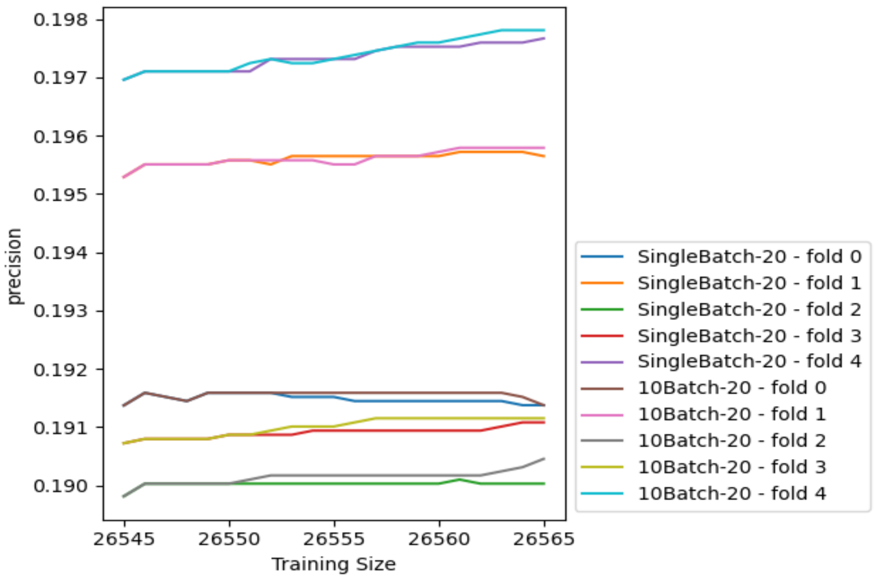


Figure 77: Learning curve precision per fold with single-batch and 10-batch sampling, adding 20 items

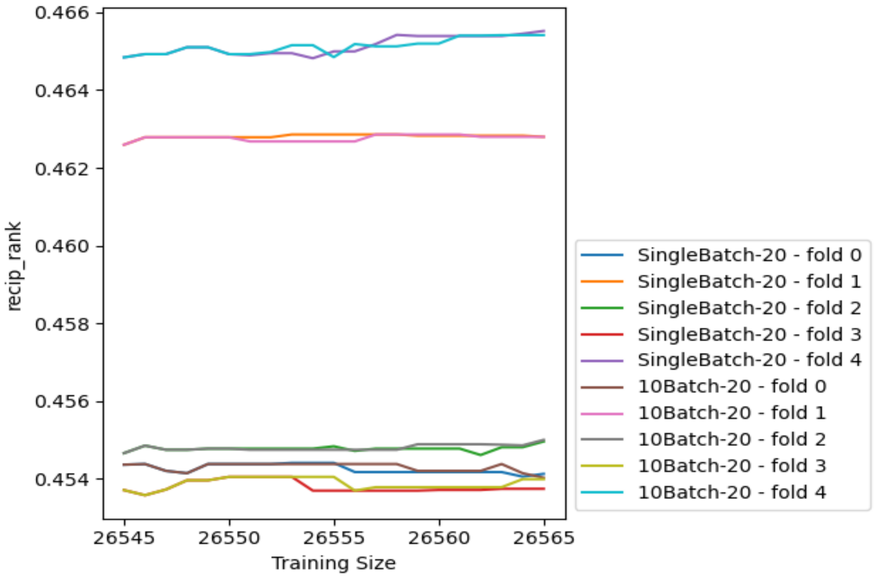


Figure 78: Learning curve recip_rank per fold with single-batch and 10-batch sampling, adding 20 items

A.7 4-batch and single-batch sampling strategy adding 40 items

Outcomes of the evaluation metrics: count of correct predicted items, HIT, MAP@K, nDCG, recip_rank, and precision for Item-Based k-NN Collaborative Filtering are represented in Figures 79 till 84. These metrics are obtained by incorporating active learning with a single batch sampling strategy and a 4-batch sampling strategy where the strategy is updated 10 times. For both strategies, 40 items in total are added. In the 4-batch sampling strategy, 10 items are added per batch.

Figures 97 till 90 represent the learning curves where the performance scores of every fold is plotted against every additional item for active learning with single batch sampling strategy and 10-batch sampling strategy.

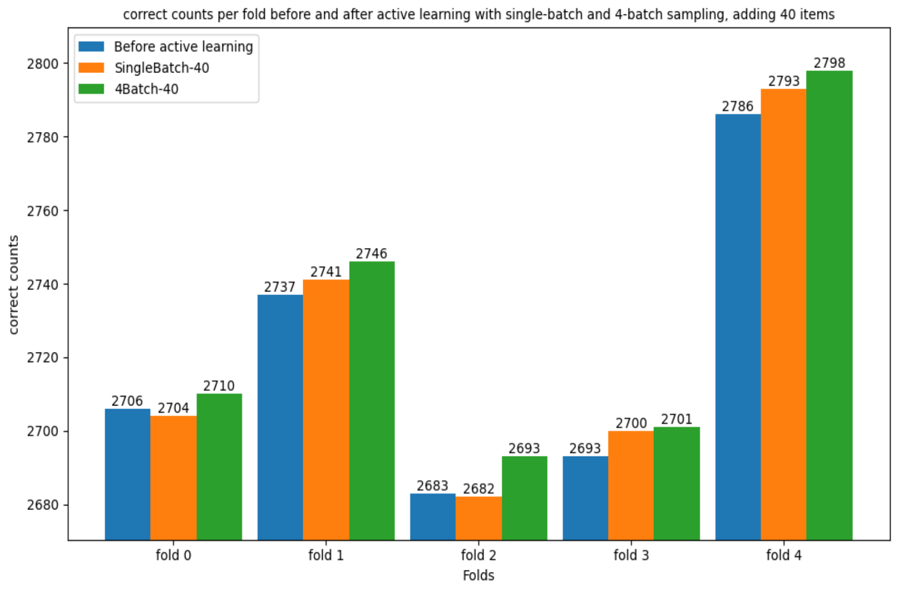


Figure 79: Count of correct recommendations per fold before and after active learning with single-batch and 4-batch, adding 40 items

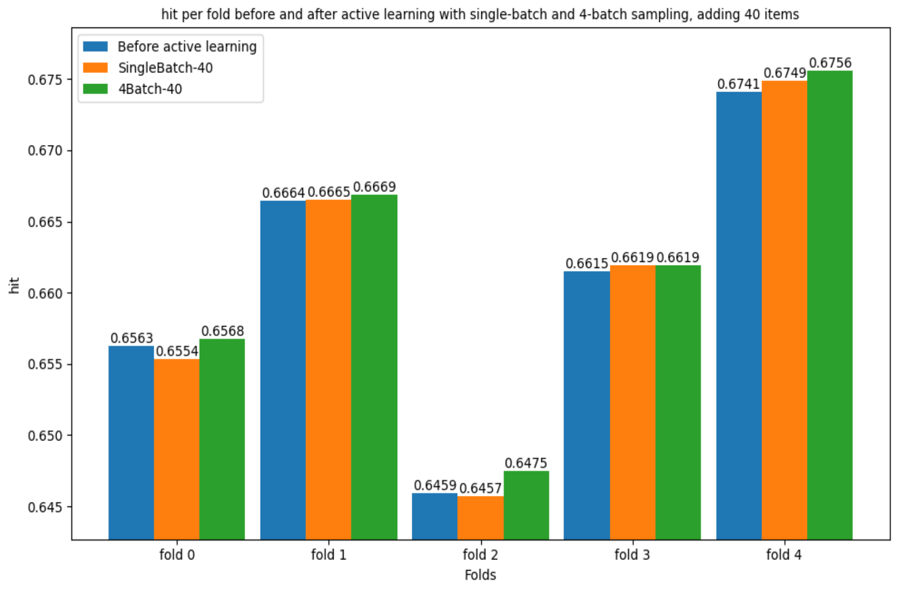


Figure 80: Hit per fold before and after active learning with single-batch and 4-batch, adding 40 items

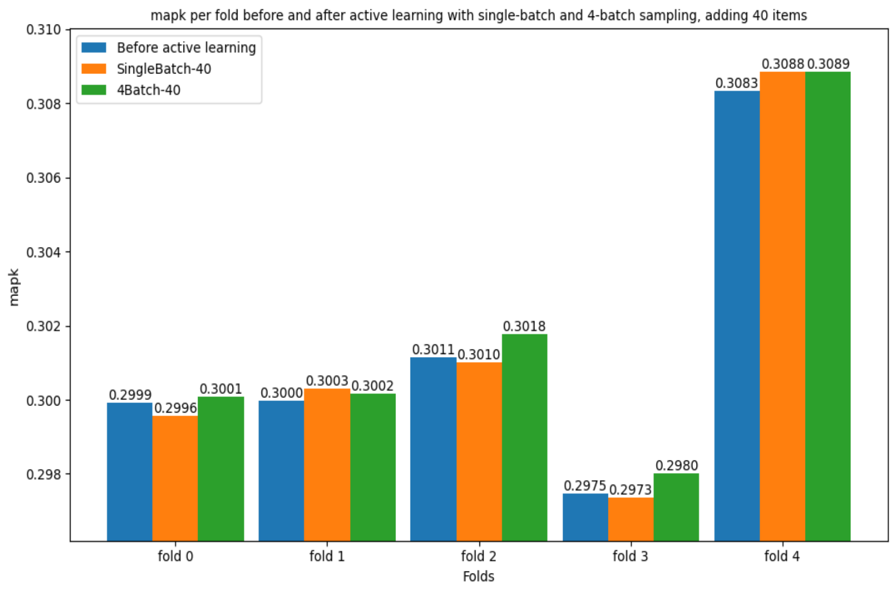


Figure 81: MAP@K per fold before and after active learning with single-batch and 4-batch, adding 40 items

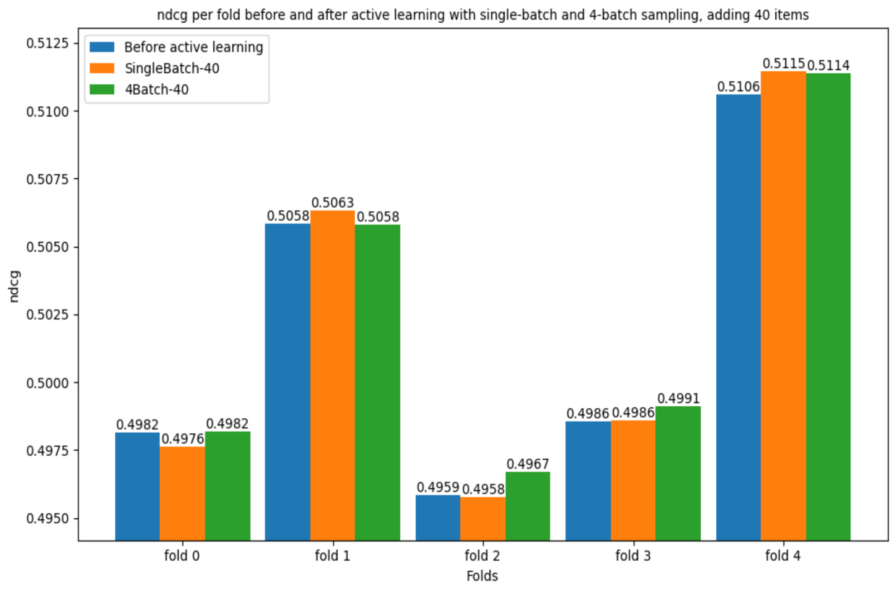


Figure 82: nDCG per fold before and after active learning with single-batch and 4-batch, adding 40 items

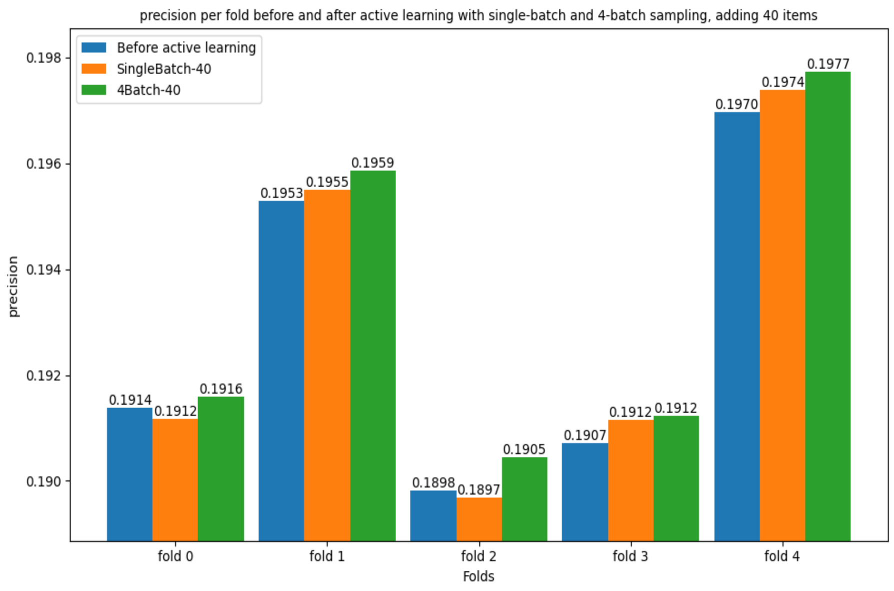


Figure 83: Precision per fold before and after active learning with single-batch and 4-batch, adding 40 items

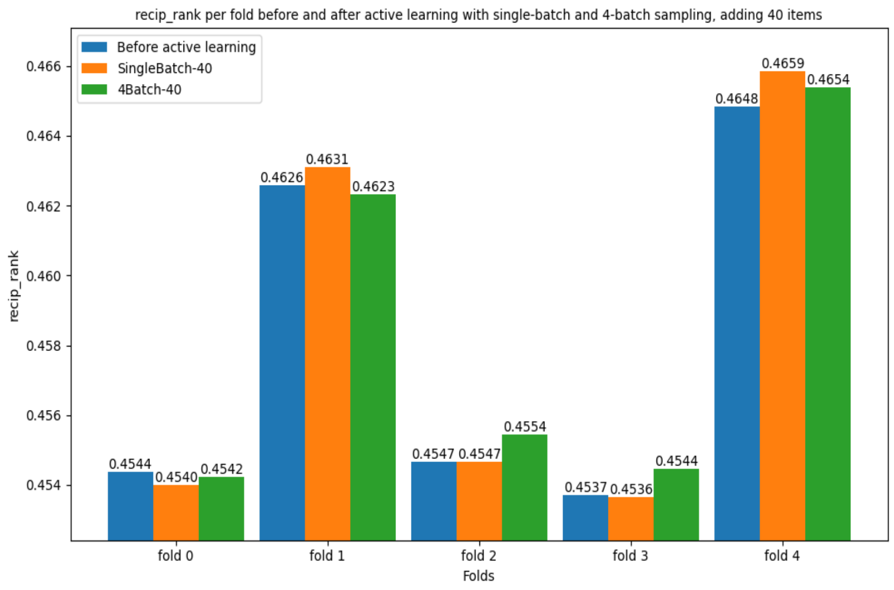


Figure 84: Recip_rank per fold before and after active learning with single-batch and 4-batch, adding 40 items

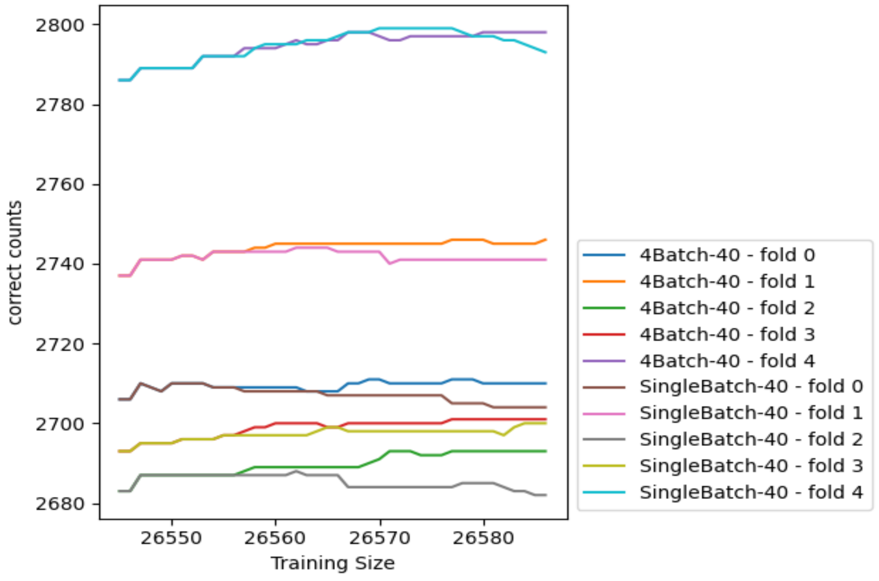


Figure 85: Learning curve count of correct recommendations per fold with single-batch and 4-batch sampling, adding 40 items

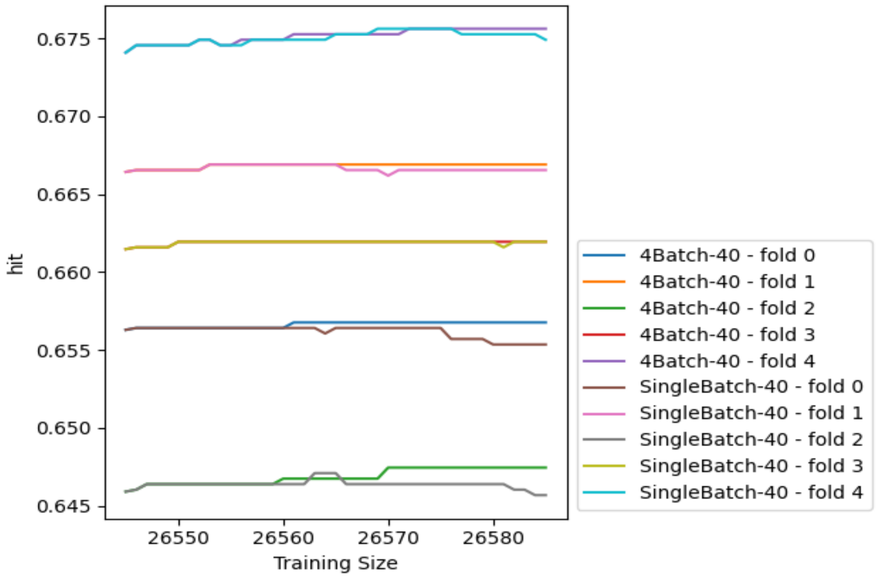


Figure 86: Learning curve hit per fold with single-batch and 4-batch sampling, adding 40 items

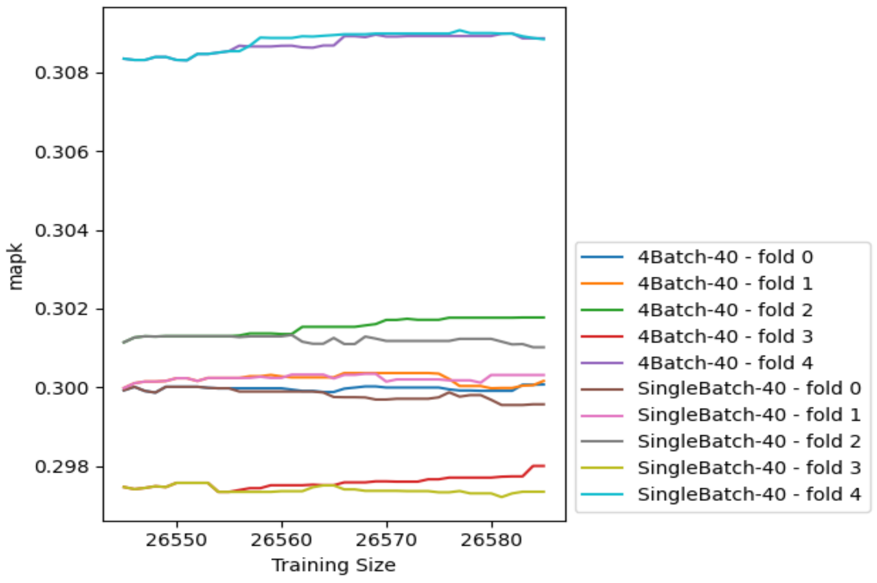


Figure 87: Learning curve MAP@K per fold with single-batch and 4-batch sampling, adding 40 items

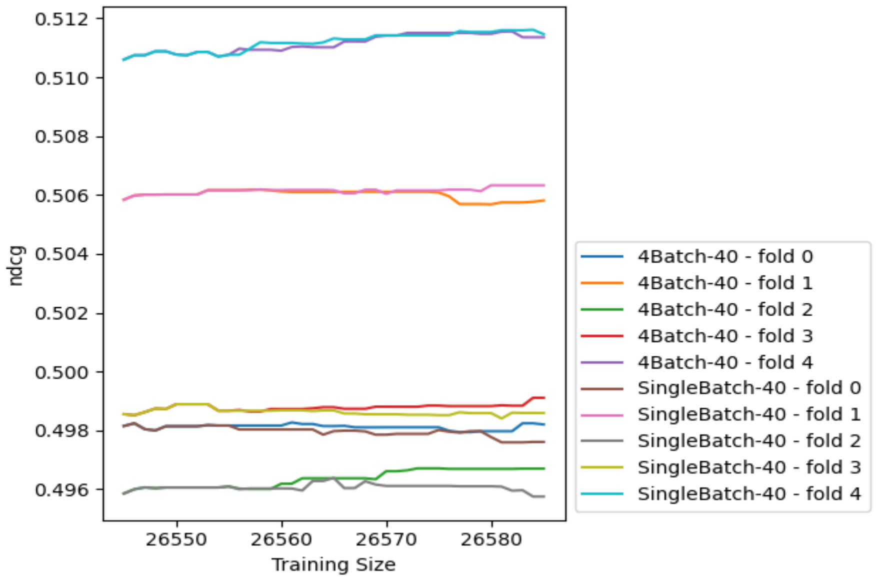


Figure 88: Learning curve nDCG per fold with single-batch and 4-batch sampling, adding 40 items

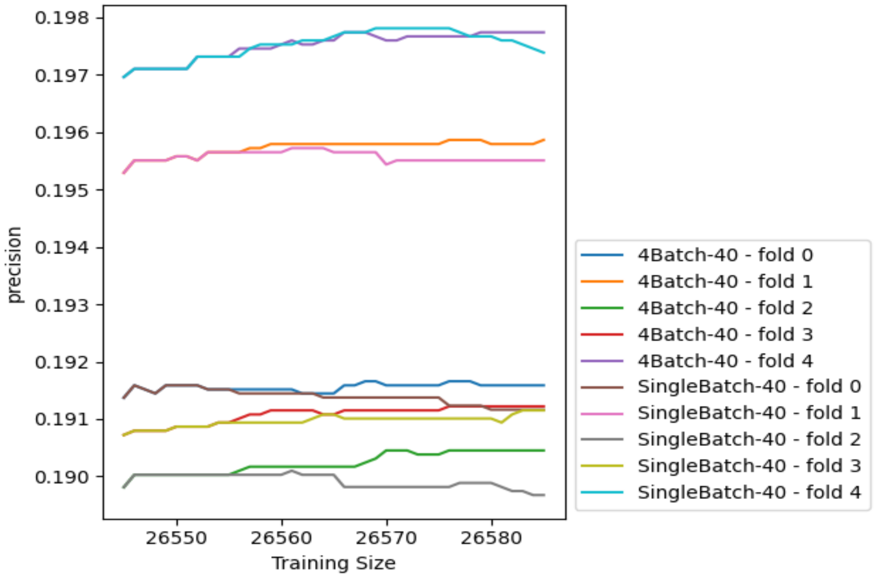


Figure 89: Learning curve precision per fold with single-batch and 4-batch sampling, adding 40 items

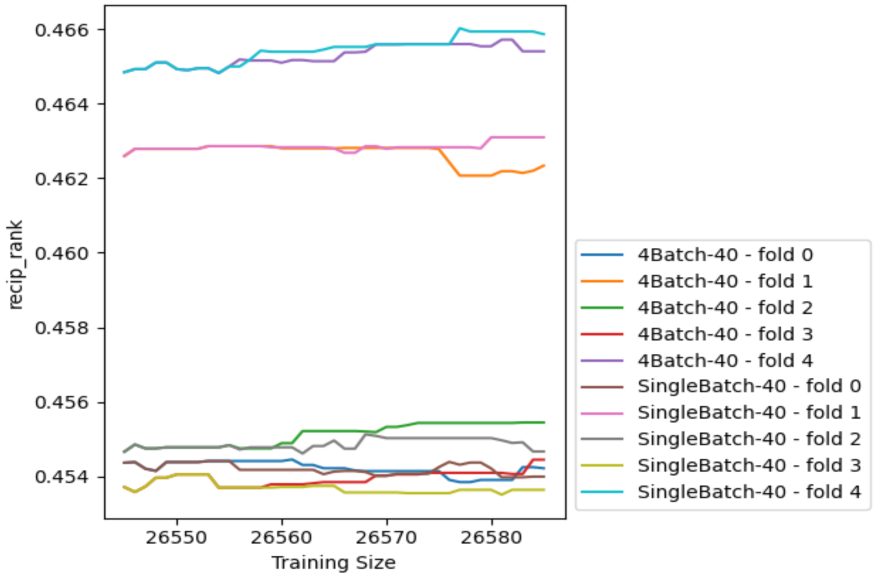


Figure 90: Learning curve recip_rank per fold with single-batch and 4-batch sampling, adding 40 items

A.8 10-batch and single-batch sampling strategy adding 40 items

Outcomes of the evaluation metrics: count of correct predicted items, HIT, MAP@K, nDCG, recip_rank, and precision for Item-based k-NN Collaborative Filtering are represented in Figures 91 till 96. These metrics are obtained by incorporating active learning with a single batch sampling strategy and a 10-batch sampling strategy where the strategy is updated 10 times. For both strategies, 40 items in total are added. In the 10-batch sampling strategy, 4 items are added per batch.

Figures 97 till 102 represent the learning curves where the performance scores of every fold is plotted against every additional item for active learning with single batch sampling strategy and 10-batch sampling strategy.

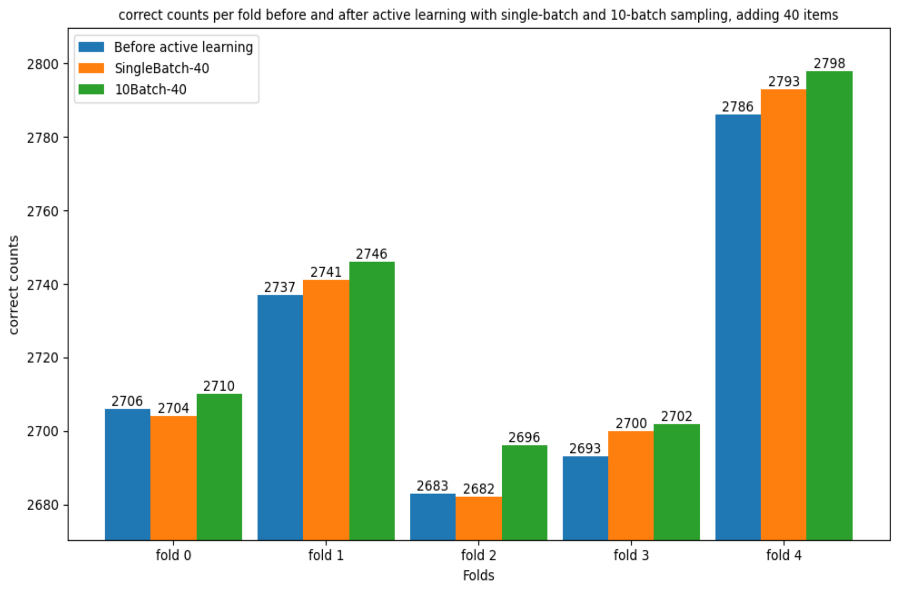


Figure 91: Count of correct recommendations per fold before and after active learning with single-batch and 10-batch sampling, adding 40 items

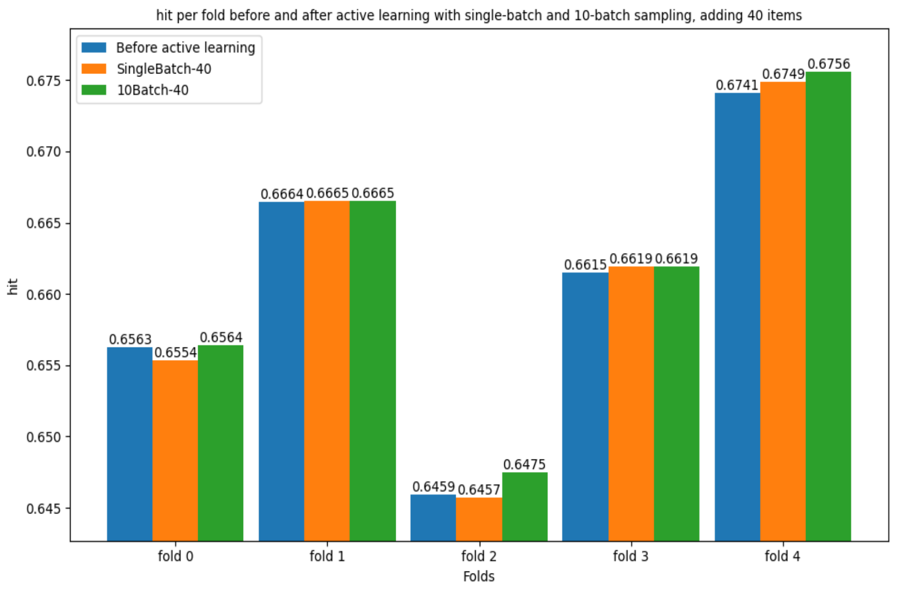


Figure 92: Hit per fold before and after active learning with single-batch and 10-batch sampling, adding 40 items

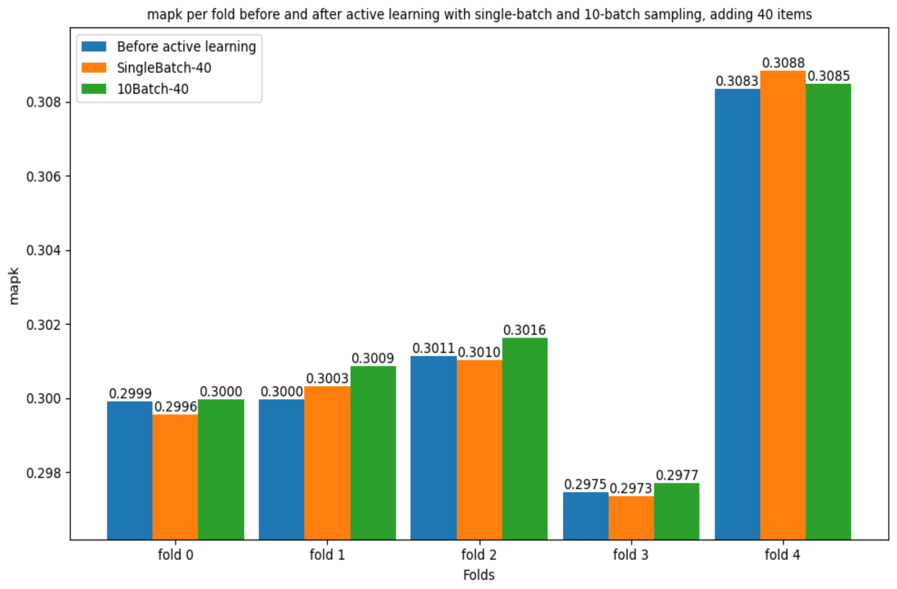


Figure 93: MAP@K per fold before and after active learning with single-batch and 10-batch sampling, adding 40 items

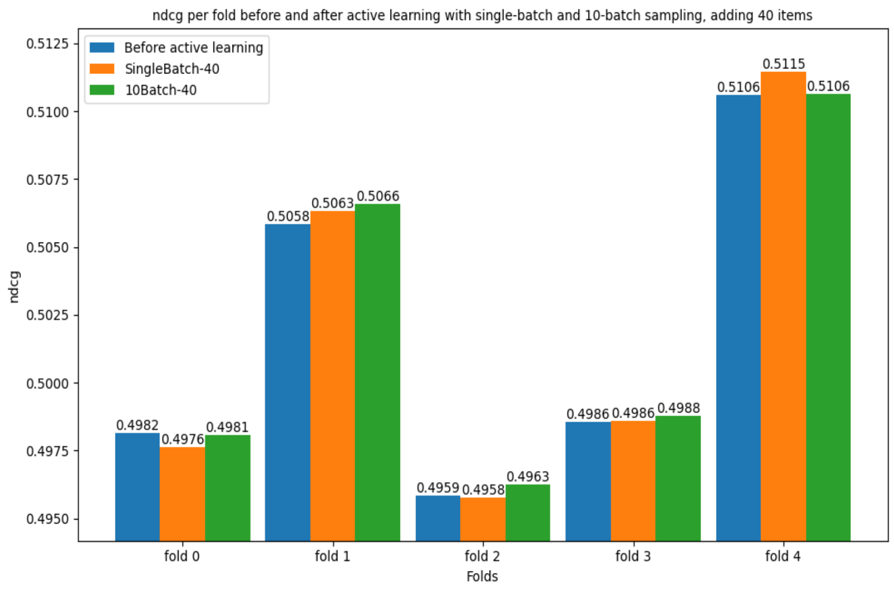


Figure 94: nDCG per fold before and after active learning with single-batch and 10-batch sampling, adding 40 items

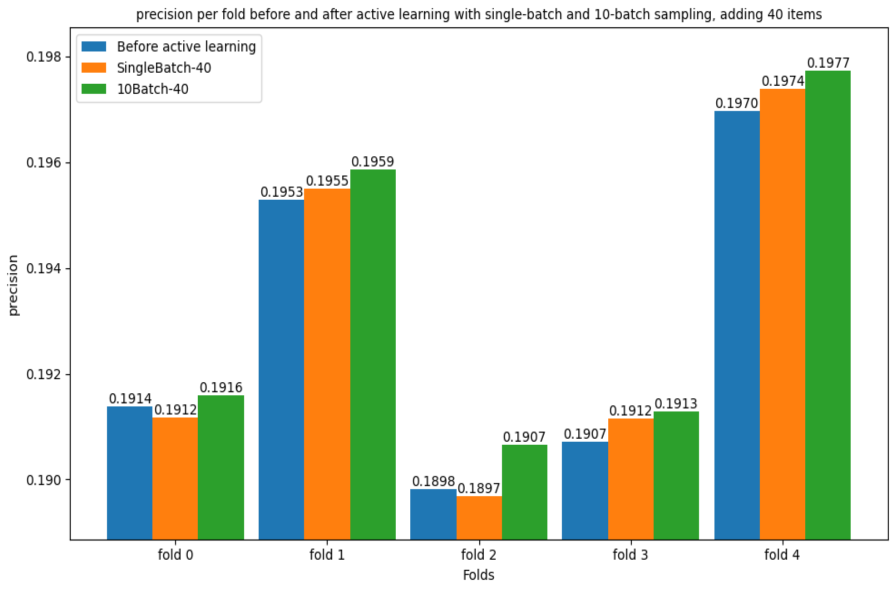


Figure 95: Precision per fold before and after active learning with single-batch and 10-batch sampling, adding 40 items

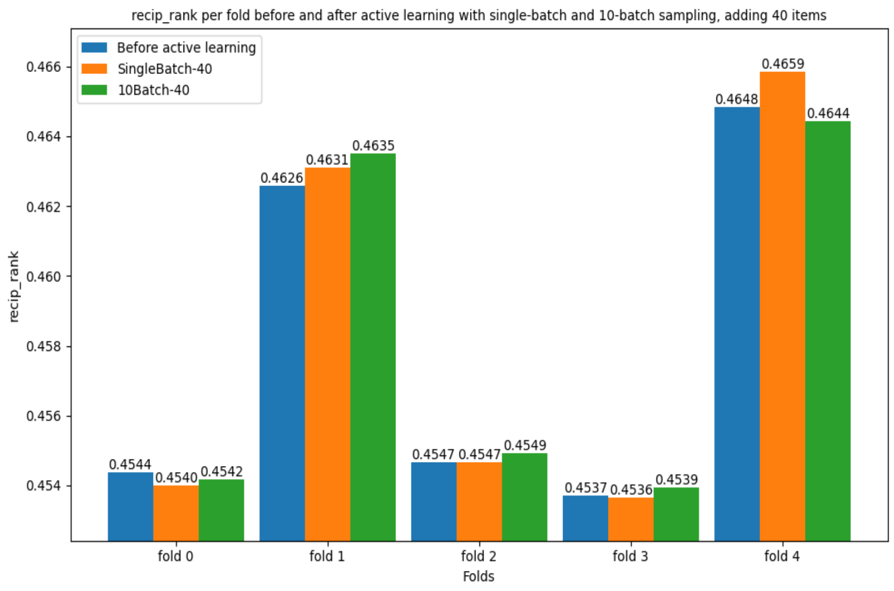


Figure 96: Recip_rank per fold before and after active learning with single-batch and 10-batch sampling, adding 40 items

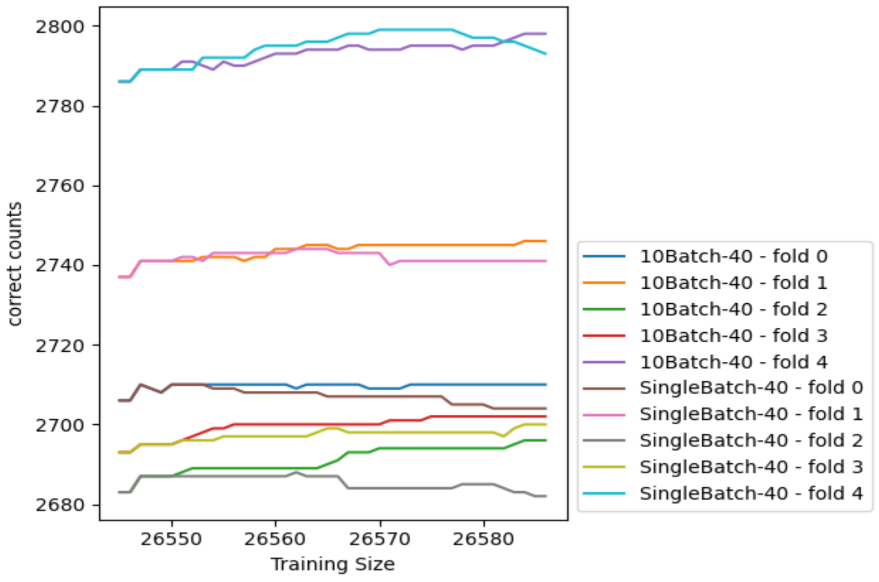


Figure 97: Learning curve count of correct recommendations per fold with single-batch and 10-batch sampling, adding 40 items

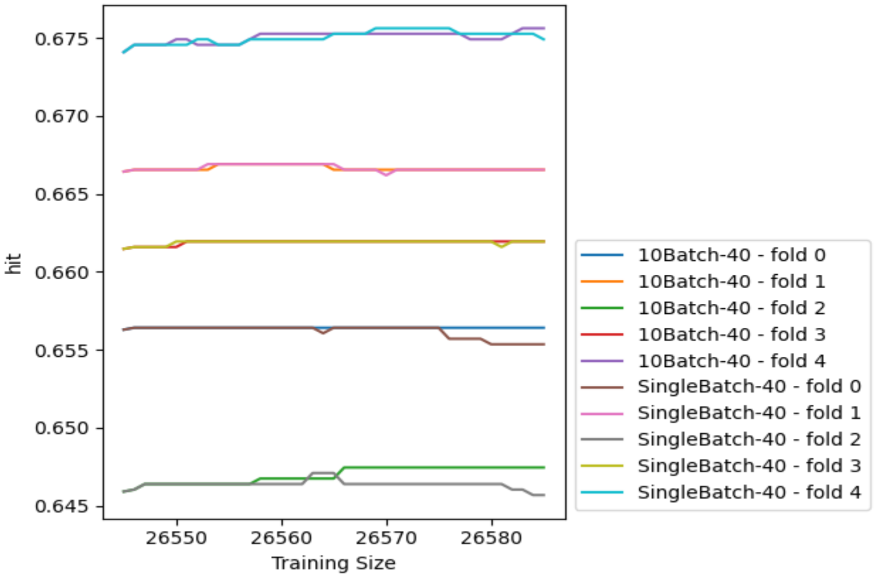


Figure 98: Learning curve hit per fold with single-batch and 10-batch sampling, adding 40 items

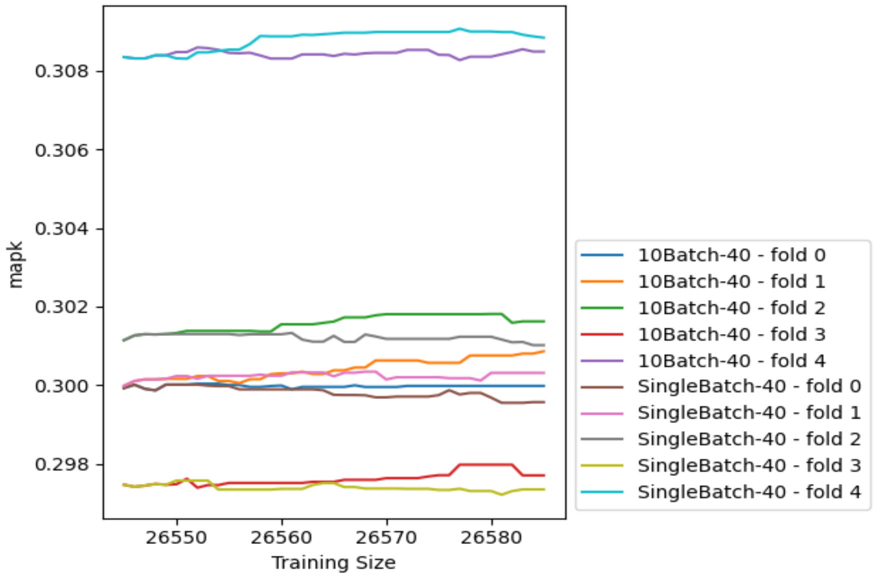


Figure 99: Learning curve MAP@K per fold with single-batch and 10-batch sampling, adding 40 items

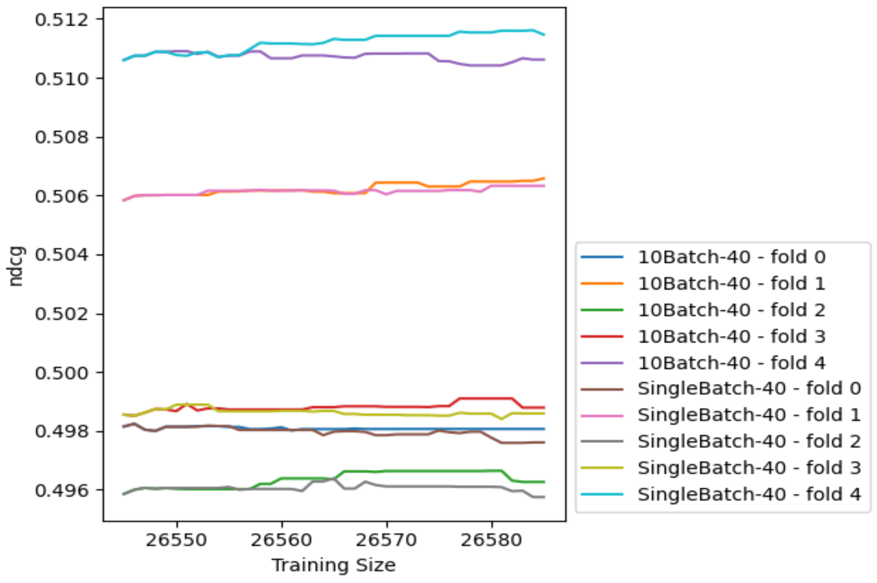


Figure 100: Learning curve nDCG per fold with single-batch and 10-batch sampling, adding 40 items

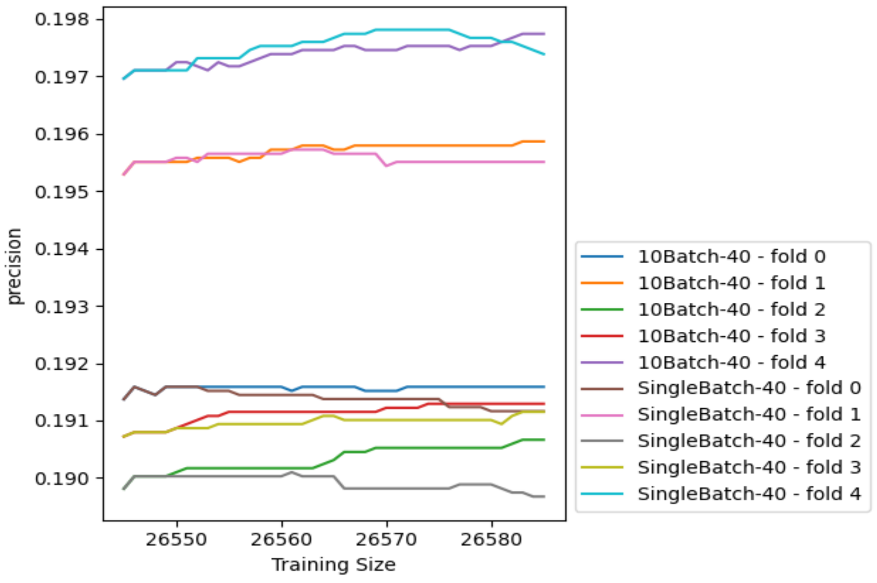


Figure 101: Learning curve precision per fold with single-batch and 10-batch sampling, adding 40 items

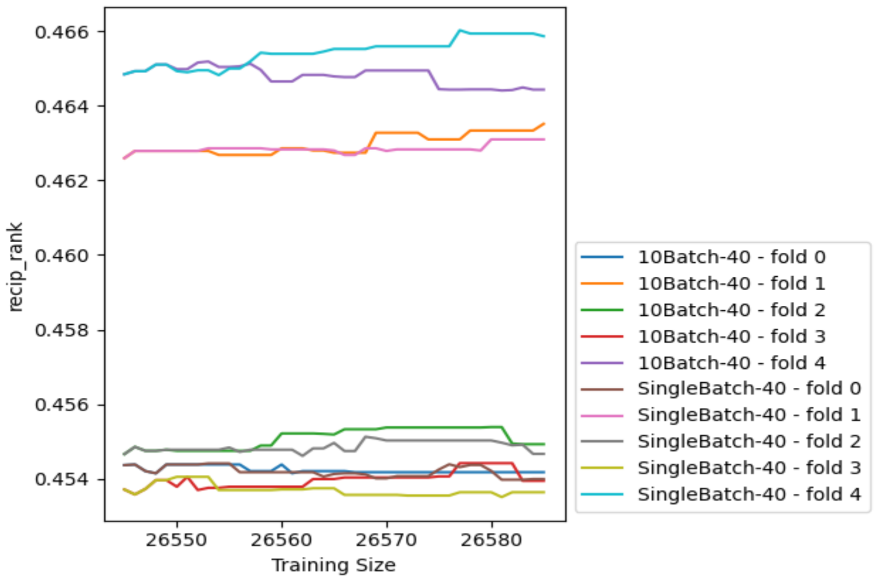


Figure 102: Learning curve recip_rank per fold with single-batch and 10-batch sampling, adding 40 items