Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

Multi-Armed Bandit algorithm for news recommendation systems

Author: Catalina Rete

Supervisor: Doug Leith

May 4, 2021

A dissertation submitted in partial fulfilment
of the requirements for the degree of
MCS (Integrated Computer Science)

Declaration

I hereby declare that this dissertation is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at `http://www.tcd.ie/calendar`.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at `http://tcd-ie.libguides.com/plagiarism/ready-steady-write`.

Signed: Catalina Rete Date: 04/05/2021

Abstract

Online platforms use recommender systems to learn user preferences and to recommend personalized items that are likely to be interesting to us. One recommender strategy widely employed is called collaborative filtering. This involves grouping users based on similar preferences and then recommending items users have liked to other users in the same group. One drawback of this technique is that the system fails to make recommendations for new items or new users introduced to the system for which it does not have any/sufficient information. The issue described is known as the cold-start problem.

In order to tackle the cold-start problem, this dissertation investigates the use of Multi-Armed Bandits algorithms for very fast learning which allow us to match the new elements with their respective clusters. To discover the preferences of new users, we show them a series of items that they need to rate. When learning about a new item instead, the goal is to show the item to a sequence of users who can distinguish between the groups in the fastest way possible. How to pick the items to be rated or the users to ask about the new item is a design choice that differentiates between the different strategies used to combat the cold-start problem.

The work is carried out on a standard news dataset and the performance is evaluated against the state of the art approaches that use Decision Trees to map the new elements to their corresponding clusters. The results show that the Multi-Armed Bandit algorithms can be used successfully to find out which clusters the new users belong to and that they generally outperform the Decision Trees approach. The work has also shown promising results when the setup is reversed and the system needs to deal with new items instead of new users.

Acknowledgements

I would like to thank my supervisor Doug Leith for his continued support and his prompt replies through each stage of the dissertation.

I would also like to thank my parents and my sister for encouraging me to pursue a Computer Science Masters degree and for always supporting me along the way.

Finally, I would like to thank my classmates, without whom the college experience would never have been as memorable.

Contents

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface
CART	Classification And Regression Trees
CB	Cluster Bandit
DT	Decision Tree
FN	False Negative
FP	False Positive
RMSE	Root Mean Square Error
TN	True Negative
TP	True Positive

1 Introduction

1.1 Motivation

The amount of information available online makes it increasingly difficult for users to find what they are looking for or to browse through. This has created a demand for online companies to develop recommendation systems in order to provide the best user experience.

The trend for recommending items is seen in almost all domains on the Internet. Whether it's music, movies, news, clothes or even ads, the system always tries to show the items it thinks are the most relevant to the user. The data collected by online companies is precious, precisely because it can be used to learn about a user's preferences and to recommend items they will be interested in. Liking, sharing, clicking on an item or even looking at it for prolonged periods of times can then serve as a way for the system to improve itself and learn more about the customers.

Having the best recommendation system seems to be a goal for many companies in their search to attract as many users as possible. Companies like Netflix, have even launched a competition that awarded $1M dollars to anyone who could improve the accuracy of their prediction system (2). One of the methods that performed the best on the dataset released is based on collaborative filtering, a recommendation approach that relies on having past feedback from the user, in the case of movies, the ratings from 1 to 5 stars that the users assigned to the movies they watched. In this way, users with similar interests are grouped together and items liked by one user are recommended to other users in the same group. Using this approach, however, leads to a really common problem called the cold-start problem: the inability of the system to make predictions when new users join, as the system doesn't have sufficient information to infer the preferences of the users and ultimately to recommend an item.

In the context of news, the users of the system may be more constant, but new items (articles) are constantly added to the system. Consequently, the system needs to quickly learn about the new item so that it can decide which readers to recommend it to. In this

dissertation, we will look at matching these news articles with groups of users. One way to do this is to use a Multi-Armed Bandit algorithm that involves showing the new articles to those users that have a track record of being predictive of the likes/dislikes of other users.

1.2 Research objectives

In order to tackle the cold-start problem in the case of news recommendation systems, the dissertation has the following objectives:

1. Find an appropriate news dataset
2. Use clustering algorithms on the dataset
3. Use the multi-armed bandit algorithm when a new user joins the system
4. Use the multi-armed bandit algorithm when a new article is added to the system
5. Evaluate the performance of the results achieved against the state of the art approaches

1.3 Dissertation overview

The remainder of the paper has the following structure:

- Chapter 2 discusses the different ways of making recommendations and addressing the cold-start problem, with an emphasis on the Cluster-Based bandits that are going to be used in the approach taken in this dissertation.
- Chapter 3 focuses on the work involved when choosing an appropriate dataset, including data analysis and preprocessing, feature engineering and the clustering process.
- Chapter 4 is a presentation of the results obtained and a comparison with the baseline chosen.
- Chapter 5 is a discussion of future work and potential improvements.

2 Background

This chapter presents the two main strategies that are used when making recommendations. It then discusses the cold-start problem and the solutions that can be employed to tackle this problem.

2.1 Recommender systems strategies

The main difference between the paradigms that emerged when building recommender systems consists in the type of data that is used to learn and track the preferences of the users:

- **Content-based filtering** analyses the users and items individually and makes recommendations by associating user profiles with item profiles.
- **Collaborative filtering** analyses multiple users and makes recommendations by detecting similarities between the preferences and ratings given by these users to common items.

2.1.1 Content-based filtering

Content-based filtering (3) is a recommendation strategy that involves showing the user those items that are similar to those that the user has previously rated highly. In this strategy, a profile is created for both the user and the item. The profiles must include attributes related to the elements in question. For example, a user profile might contain their age, gender, location and so on. Similarly, in the case of items, such as movies, the profiles might contain the genre, the participating actors, language and more. The recommendation task becomes matching the user profiles against the items profiles.

In order to create these profiles, the system generally requires a lot of information about the user and the items. There is also some heavy preprocessing involved in extracting attributes from the data, which might not be as straightforward as in the case of movies. For example, when dealing with documents, such as news articles, one way of creating the item profile is to analyse the content and map it to a feature vector using the bag of words model and

TF-IDF (statistical method that is used to measure the relevance of a word in a document). Then a similarity measure is used to determine the similarity between items. One common technique is to use the cosine similarity on the item vectors. Predicting what rating a user would give to a new item in the system involves finding the most similar items that the user has already rated.

Robin van Meteren and Maarten van Someren have previously built a system which suggests small articles about home improvements using content-based filtering (4). To test the system, they created fictitious users and assigned topics to them, then they tried recommending documents containing those topics, as those should be relevant to the user. One issue that arose is related to classifying documents into topics, because documents about multiple topics contain a lot of terms that make it then harder to distinguish between them.

Content-based filtering has also been applied in the context of news recommendation systems. One such system, called YourNews (5) creates a user profile for each news topic available on its website (e.g. World News, Business, Technology etc.). The recommendations are based on short-term profiles, which take into account the 20 most recent articles read, and long-term profiles that record the entire previous browsing behaviour. An interesting feature about YourNews is that it also allows the user to modify their own profiles, giving them more control over their recommendations. However, even when assessing the performance of such system, a lot of manual work is required to annotate the articles as irrelevant or relevant.

This technique does not suffer from the cold-start problem. When a new item enters the system, we can still make predictions about it based on the history of the user, even if other users haven't seen it yet. However, there are several other disadvantages that make the collaborative filtering more appealing and also more accurate. The features extracted from the items that make up the item profile are often manually selected and they can require extensive domain knowledge. Even after finding these features, they may not be enough to distinguish between what a user likes or dislikes, similar to the home improvement recommender system. There is also an issue when it comes to recommending novel items to the user. Because the recommendations are based on what we currently know about the user's preferences based on their history, the recommender system fails to recommend completely new items to the user, e.g. if a user has only read books by Stephen King, the recommender system may be inclined to only recommend books by the same author.

2.1.2 Collaborative filtering

As opposed to the Content-based filtering method which relies on information about a single user when recommending new items, the collaborative filtering approach involves looking at

the feedback generated by multiple users and finding correlations between the items liked by one user and the items liked by the rest. The term *Collaborative filtering* has been first used in the context of filtering electronic mail. The developers of Tapestry (6) recommendation system recorded the reactions of multiple users when reading documents to determine whether the documents were interesting or not and thus to filter out the ones that nobody wanted to read.

Collaborative filtering doesn't require the creation of profiles, instead it looks at the past behavior of the user and the ratings they gave to the items, whether they gave 1 star or 5 stars to the movie, liked/disliked the article or even less obvious feedback such as clicking or looking at an item for long periods of time. Domain knowledge is not required in the case of collaborative filtering.

One way of performing collaborative filtering is through the neighbourhood method. This involves finding items the user has liked and then finding similar users users who have also liked these items. By looking at the ratings given to the "neighboring" items that these users have liked, recommendations can be made. This is an item-oriented approach.

Another way of finding new user-item associations is by using latent factor models. The system uses the ratings to infer factors/features about both the items and the users. These features could be apparent such as classifying movies into genres and then finding the preferences of the users, or they could be very abstract. To find these latent features, we use a technique called Matrix Factorization.

Matrix factorization is a common approach used in recommender systems (7). Given a set of users and items, together with their ratings, we can create a matrix R, where each row in R corresponds to a user, and each column to an item. The value of the matrix element (i,j) is the rating given by user i to item j. Matrix factorization is then used to create a latent factor model, where the interaction between users and items is mapped as an inner product. A simple example can be seen in Figure 2.1, where users 1,2,3 have been asked to rate items A,B,C on a scale of 1-3. 0 values mean they haven't seen the items before and there are no ratings recorded for them. This is often the case in a system, as users have generally seen only a small fraction of the items on record, the rating matrices created tend to be really sparse.

The Ratings matrix is decomposed in the following way:

$$R = Q^T P.$$

Where Q is a measure of the items and to what extent they present the factors extracted from the ratings patterns and P is the vector that corresponds to users and their interest in these factors. The system calculates the mapping Q and P by fitting the observed ratings

and minimizing the regularized squared error on these ratings. This can be done through classical techniques such as Stochastic gradient descent.

	A	B	C
1	1	3	0
2	0	0	0
3	2	0	2

Figure 2.1: Example of a ratings matrix

While Collaborative filtering is superior to Content-based filtering due to its domain-free implementation, it is susceptible to the cold-start problem. When a new user is introduced to the system, there are no ratings available for them that can be used in the Ratings matrix, and so the system cannot make any recommendations without extra information.

2.2 The cold-start problem

The cold-start problem is an issue that arises in recommendation systems that use Collaborative filtering when a new item or a new user enters the system because there aren't any ratings associated with these new elements. In the absence of these ratings, the system cannot infer any features. Thus, we need to quickly learn information about these.

2.2.1 Strategies for learning the preferences of new users

The first thing we do when we want to learn about someone new is to ask them questions. How we ask these questions and what type of questions we ask is crucial in the learning

process. For example, if we met someone new and we asked them if they enjoyed the Titanic movie, we would probably find nothing new about their preferences because almost everyone who watched this movie has rated it highly.

Before discussing the algorithms for modelling these questions, we have to take a look at the general strategies that can be used for picking the questions or the items that we show to a new user in order to learn as much about them as possible (8):

- **Random**. One obvious way of choosing the items to show to the users is to have a random strategy. However, depending on the domain used, this might not work very well. If we asked users about random movies, there's a high chance they never even heard of the selected items and so we cannot find any information about them.
- **Popularity**. Another strategy is to ask the user about popular items that many others users have rated. While an easy strategy to implement, the information learned can be minimal, due to the fact that really popular items tend to be liked by most people who watch them, e.g the Titanic example.
- **Entropy**. This strategy involves asking the user about the items that will reveal the most information about their preferences, i.e if an item is controversial, we can find much more about the user's preferences than asking about popular items. Nevertheless, this strategy also has its disadvantages, as the information found may be unusable by the recommendation system if it doesn't contain enough other users that have also rated these items.

Each strategy described presents a series of advantages and drawbacks. It's clear that using only one will not produce the best results. Most of the times, a balanced strategy is preferred where multiple strategies are combined into one.

The idea of asking the questions that will lead to the fastest learning process can also be inverted and applied to when a new item is introduced to the system as opposed to a new user. The problem then becomes to find the users whose ratings of the new item will be the most predictive of other users' preferences.

2.2.2 CART Decision tree

A proposed solution to the cold-start problem that gave very promising results is Adaptive Bootstrapping using Decision Trees (1). This has been tested on the Netflix movie recommendation dataset. In order to find out the preferences of the new user, this approach involves an interview phase, where the user is asked to rate certain items. Their responses influence the choice of the items to be rated next. Decision Trees are really popular when it comes to solving classification and prediction problems (9), but it turns out that they can also be used to model the interview phase.

The Decision Tree proposed for Bootstrapping recommendations has the following structure:

- Ternary decision tree
- Each node corresponds to an item that has to be rated by the user
- The path taken from the node is decided by one of the three available ratings: "like", "dislike" or "unknown". If the dataset contains more ratings, e.g from 1-5 stars, these have to be mapped in the 3 categories. In the case of the Netflix dataset where the average rating is 3.6, every rating that fell below this threshold was classified as "dislike" and above as "like".
- Each node also represents a group of users. The mean rating of the users is used to predict item ratings.
- When constructing the tree, an item is chosen at each node so that it will partition the users into the 3 sub trees in a way that minimizes the total squared prediction error.
- The termination criteria is usually reaching a predefined tree depth (6 or 7) because going further rarely yields benefits. Another termination criteria is that the item chosen at the node cannot reduce the square error or that the number of users left for the current node drops below a certain threshold.

The tree structure described is exemplified in Figure 2.2. If a user is asked about their rating of Pearl Harbor and they have never seen this movie before, they will be routed to the Dr. Strangelove node. Similarly, for a dislike they will be asked next about Sweet Home Alabama. If they liked Pearl Harbor, they will be asked to rate Sister Act. At each step, the Root Mean Square Error (RMSE) is calculated using the predicted ratings of the item and the actual rating given by the user.

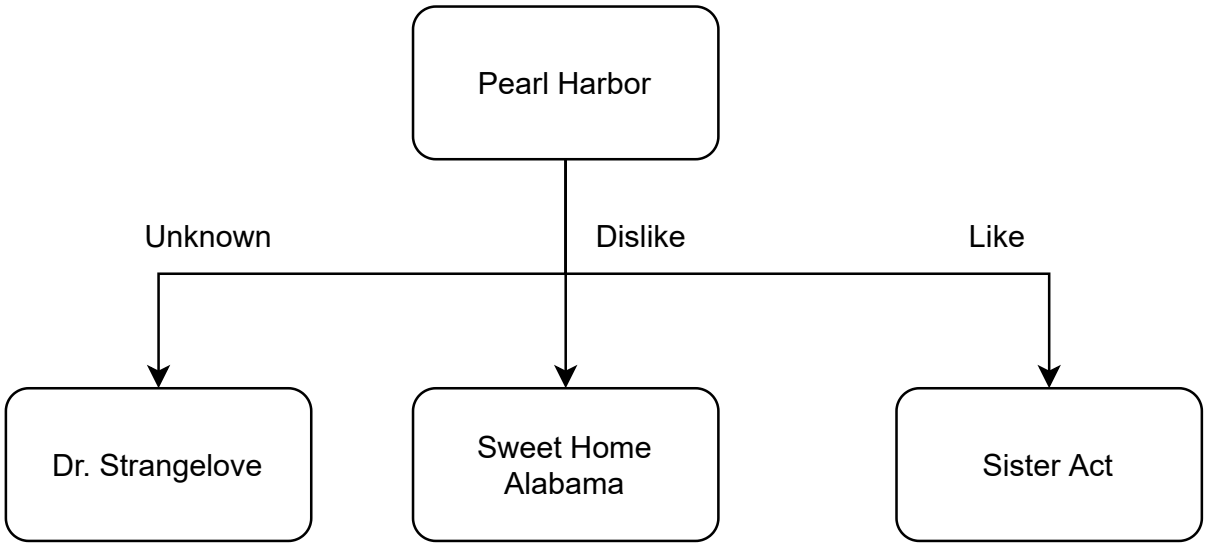

Figure 2.2: Decision Tree structure example adapted from (1)

To successfully minimize the cost function associated with the tree, the items that are chosen to partition the tree often present a combination of the characteristics mentioned in the previous section (popular items, controversial items and so on). After the new user traverses the tree and finally reaches the leaf node, or the termination criteria, the system should be able to predict items to the user. There are different recommendation policies that can be implemented. One strategy is to recommend to the user the items that are rated the highest at the final node they arrived at.

2.2.3 Cluster-based Bandits

Multi-armed bandits algorithms, first described by Robins (10), are a type of reinforcement learning algorithm. The "multi-armed bandit" name is inspired by traditional slot machines (one-armed bandits), although they generally have more than one arm in the problems they address. Each arm is associated with a reward. A gambler without any prior knowledge of what each lever would give them, must figure out the sequence of arms to pull in order to maximize their reward. After pulling several arms and seeing the rewards, an important question arises: should the gambler keep pulling the arm that awarded the most so far, or should they continue pulling different arms to find out about the potential reward? This is a very common problem in reinforcement learning and it's known as the exploitation vs exploration trade off. Therefore, the multi-armed bandit problem refers to the problem of finding the best strategy in order to maximize the reward in a series of steps.

A lot of algorithms have been proposed to solve this problem. One common strategy is called the ϵ-Greedy strategy. It involves pulling a random arm with a frequency of ϵ, and the rest of the times pulling the best arm as estimated thus far. The exploration, which consists of pulling the random arm, can be done at different stages. If it's done all at the beginning, the strategy is usually called ϵ-first strategy. Another variation of the ϵ-greedy strategy is the ϵ-decreasing strategy. As the name suggests, what differentiates this algorithm from the rest is that the ϵ value decreases as the number of pulls increases. There are many more different strategies and variations but studies (11) show that when the arms reward distributions are normally distributed, the ϵ greedy strategies perform really well and are hard to outperform.

These strategies could also be applied to solve the cold-start problem by having the user rate different items in order to learn more about them. However, the problem encountered is that if we treat each item as an arm, the learning will be slow because there are many arms to pull. The new variation of Multi-Armed Bandits, called Cluster-Based Bandits (12), developed by Sulthana Shams, Daron Andreson and Douglas Leith at Trinity College Dublin, proposes associating an arm of the bandit with a group instead of an item.

The cold-start problem is then addressed in the following way:

1. Cluster users into groups based on interests.
2. When a new user joins the system, attempt to learn their preferences by presenting them with a sequence of items and ask them to rate the item (e.g. from a scale of 1 to 5).
3. After having asked enough questions, assign a user to a cluster.

Once we know which cluster the user belongs to, we can recommend items to them as we would to the other users in that cluster. The important part of the algorithm consists in choosing which items to show to the user in order to find the correct cluster in as few steps as possible. The paper proposes finding the items it describes as "distinguisher" items. Assuming we know an estimation of the mean ratings and variances of the items in each group, a distinguisher item v helps distinguish between 2 groups, g and h, when it has the following properties:

- The ratings are reliable, i.e the variance of the item v in group g is small.
- The mean rating of the item in group g is very different from the mean rating of the item in group h.

These properties lead to defining the formula for finding the distinguishers items as follows:

$$\Gamma(v) = \frac{(\mu(g, v) - \mu(h, v))^2}{\sigma^2(g, v)}$$

Where μ are the mean ratings by the group in question for each item and σ values are the variances of the item in each group. The largest gamma values correspond to the best distinguishers.

The exploration phase in this algorithm is represented by the user rating the distinguisher items (in a descending order, i.e. find the best distinguishers between each groups), whereas the exploitation involves the user rating the items with the highest predicted ratings (items that have the highest mean rating for the group we are estimating the new user to belong to). The Cluster-Based Bandit attributes its fast performance to having an initial exploration phase. After this phase, the algorithm has to balance between exploring new groups and exploiting to confirm the group currently estimated.

The Cluster-Based Bandits performance has been evaluated on 3 datasets: Netflix, Jester Jokes and Goodreads Books datasets. The results have been compared to a CART Decision Tree algorithm (similar to the one described in the previous section). The Cluster-Based Bandits have outperformed the baseline chosen in both accuracy and performance (number of items the user has to rate before finding out their cluster).

The goal of this dissertation is to explore whether the algorithm performs similarly in a different domain, namely in the context of News recommendation systems. In addition to

this, it will also investigate what happens when a new item enters the system, instead of a new user, unlike the setup for the evaluations already performed.

3 Design

This chapter describes the work that went into finding an appropriate news dataset and processing the data to extract the relevant features. It also discusses the clustering algorithm used.

3.1 User Clustering

The Multi-Armed Bandit algorithm works with groups of users, so the first step in reaching the dissertation objective is to form clusters of users. This is achieved by finding an appropriate dataset and then using BLC: Private Matrix Factorization (13). Another clustering algorithm could have been used as well.

While recommender systems that use matrix factorization techniques are widely used, they are susceptible to different kinds of privacy attacks. This happens even when the data uses anonymized IDs. A classic example of this is the Netflix recommendation competition that had to be cancelled due to a privacy lawsuit triggered by the fact that the data could easily be de-anonymized (14). In consequence, if the databases used by the recommender systems are compromised/leaked, the users can potentially be identified which raises a lot of privacy concerns.

The BLC algorithm aims to solve the privacy issues, by leveraging the fact that users can be grouped together by interests. In fact, it proves that most datasets require fewer than 20 groups to give a good performance. Using BLC, the recommender system doesn't need information about individual users and it can successfully make recommendations to the groups they belong to. The clustering process is performed using some artificial "pseudonyms" or nyms for short, that are shared between users with similar interests.

The matrix factorization approach used in BLC decomposes the matrix R of users u and items v as $P^T U^T V$, where matrix P^T maps from users to nyms. This decomposition is performed in a privacy-enhanced manner, without using cryptography.

Despite the privacy enhancements and the modifications they entail, the algorithm is proven to preserve the recommendation accuracy. In some cases, it even outperforms the state of

the art algorithms due to the shared nature of the nyms, which allows the system to make direct connections between users' preferences.

There are several configurable parameters:

- **Nyms**. The number of nyms is directly related to the number of clusters the users are going to be split into. Generally, no more than 32 nyms is ever required.
- **Features**. This represents the number of latent features to infer from the ratings patterns.
- **L2 regularization**. The L2 regularization value is used to ensure that the algorithm is not overfitting the data.

3.2 Data Preprocessing

The dataset used is SmartMedia Adressa News Dataset (15) which was published by Norwegian University of Science and Technology (NTNU) and Adressavisen (local newspaper in Trondheim, Norway). It is available in 2 versions: one week worth of data and 3 months worth of data. For the purpose of this report, we are using the shorter version, which covers one week of web traffic on the newspaper page (January 1st to January 7th 2017).

From the dataset, we extract the news articles, along with the anonymized users who have viewed the articles for at least one second. In particular, we are only interested in the information that can be seen in Table 3.1.

Table 3.1: Relevant attributes from the Norwegian news dataset.

Attribute	Description
userID	Identifier that can be used to differentiate between different users/devices.
activeTime	The time the user has spent on a page in seconds. Can be missing.
ID	Identifier that can be used to differentiate between different articles. This will be the same for different URLs that are considered equivalent according to Cxense's algorithm. (Note: This is called documentID in the official documentation, but just id in the actual dataset)

There is a separate json file for each day of the week, with each json line in the individual files representing a reading event. The document ID that is used to differentiate between different articles is missing from a large portion of the entries. We assume that this happens when the user is simply browsing the homepage or other menus and pages of the online newspaper that do not necessarily represent a news article. Filtering the data to keep only

the articles that are assigned an ID means removing a large chunk of the data as indicated in Table 3.2 .

Table 3.2: Norwegian dataset filtering breakdown.

Day	Number of events	Percentage of data removed
Sunday	1513739	71%
Monday	1613128	68%
Tuesday	1648346	70%
Wednesday	1496417	59%
Thursday	1327429	65%
Friday	1356987	78%
Saturday	1087223	70%

There are also duplicate entries which consist of a user reading the same articles multiple times on different occasions. We remove the duplicates and keep the entries with the largest reading time (as we assume that a user going back to the same article indicates a high interest in the article). Removing duplicates means removing a further 7.92% of the data. This leaves us with the data described in Table 3.3.

Table 3.3: Norwegian news dataset statistics after removing irrelevant data and duplicate entries.

Users	267718
Items	9426
User-Item ratings	1216780

We analyse the resulting data by plotting a histogram of the number of articles that a user reads, Figure 3.1. On average, a user reads roughly 5 articles and the average number views for an article is 130.

The time spent on an article ranges from 1 second to 895 seconds. A full distribution of the times is seen in Figure 3.2. Unfortunately, the word count of each article is not present in the versions of the dataset that are readily available online.

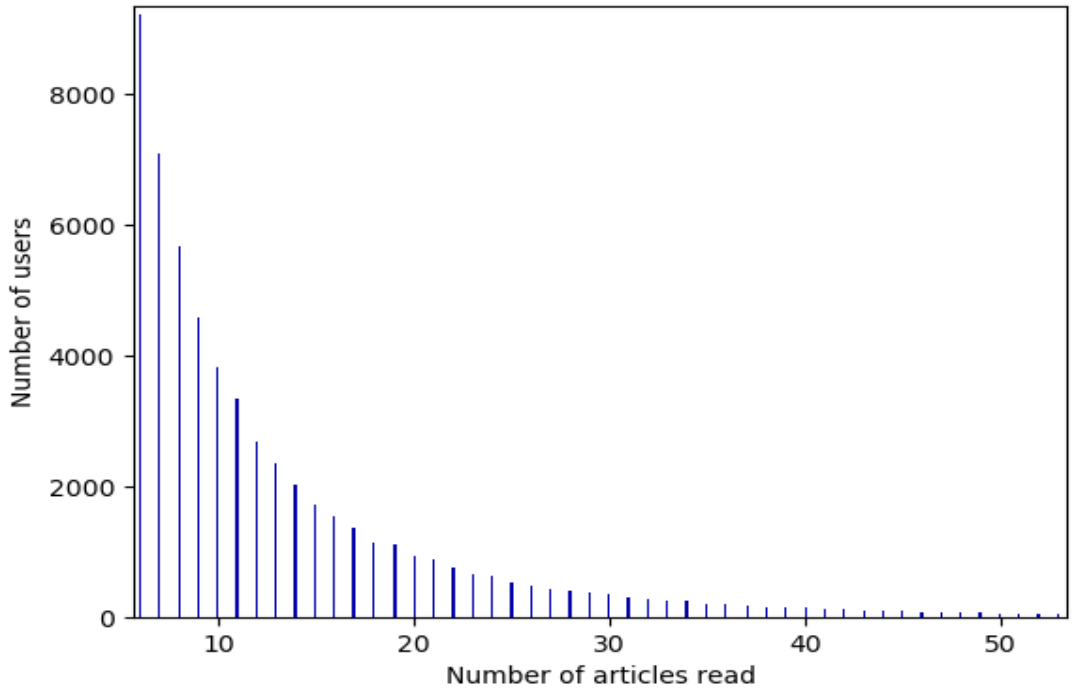

Figure 3.1: Reading behaviour analysis

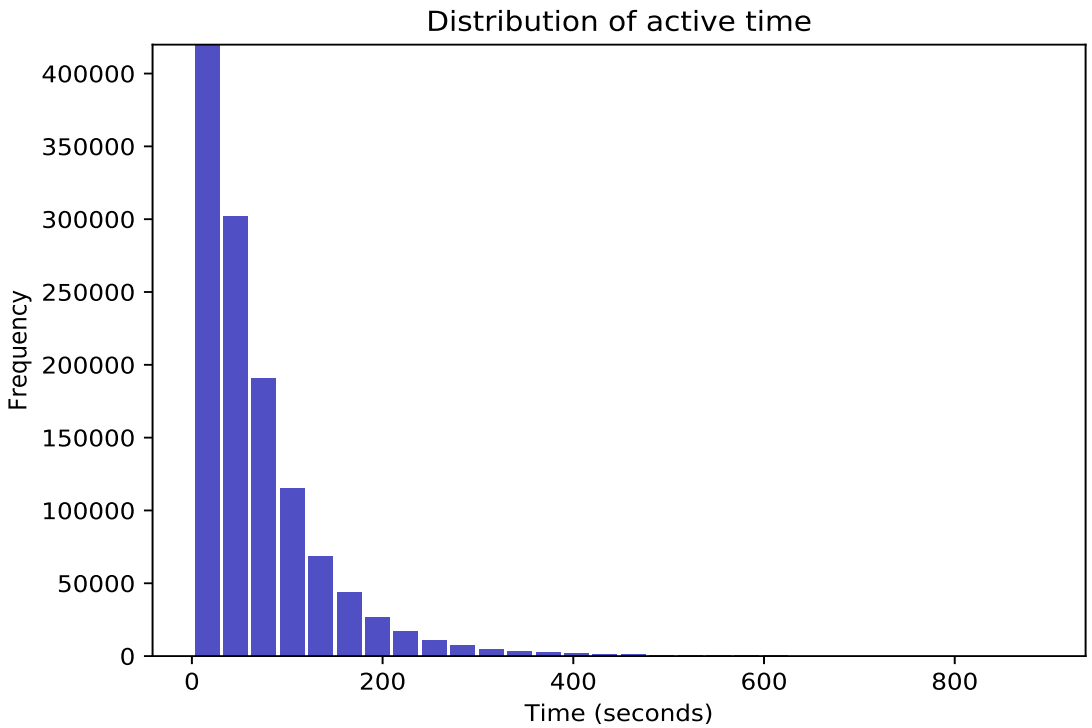

Figure 3.2: Distribution of activeTime in seconds

There are two important limitations about the dataset. There is a distinction between normal users and subscribers, the former being unable to access articles only meant for paid users. This means that non-subscribers simply cannot read all the articles that are interesting to them. Another limitation that stems from the same 2-tier user system is that normal users cannot be tracked between sessions, which means that every session constructs a new userID. This applies in the case of different devices as well. The only way to differentiate between subscriber users and non-subscribers is to look for users that have read articles which contain the word "pluss" in their associated URL, as these are only available in the premium version of the newspaper. Since the first step in this project involves clustering users based on interest, we suspect that users who appear multiple times in the dataset due to having different IDs will be placed in the same cluster, as their interests should be fairly constant during the time span of a week.

3.3 Measuring Performance

A common practice when training machine learning models is to split the dataset (e.g. 70%/30%) into training and testing. After training the model, we apply it on the held-out data and we use different accuracy metrics in order to understand how well our model is performing. There are multiple ways to do this and several accuracy metrics available for use. More often than not, a combination is required to measure the performance of the system accurately. This chapter explores some of the most common metrics in order to motivate the choice we made when choosing the metrics for our system.

Root Mean Square Error (RMSE) has been traditionally used to evaluate the performance of recommender systems.

$$RMSE = \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}(Predicted_i - Actual_i)^2}$$

The Netflix Prize competition decided the winner based on which team could improve the RMSE of its algorithm over 10%. This metric is also the default one used to measure the performance of the BLC algorithm.

Another way to assess the performance of the recommendation system is to view the ratings of the system as classes and to analyse the confusion matrix. A confusion matrix is a type of table, where the rows represent the predicted classes, and the columns the actual classes. In the case of binary classification, where one class is called "negative" and the other "positive", the confusion matrix would look like in Figure 3.3 .

	True class	
Predicted class	**Positive**	**Negative**
Positive	True positive (TP)	False positive (FP)
Negative	False negative (FN)	True negative (TN)

Figure 3.3: Confusion matrix for binary classification

This can be expanded to multi-label classification, where the values on the diagonal represent how many of the actual class have been correctly predicted. Additionally, using the information in the table, we can define further metrics that are commonly used in evaluating model performance:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Recall = \frac{TP}{TP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

While accuracy, the number of correctly classified out of the total number of predictions, is perhaps the most intuitive, it can be misleading in the case of imbalanced datasets. Imbalanced datasets have a disproportionate number of samples of one class, e.g. 90% are negative, while only 10% are positive. Even if we only ever predicted negative, then our model would have a 90% accuracy, which may seem great at first glance, but in fact, we're completely ignoring the positive class. If the positive class represented someone carrying a virus, then the high number of false negatives would be highly concerning. Similarly, if the proportions of the classes are inverted (10% negative, 90% positive) and we only predicted positive we would have a high number of false positives. A classic example is when the positive class represents a spam email, then a lot of the important emails would also be classified as spam. This is why recall and precision are important for the performance assessment. In fact, they are usually combined in what's known as the F1 score using the following formula:

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

If we had a 5-star system for our recommendation system, we could consider each category (1 star, 2 stars etc.) as a class/label. We can expand the metrics above to work for

multi-label classification. Depending on the averaging technique used, there are a few slightly different approaches:

- **micro-average**. Calculate the F1 score by using the total number of TPs, FNs and FPs, regardless of the label.
- **macro-average**. Calculate the F1 score for each label (1 star, 2 stars etc.) and then the unweighted mean of these scores.
- **weighted-average**. Calculate the F1 score for each label and then average them together by also taking into consideration the number of samples available for each label.

For the rest of the report, we will mainly focus on the confusion matrices and the macro-averaged F1-score when looking at our model's performance. This is due to the fact that our dataset is imbalanced (most people don't spend too much time reading a news article). We will also compare the accuracy of the system with a baseline accuracy which is obtained by predicting the most common label in the data.

3.4 Feature Engineering

In order to make recommendations, the system needs to understand which items the users disliked/liked. Sometimes, this can be straightforward, when users rate the item on a scale, for example from 1 to 10 stars like on IMDB[1], or use a dislike/like button similarly to Reddit[2] posts. This is called Explicit Feedback and it's the most convenient to work with.

When users don't show their preferences in an evident way as the examples above, the system can learn from their behaviour instead, e.g. browsing history, mouse movements or search history. This is called Implicit Feedback.

In the case of the Norwegian News dataset, we can take what is essentially implicit feedback, the time spent on an article, and attempt to transform it into explicit feedback. This is done by splitting up the different times into buckets, to which we then assign ratings. For example, if the user has spent less than 30 seconds on an article, we assign a rating of 1 which would signify they are not very interested in that item, more than 30 seconds would be a rating of 2 and so on. As the dataset is imbalanced (not many people spend a long time on an article), a higher rating is associated with the buckets that contain longer reading times in order to add more weight to these buckets.

The tuple (user, item, rating) is used to create a matrix that is then fed to the BLC algorithm. The resulting ratings matrix R is sparse, with only 0.048% of elements observed

[1] https://www.imdb.com/
[2] https://www.reddit.com/

(the other user-item ratings are unknown). The matrix is then split into train and test data (usually 90%/10% split). We run the BLC algorithm and we analyse the performance of the model on the held-out data using confusion matrices and the F1-score.

The challenge is to find the appropriate splits for distributing the times into the buckets. We start with 2 buckets and we dynamically try out different thresholds: first bucket (0s,Xs) and second bucket (Xs, maximum time), where X is between 10 seconds and 150 seconds and the maximum time spent on an article is 894 seconds. We are stopping at 150 seconds because the data is skewed towards shorter reading times. X is increased with a step of 10 seconds, e.g. 10s, 20s and so on. After deciding on the buckets time range, we assign ratings to each bucket. To start off, we associate the two buckets with the ratings 1 and 2. We run the BLC algorithm with 90%/10% train-test split on the ratings matrix, and we plot the performance metrics obtained from analysing the held-out data. This can be seen in Figure 3.4.

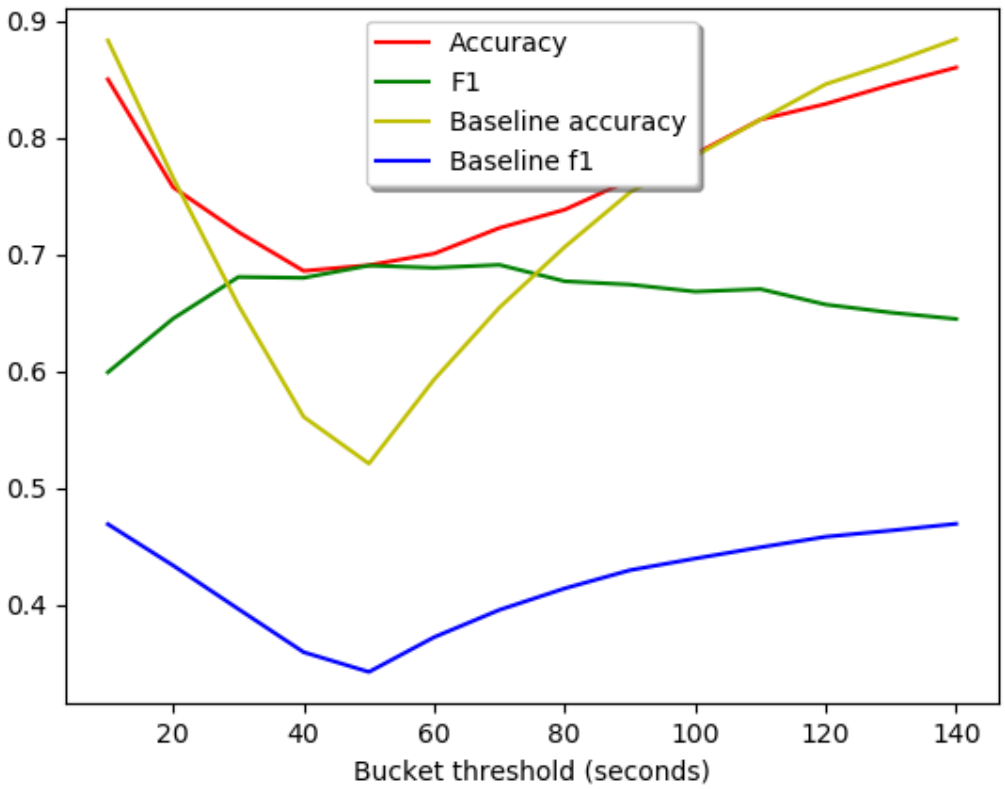

Figure 3.4: Performance analysis when choosing different thresholds for 2 buckets

To give even more weight to the higher reading times, we can change the values associated with the buckets. Figure 3.5 shows a comparison of the F1-score when using different ratings. It's important to note that before calculating any metrics, we need to clip the predictions to the interval desired (e.g. 1-10), as well as threshold the predictions (e.g. when

using ratings 1 & 10, any predicted value below 5 will be considered 1 and above 5 will be considered 10). Figure 3.5 shows that the results for the different ratings are fairly similar and that an optimal split for two buckets occurs at around 50 seconds.

We can analyse this in more detail by looking at the confusion matrix when splitting the buckets: (0s, 50s) and (50s, 894s) and using the ratings values 1 and 5.

$$\begin{pmatrix} 30987 & 16664 \\ 11958 & 31873 \end{pmatrix}$$

The confusion matrix tells us that in the case of rating "1", 30987 were predicted correctly, while 16664 were predicted as a "5". Meanwhile, 31873 values were predicted correctly as a "5", while 11958 were predicted as a "1". Overall, each class performs well, with the number of correct predictions almost triple the incorrect one (hence the 0.69 F1-score).

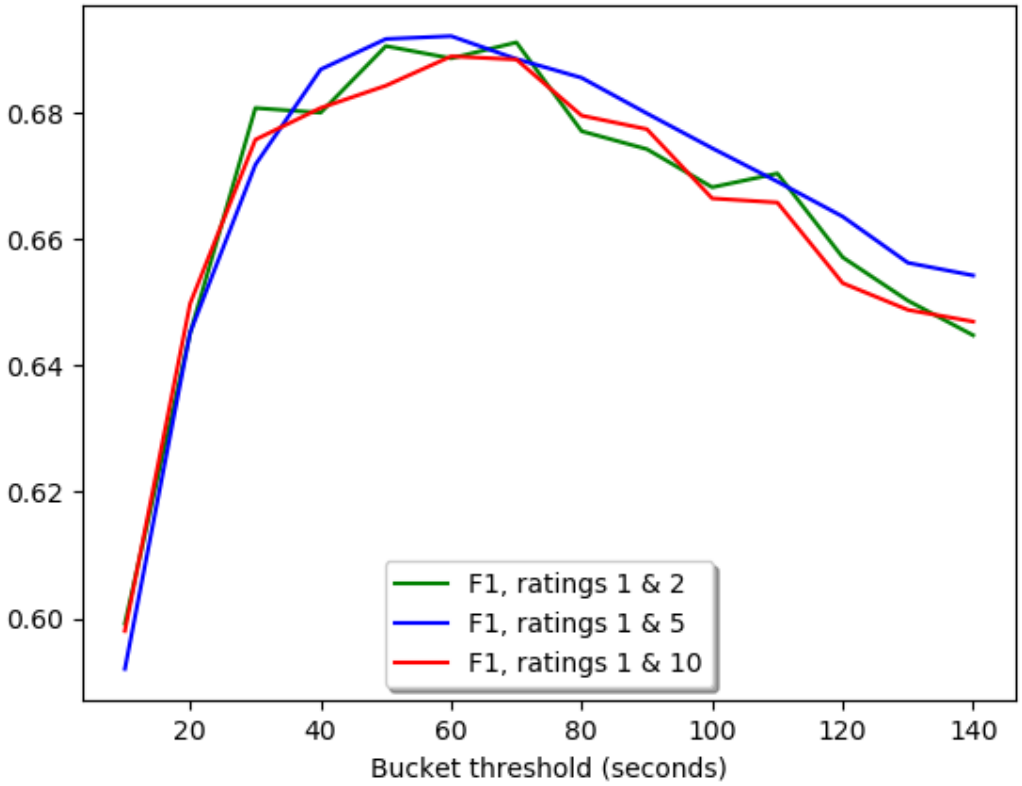

Figure 3.5: Performance analysis when associating the buckets with different ratings

We can go a step further and split the data into 3 buckets. One way in which we can do this is to keep the 50 seconds threshold found, and divide the other intervals to see which would allow creating a third bucket while preserving the accuracy.

If we divide the first bucket into 2, we will have the following 3 buckets: (0s, Xs), (Xs, 50s) , (50s, max time), where X is between 10s and 40s. We are using the same method as above to find X, trying out different values and calculating the accuracy metrics on the held-out data (90%/10% train-test split). The results can be seen in Figure 3.6.

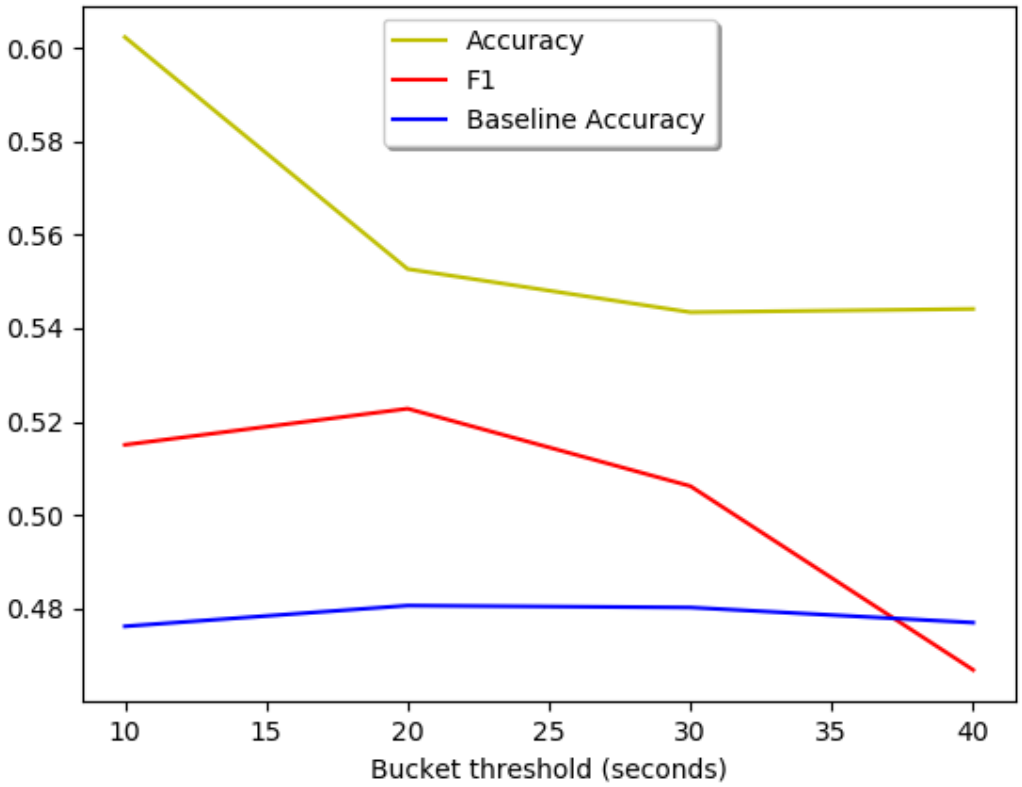

Figure 3.6: Performance analysis when choosing different thresholds in the (10,50) interval for 3 buckets

If we divide the second bucket into 2, we will have the following 3 buckets: (0s, 50s), (50s, Xs), (Xs, max time), where X is between 60s and 250s. Using the same method for finding X, the results can be seen in Figure 3.7

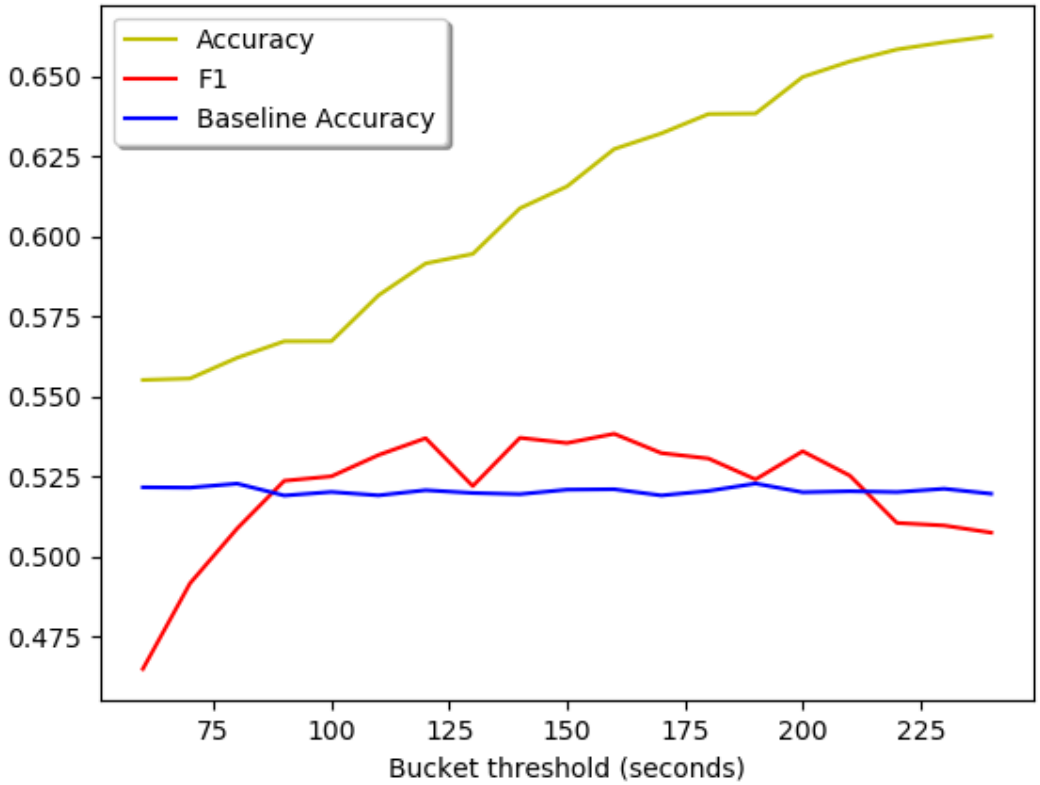

Figure 3.7: Performance analysis when choosing different thresholds in the (50,250) interval for 3 buckets

Whether it's splitting up the first bucket or the second one, the F1-score doesn't seem to rise above 0.54. This is how the confusion matrix looks like when X is 160s (took an example at random which gave an F1-score close to the best we achieved) and F1 = 0.534.

$$\begin{pmatrix} 33975 & 12692 & 927 \\ 12938 & 20981 & 1967 \\ 1194 & 4451 & 2068 \end{pmatrix}$$

From the confusion matrix, it is clear that the algorithm is not performing great when trying to identify the third class with only 2068 values correctly classified. However, it is doing doing well in the case of the other 2 classes.

The BLC output after applying it on the 2 variations of the dataset contains an estimation of the mean ratings and variance of each group and their corresponding items. These values can then be used in the Cluster-Based Bandits algorithm to determine whether the system can successfully learn about new user's preferences.

3.4.1 Hyperparameter tuning

As mentioned in the first section of this Chapter, the BLC algorithm includes several hyper parameters that can be tuned. The results above have been achieved by using a total number of 8 nyms, 10 latent features and a value L2=2 for regularization. The results didn't indicate any overfitting tendencies by the algorithm, however we did perform some Hyperparameter tuning to double check whether the results would change. The following combinations of hyperparameters have been used:

- Nyms values in [4,8,16,32]
- Latent features values in [4,8,19,12]
- L2 regularization values in [1,2,5,50,100]

The results using the different combinations of parameters remained rather constant, with negligible maximum difference of 0.3 in accuracy and f1 score, e.g. 0.70 vs. 0.67 when using 2 buckets for the ratings.

3.4.2 Transposing the ratings matrix

When arranging the matrix in the following format: (user, article) = rating, we are clustering the users, so that when a new user joins the system, we simply find the cluster they belong to and then recommend the articles we would recommend to the rest.

In order to consider the opposite problem, when a new item joins the system, we need to cluster based on the articles instead. This involves transposing the ratings matrix used by the BLC algorithm. However, when transposing the matrix from 267718 (users) x 9426 (items) to 9426 x 267718, the BLC algorithm is extremely slow. Thus, we need to thin out the users. This is done by removing users that only rate one item. There are a total of 105279 users that only rate a single item. Removing these yields the data described in Table 3.4.

Table 3.4: Norwegian news dataset statistics after removing the users who only rate one item.

Users	162439
Items	7467
User-Item ratings	1111501

Using the same steps and thresholds for the buckets as before, the accuracy and f1 remain very similar, as it can be seen in Table 3.5 .

In conclusion, after processing the Norwegian news dataset, we can apply the Cluster-Based Bandit algorithm on 4 different datasets. Two of these will be used when assessing the performance of the algorithm when a new user enters the system, while the other two can be used for new articles that are added to the portal.

Table 3.5: Accuracy and f1 after using BLC Matrix factorization on the transposed Ratings matrix.

	Accuracy				f1			
	Number of groups							
	4 nyms	8 nyms	16 nyms	32 nyms	4 nyms	8 nyms	16 nyms	32 nyms
2 buckets	0.66	0.67	0.67	0.67	0.66	0.67	0.67	0.67
3 buckets	0.59	0.59	0.59	0.59	0.48	0.48	0.48	0.50

The BLC output consists of the clusters description and the estimated mean rating and rating variance of the items in each cluster.

3.5 Alternate datasets

Using the Norwegian news dataset, we create ratings from view time in an attempt to understand whether the user liked or disliked the articles they read. There are also other ways to detect users' preferences and these include click through rates, reactions, sharing behaviour etc.

3.5.1 Twitter Dataset

Twitter is a micro-blogging service where registered users can write down their thoughts within short posts called tweets. They can also like and share (retweet) other people's tweets. While it is perceived as a social network, some argue that it's in fact a news media outlet. It has been shown that out of the trending topics, over 85% are related to news articles (16).

Consequently, we explored whether we can use Twitter data as feedback for our recommender system. In particular, retweeting a news article is a strong indicator that the person likes that article.

The dataset published by Harvard Dataverse (17) includes a list of tweet IDs that contain posts about news articles. Using the Twitter API (Application Programming Interface) [3], we can retrieve the ID of the author and the URL of the shared news article, Table 3.6. As these are historical tweets (posted or shared in 2018), the tweets retrieved using the API do not count towards any quota, so an unlimited number could be potentially fetched. However, it's important to note there is a 300 requests limit within a 15-minute interval. Thus, fetching all the tweets in the list could take several hours.

We created a matrix where the 2 dimensions were formed by the User IDs and the URLs and the value was either 1 if the user tweeted the article, or 0 if they didn't. We used BLC, as

[3] https://developer.twitter.com/en/docs

Table 3.6: Example of a Twitter entry that was fetched using the Twitter API

Twitter ID (used to retrieve the rest of the information)	1000002874020069376
User ID	204536811
Extended URL	http://www.cbc.ca/news/canada/windsor/green-party-wants-respect-for-more-than-just-its-environmental-reputation-1.4676992
Text of Tweet	Local @OntarioGreens candidates wants respect for more than just its environmental reputation https://t.co/JpRtG0Vt57

well as algorithms tailored for Implicit Feedback provided by the lenskit library[4]. There was a poor performance in both cases.

While Twitter data has been used successfully in recommender systems based on content-based filtering (18), the data we fetched cannot be used for a collaborative filtering approach. This is because collaborative filtering algorithms need to infer latent factors from the rating patterns. In our case, we only know whether a user has retweeted an article or not, but we don't know which articles they saw and chose not to retweet, thus we can't recommend new items to them in the absence of more data about their behaviour.

[4]https://lenskit.org/

4 Evaluation

The evaluation consists in comparing the accuracy and convergence times obtained using the Cluster Based Bandits against the Classification and Regression Tree (CART) algorithms described in the Background chapter. Having clustered the users, the BLC algorithm provides an estimation of the variances and mean ratings of each item in each group. For example, if we had 4 groups, we would have the variances for each item in each of the groups, that would be a matrix of the following size 4 (groups) x 9426 (items). The Cluster Based Bandit algorithm and the Decision Tree use this information to decide which items are the most valuable when deciding between clusters.

There are a few parameters for the Multi-Armed Bandit algorithm that can be configured. If a new user joins the system, we set out the number of items that we want to show to the user, this is also the depth of the decision tree that we're going to use. For the purpose of the evaluations, we use 25 items, but we show that fewer are needed for the algorithms to converge.

In order to make these measurements, we need to generate new users to be added to the system. One way of doing this is to split the data into training and testing data, new users can then be picked from the testing data. If we need a rating for an item that the user chosen has not rated, we can simply choose one more user and merge the ratings. The Cluster Based Bandits paper has shown that instead of doing this longer process, we can also just generate new item ratings for the new users by randomly selecting ratings with the same mean and variance as the training data without affecting the performance of the algorithm.

At each step of the algorithm, a group is estimated as the one the user belongs to. Having artificially generated the users, we know the group they are part of, so we calculate the accuracy value as the number of times the correct group has been estimated by the algorithm. Convergence time is calculated by looking at the number of items that were rated before the regret reaches 80% of its final value. The regret is the sum of the differences between the rating of the item we're asking the user to rate and the unrated item that the user would rate the highest. All the values are calculated using 1000 users.

In addition to comparing the Cluster Based Bandits to the Decision Tree results, we are also going to look at the Cluster Based Bandits results without the initial exploration phase. This showcases how having the initial exploration phase can lead to convergence much faster.

The exact same process would take place for the new items, instead of new users.

4.1 New user

This chapter analyses the results obtained when the cold-start problem involves a new user joining the system.

4.1.1 Two buckets

First, we will take a look at the results obtained when splitting the reading times in 2 buckets using the 50 seconds thresholds.

Table 4.1 contains the overall results (mean accuracy and convergence time) obtained when clustering the users in 4,8,16 and 32 groups. The Decision Tree baseline chosen is a really strong baseline, so it's no surprise to see that for some of the groups, it actually outperforms the Cluster-Based Bandit. The following paragraphs will look into the details of some of the nyms that showed interesting results.

When clustering the users into 4 groups, the accuracy of finding the correct group is 1.00, which means that both algorithms can successfully find the group the new user belongs to. However, the mean convergence time is actually better for the Decision Tree. If we take a closer look at Figure 4.1 we can see that the Cluster Bandit algorithm takes much longer to converge when the user is in the first group as opposed to the 4th group.

Table 4.1: Cluster Based Bandits (CB) vs. Decision Tree (DT) vs. Cluster Based Bandits without the initial exploration phase (CB-) mean accuracy and convergence time when a new user enters the system and the reading times are split into 2 buckets.

	Mean Accuracy				Mean Convergence time			
	Number of groups							
	4 nyms	8 nyms	16 nyms	32 nyms	4 nyms	8 nyms	16 nyms	32 nyms
CB	1.00	0.96	0.95	0.85	6.23	5.63	7.26	9.56
DT	1.00	0.97	0.93	0.77	2.83	9.33	7.60	9.67
CB-	1.00	0.96	0.90	0.77	-	-	6.27	8.18

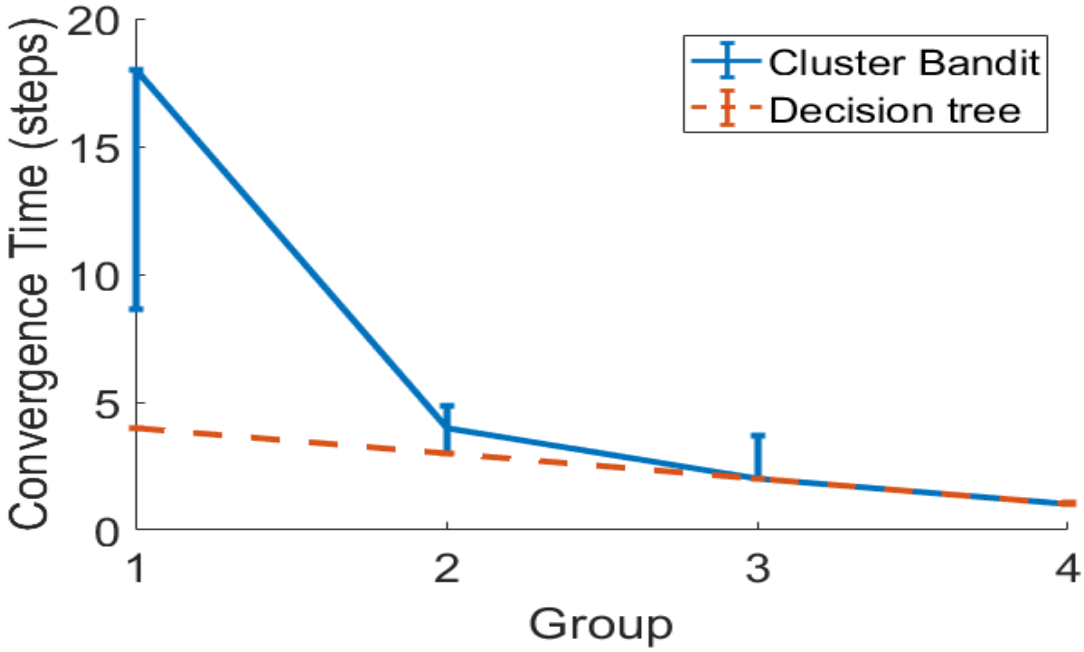

Figure 4.1: Convergence time for 4 nyms

The Cluster-Based Bandit algorithm relies on finding distinguishers to achieve a fast convergence time. To reiterate, a distinguisher item v is the item that helps distinguish between 2 groups, g and h, because it has the following properties:

- The ratings are reliable, i.e. the variance of the item v in group g is small.
- The mean rating of the item in group g is very different from the mean rating of the item in group h.

A possible explanation for why Group 1 converges much slower than Group 4 could be related to the values of the distinguishers items found for each group. To investigate further, we can plot a histogram of the distinguishers identified in each group, see Figure 4.2.

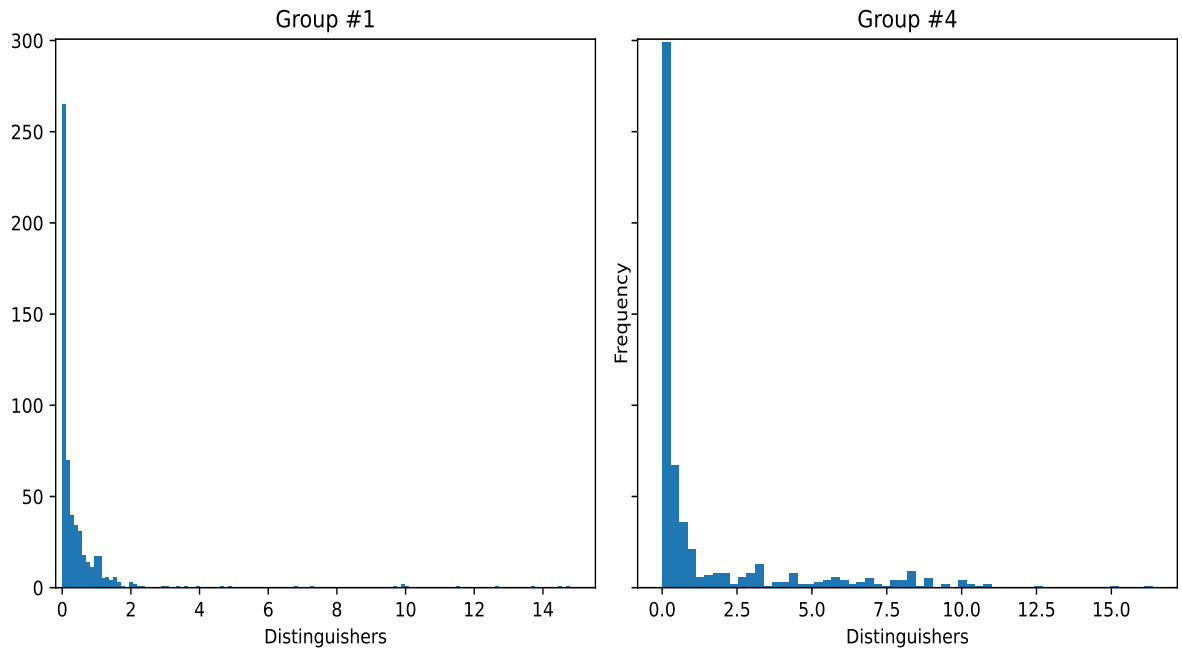

Figure 4.2: Comparison of distinguishers between group 1 and group 4

The fast learning happens when the user is asked to rate the items with the highest gamma values (the largest distinguishers). The histogram plot in Figure 4.2 indicates that Group 4 has better distinguishers, with higher values and this can be used to explain why the learning is so much faster.

When dividing the users into 8,16 and 32 nyms the results, both accuracy and convergence time are much closer in value. As the results are fairly similar, we will only investigate the 16 nyms setup in more detail. Figure 4.3 contains 4 graphs describing the accuracy achieved by individual groups, the amount of steps it takes to converge and the accuracy and regret over time in the case of a randomly picked group. Overall, results are very close to each other, both in accuracy and in convergence time. We can see how the Cluster Bandit without exploration achieves the lowest regret over time. This is due to skipping the initial phase, where the user is asked to rate distinguishers. Distinguishers are not the optimal items that would be rated the highest by the user, therefore there is an extra cost associated with exploring.

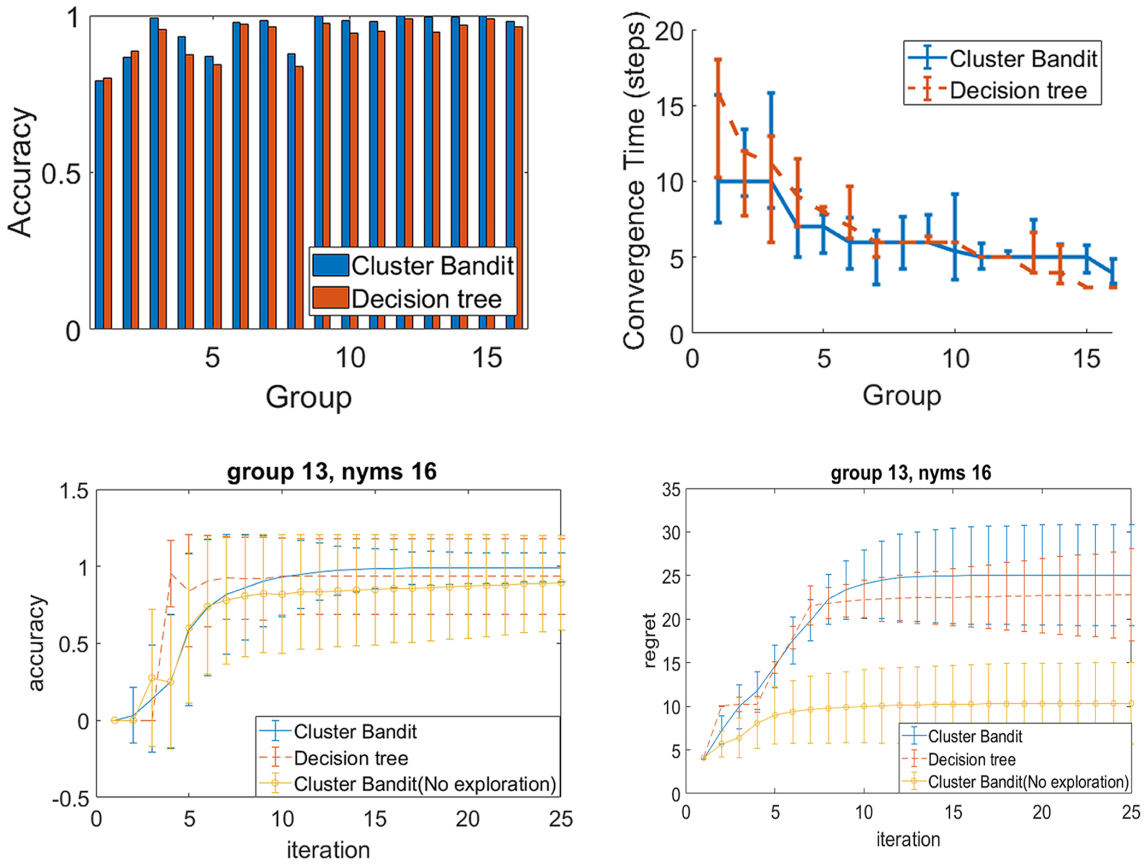

Figure 4.3: Overall results for 16 nyms, 2 buckets

4.1.2 Three buckets

When using 3 buckets, the results in Table 4.2 indicate that the Cluster-Based Bandits is doing a much better job at detecting the correct group, when it comes to both accuracy and number of steps it requires to do so. Without the initial exploration phase, the Cluster Based Bandit doesn't converge in 25 steps, which shows the benefit of having this extra step in the algorithm.

Having 3 buckets means that we now have 3 ratings (1,2,3) used as feedback for the items in the dataset. As users don't generally give the same ratings to an item, having more ratings increases the intra-cluster variability. This is described by the Cluster Based Bandit paper as introducing noise to the data. Decision Trees are known to be less accurate in the presence of such noise. This could be an explanation for why the accuracy of CB is much better in the case of 3 buckets.

The results are similar between the groups again, so we will only analyse the setup that consists of 16 nyms in more detail. The top two graphs in Figure 4.4 show that both the accuracy and the convergence times are better when using the Cluster-Based Bandit. Up to a 20% increase in accuracy can be seen for certain groups. The bottom graphs show the

Table 4.2: Cluster Based Bandits (CB) vs. Decision Tree (DT) vs. Cluster Based Bandits without the initial exploration phase (CB-) results when a new user enters the system and the reading times are split into 3 buckets

	Accuracy				Mean Convergence time			
	Number of groups							
	4 nyms	8 nyms	16 nyms	32 nyms	4 nyms	8 nyms	16 nyms	32 nyms
CB	1.00	0.99	0.91	0.96	3.74	6.26	6.48	7.41
DT	0.96	0.80	0.74	0.73	4.92	7.98	8.28	8.76
CB-	0.99	0.98	0.94	0.90	-	-	-	-

accuracy and regret over time for a randomly selected group, in this case group number 3. Group 3 converges after around 10 items shown to the user and it can be seen how the CB accuracy surpasses the Decision Tree accuracy after 10 iterations. The regret is also much higher for the Decision Tree. Both algorithms seem to be performing terribly in the case of Group number 6, achieving 0 accuracy. Group 6 contains 31682 users and there are 540 items that have at least 5 ratings. Comparing the mean ratings and variances of this group with the others didn't bring to light any obvious conclusion as to why the accuracy is 0. Further investigation is needed to check whether this is a flaw of the algorithm or there are issues with the data.

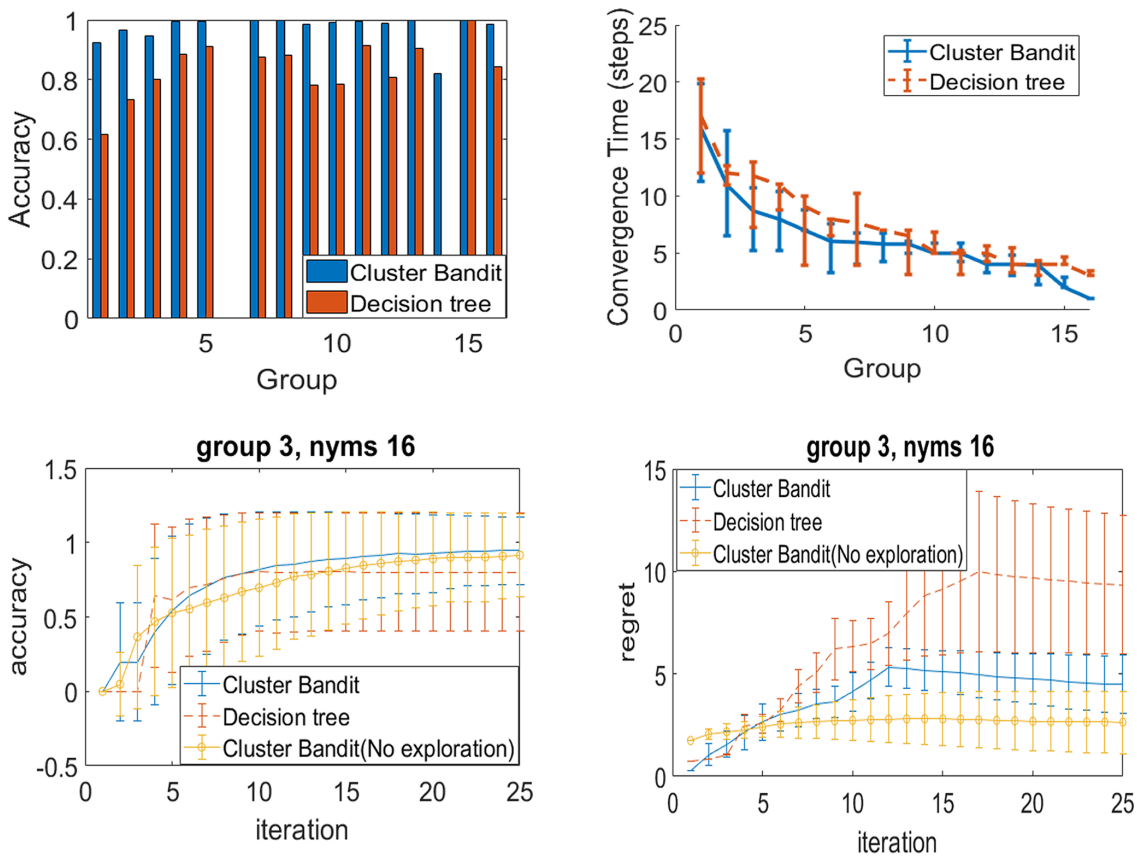

Figure 4.4: Overall results for 16 nyms, 3 buckets

Nevertheless, considering the results for the other groups in this setup, as well as the performance indicated when using fewer or more nyms, it is clear that the Cluster-Based Bandit can be used successfully to learn about a new user in a News recommendation system. At the same time, the algorithm is outperforming the Decision Tree implementation.

4.2 New item

This chapter discusses the results obtained when the cold-start problem involves a new item entering the system, instead of a new user. As described in the Design chapter, the Ratings matrix for this setup is transposed, meaning instead of clustering the users, we are now clustering the articles. A challenge with this setup is that the users of the system are very few compared to the amount of items and transposing the matrix makes the clustering process really difficult for 2 reasons:

- Not enough articles to be clustered in 16/32 nyms.
- The estimation of the variances and mean ratings produced by BLC are not as accurate.

The first point is straightforward, when clustering only 7467 articles, it's obvious that some clusters will have very few articles if we aim to divide the articles as much as possible.

When clustering the users, the BLC algorithm estimates the variance and the mean ratings of each item by the users in their respective clusters. In the new setup, the BLC estimates the variance and mean ratings that the users have given to the articles in the clusters. The Ratings matrix is sparse, in the Design chapter we've seen that an article has been viewed on average 130 times, but a user has read on average only 5 articles. So depending on the amount of articles assigned to each cluster, the estimations produced by BLC are more or less accurate.

To partially overcome this problem, the variances are increased with a value that is inversely proportional to the amount of ratings that make up the estimations (num in the following formula):

$$variance = variance + 0.5 * sqrt(log(1/0.2)./num);$$

The effect of this change is that the estimated variance for the users who only rated very few items will be very high and so these users will be unlikely to be used as distinguishers between groups. Due to the small number of articles to be clustered, we are only going to investigate the setups for 4 and 8 nyms.

4.2.1 Two buckets

Table 4.3 shows the mean accuracy and mean convergence time obtained when splitting the reading times in 2 buckets using the 50s threshold. The results indicate that it's enough to ask an average of 5 users about a new article in order to determine its cluster.

Table 4.3: Cluster Based Bandits (CB) vs. Decision Tree (DT) vs. Cluster Based Bandits without the initial exploration phase (CB-) results when a new item is introduced to the system and the reading times are split into 2 buckets.

	Mean Accuracy		Mean Convergence time	
	Number of groups			
	4 nyms	8 nyms	4 nyms	8 nyms
CB	0.95	0.94	4.34	6.18
DT	0.81	0.81	6.99	8.84
CB-	0.92	0.79	-	-

An accuracy breakdown can be seen in Figure 4.5. Figure 4.5 (b) shows only 5 groups, even though we tried to split the articles in 8 groups. The Cluster-Based Bandit was only applied on 5 groups due to the constraints applied with regards to the number of articles that are in a particular cluster. The estimations of the mean and variances of the ratings were not good enough to produce meaningful results for the rest of the clusters because there were fewer than 50 users giving more than 2 ratings to the articles in those clusters.

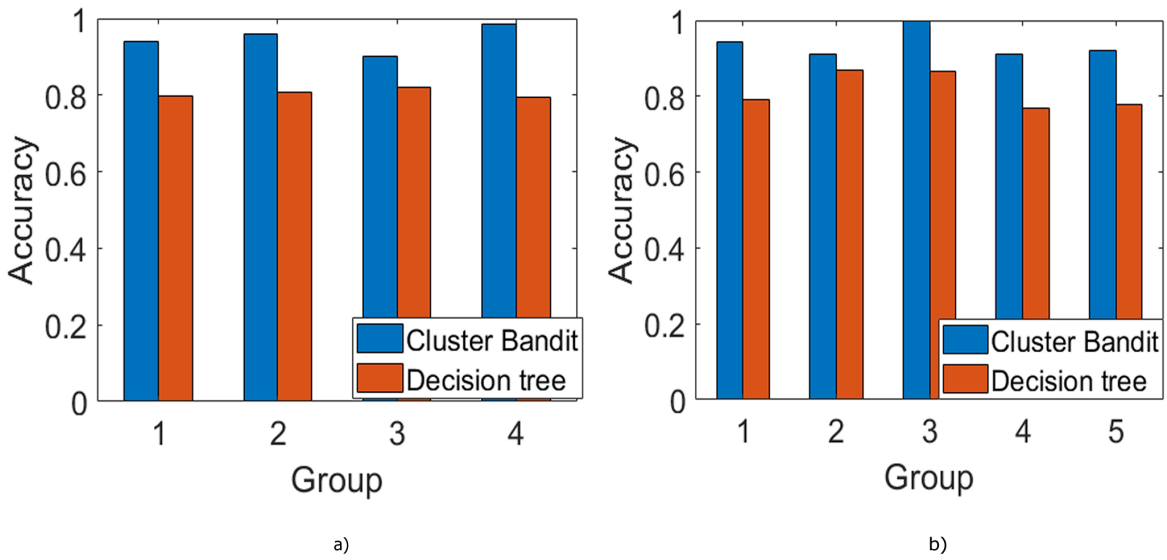

Figure 4.5: Accuracy breakdown for 4 and 8 nyms, 2 buckets.

4.2.2 Three buckets

Table 4.4 presents the mean accuracy and mean convergence time obtained when splitting the reading times into 3 buckets using the 50s & 160s thresholds.

Table 4.4: Cluster Based Bandits (CB) vs. Decision Tree (DT) vs. Cluster Based Bandits without the initial exploration phase (CB-) results when a new item is introduced to the system and the reading times are split into 3 buckets.

	Mean Accuracy		Mean Convergence time	
	Number of groups			
	4 nyms	8 nyms	4 nyms	8 nyms
CB	0.99	1.00	3.94	2.80
DT	0.96	0.71	5.78	7.84
CB-	0.96	0.96	-	-

An accuracy breakdown is presented in Figure 4.6. Figure 4.6 (a) corresponds to the 4 nyms setup and Figure 4.6 (b) to the 8 nyms setup. While the accuracy remains high for all the groups when clustering the users into 4 nyms, this is not the case for the 8 nyms setup. Attempting to divide the users into 8 nyms means the Cluster-Based Bandits ends up only getting evaluated on 3 nyms because the rest do not satisfy the conditions described earlier with respect to the number of articles and number of ratings available for each cluster.

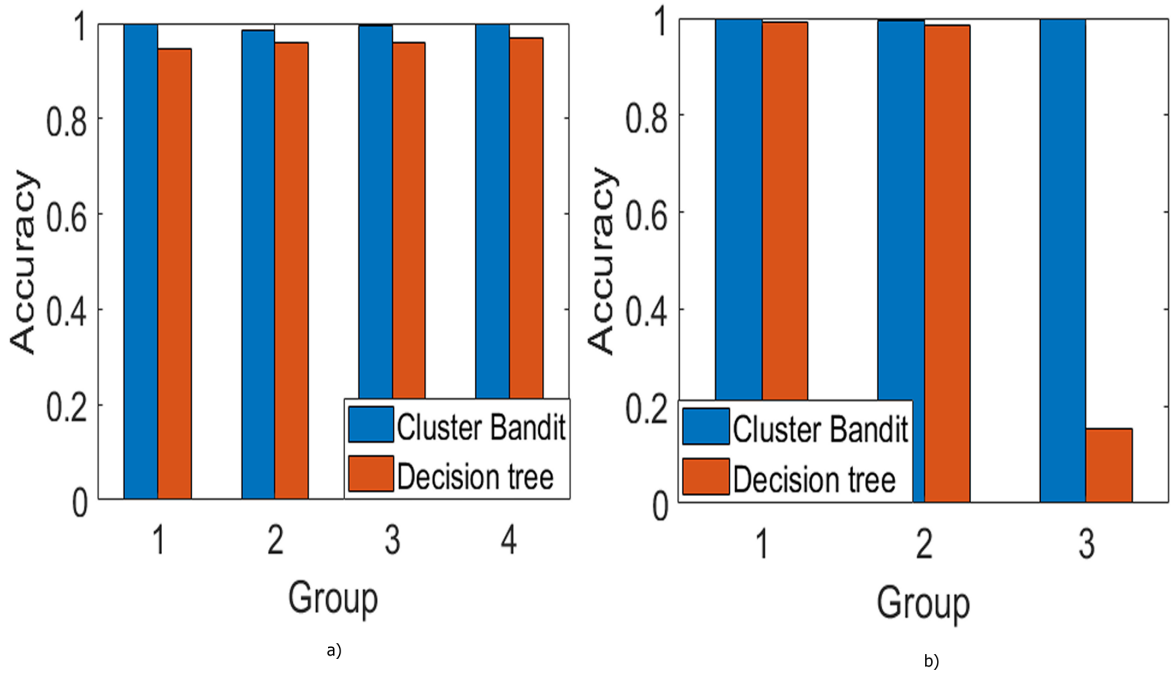

Figure 4.6: Accuracy breakdown for 4 and 8 nyms, 3 buckets.

To show this in more detail, we can look at the distribution of articles per cluster. When using BLC, each cluster contains the amount of articles described in Table 4.5.

In order to have some confidence in the mean rating and variances given for these articles, we choose to only look at the nyms where there are at least 50 users that have read more than 2 articles. Table 4.6 shows how this condition leads to only using the Cluster-Based

Table 4.5: Number of items in each cluster, 8 nyms, 3 buckets

Cluster	1	2	3	4	5	6	7	8
Number of items	2943.0	1327.0	1850.0	401.0	324.0	177.0	382.0	63.0

Bandits on 3 nyms instead of 8.

Table 4.6: Number of users in each cluster satisfying the Cluster-Based Bandits conditions

Cluster	1	2	3	4	5	6	7	8
Number of users	86400	9582	28741	32	35	16	30	3

Nevertheless, the Cluster-Based Bandit seems to maintain its advantage over the Decision Tree in both accuracy and convergence time, which indicates that this type of algorithm can also be used when the cold-start problem involves introducing a new item in the recommendation system. However, further work is needed to determine the confidence of the results obtained.

5 Future Work

5.1 Data improvements

There are 2 improvements that could be made to the way the features are engineered:

- Distinguish between subscribers and non-subscribers users.
- Normalize the time spent on an article with the amount it takes to read the full article. Unfortunately, this information was not present in the version of the dataset that's readily available online.

The first suggestion could improve the recommendation system a lot because it would densify the Ratings matrix as the number of users who are subscribed to a newsletter is considerably smaller than the number of total users surfing a news portal. A subscriber is also more likely to read more articles from the portal which would also improve the setup for new items being introduced to the system.

Instead of relying on a brute-force approach to split the seconds spent on an article into buckets and then assign ratings, we could compare this information with the amount it takes to actually read the entire piece. Then if an article takes on average 3 minutes to read, and a viewer spent roughly 3 minutes on it, we could assign the highest rating for this (user, article) pair.

5.2 Cluster-Based Bandits caveats

When it comes to the Cluster-Based bandit algorithm, one aspect that has to be addressed is the confidence in the mean ratings and variances used. The mean ratings and variances are estimations and they largely depend upon the amount of users in a group and the minimum number of ratings for an item. When choosing the threshold for which groups to consider for the algorithm (e.g. whether the group has more than 50 items with at least 4 ratings), the choices in both the setup of this dissertation and the other studies conducted in the Cluster-Based Bandit paper are rather arbitrary. Further work is needed to come up with a more systematic way of deciding these thresholds.

The user ratings of the items shown are generated from the data to simulate reality. However, the Cluster-Based Bandits algorithm doesn't model a "don't know" option for the items that are presented to the users, i.e. it always expects a rating. This is definitely possible in reality as well, if we asked the users to read the articles before rating them in the case they were unfamiliar with them. However, when applying these solutions to a real-world scenario, e.g. when signing up someone for a service, other aspects need to be taken into consideration. These include the user effort required for this sign-up process. Even if the algorithm converges really quickly, i.e. only 5 items necessary for assigning the user to a cluster, if these items were articles the user has never seen before and they have to spend time reading them in order to provide ratings, the process may seem too tedious.

However, the reverse setup would probably work pretty well, as we would only need to show a new article to fewer than 10 users to decide which other readers on the platform are going to be interested in reading about it.

5.3 Other thoughts and ideas

Collaborative filtering is very similar to a black box, where abstract features are inferred by the algorithm from the ratings patterns. If we knew the topics and the content of the articles, we could analyse the recommendations made by the system to see if the recommendations belong to the same topic and whether the features extracted from the ratings matrix are largely related to the categories the articles fall in. Whereas Netflix may ask the user to rate specific movies on a scale from 1 to 5 when joining their service, some news websites or applications ask the user about the topics they are interested in instead (e.g. World News, Environment, Technology etc.). I believe the way news are consumed versus music or movies is slightly different. While it's true that in all categories, new items and trending items are of interest, users are more likely to listen to an old song or watch an older movie than they are to read about some news article that refers to events from the previous years.

When learning about a user's preferences, either through content-based filtering or collaborative filtering, it's also important to consider the time component. In the case of the dataset used, the behaviour analysis has been conducted over the course of a week. However, even if a news portal has more data at its disposal, it has to consider that users' interests change rapidly over time, particularly in the news recommendation systems, where a trending topic could emerge on a daily basis.

6 Conclusion

6.1 Dissertation objectives

This section describes to what extent the dissertation objectives were fulfilled.

6.1.1 Finding an appropriate news dataset

This research objective has been achieved by using data preprocessing and feature engineering on the Adressa News dataset. While being able to successfully split the reading times of the users into buckets and to associate ratings with these buckets, the feature engineering part can be improved by also taking into calculation the average time it takes to read an article, as described in the future work section.

Another dataset containing Tweet entries about news articles has been explored, but the data turned out to be ill-suited for the Collaborative filtering recommendation approach.

6.1.2 Clustering

User clustering has been achieved by using BLC matrix factorization, although another clustering algorithm could be used. BLC adds the extra benefit of privacy when clustering the users, but this is not as relevant when the setup is reversed and the articles are the ones being clustered. The BLC algorithm is optimized to perform the matrix factorization in a very fast way when the number of users is larger than the number of articles. When transposing the matrix, BLC takes a really long time to generate the clusters, so another clustering algorithm may be more suitable for this task.

The clustering of both the users and the articles has been carried out with a high degree of accuracy and a good F1 score (roughly 0.7 for 2 ratings and 0.6 for 3 ratings).

6.1.3 Finding the cluster of a new user

This dissertation objective has been achieved by proving that the Cluster-Based Bandits can successfully find the cluster of a new user with 0.90 accuracy or more. In most cases, the

Multi-Armed Bandits is also outperforming the Decision Tree baseline with regards to the number of items that need to be shown to the user before finding the cluster.

6.1.4 Finding the cluster of a new item

I consider this objective to be partially achieved. The results of the Cluster-Based Bandits on the 4 and 8 nyms setup seem promising, but in order to have a higher degree of confidence in these results, we would need a denser Ratings matrix. We would also need to define a more systematic way to decide which groups are good enough to consider for the algorithm, i.e. if a group doesn't have enough ratings or users it should be discarded as we don't have confidence in the mean ratings and variances estimated for these groups.

6.2 Closing remarks

When surfing the Internet, we are constantly bombarded with breaking news information. Filtering through this information has become almost impossible and this is why recommender systems look so appealing to online platforms that aim to improve the user experience. The recommendation strategies employed work best with explicit feedback. However, unlike movies or music, users are less likely to explicitly rate news articles and this poses a challenge when trying to learn more about the users and the items in the system. Even when gathering enough information about the current users of the system, there are always new users and items added. When using a collaborative filtering recommendation approach, the items and users of the system can be grouped into clusters based on similarities. This research has shown that using the Cluster-Based Bandits algorithm allows the system to quickly learn about any new elements introduced to the system (users or items) and to assign them to their corresponding clusters, thus successfully tackling the cold-start problem.

Bibliography

[1] Nadav Golbandi, Yehuda Koren, and Ronny Lempel. Adaptive bootstrapping of recommender systems using decision trees. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 595–604, 2011.

[2] James Bennett, Stan Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. Citeseer, 2007.

[3] Pasquale Lops, Marco De Gemmis, and Giovanni Semeraro. Content-based recommender systems: State of the art and trends. *Recommender systems handbook*, pages 73–105, 2011.

[4] Robin Van Meteren and Maarten Van Someren. Using content-based filtering for recommendation. In *Proceedings of the machine learning in the new information age: MLnet/ECML2000 workshop*, volume 30, pages 47–56, 2000.

[5] Jae-wook Ahn, Peter Brusilovsky, Jonathan Grady, Daqing He, and Sue Yeon Syn. Open user profiles for adaptive news systems: help or harm? In *Proceedings of the 16th international conference on World Wide Web*, pages 11–20, 2007.

[6] David Goldberg, David Nichols, Brian M Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[7] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[8] Al Mamunur Rashid, Istvan Albert, Dan Cosley, Shyong K Lam, Sean M McNee, Joseph A Konstan, and John Riedl. Getting to know you: learning new user preferences in recommender systems. In *Proceedings of the 7th international conference on Intelligent user interfaces*, pages 127–134, 2002.

[9] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[10] Herbert Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535, 1952.

[11] Joannes Vermorel and Mehryar Mohri. Multi-armed bandit algorithms and empirical evaluation. In *European conference on machine learning*, pages 437–448. Springer, 2005.

[12] Douglas Leith Sulthana Shams, Daron Anderson. Cluster-based bandits: Fast cold-start for recommender system new users. In *Proc SIGIR*, 2021.

[13] Alessandro Checco, Giuseppe Bianchi, and Douglas J. Leith. Blc: Private matrix factorization recommenders via automatic group learning. *ACM Trans. Priv. Secur.*, 20 (2), May 2017. ISSN 2471-2566. doi: 10.1145/3041760. URL https://doi.org/10.1145/3041760.

[14] Ryan Singel. Netflix cancels recommendation contest after privacy lawsuit. *Wired Magazine*, 2010.

[15] Jon Atle Gulla, Lemei Zhang, Peng Liu, Özlem Özgöbek, and Xiaomeng Su. The adressa dataset for news recommendation. In *Proceedings of the International Conference on Web Intelligence*, WI '17, page 1042–1048, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349512. doi: 10.1145/3106426.3109436. URL https://doi.org/10.1145/3106426.3109436.

[16] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is twitter, a social network or a news media? In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, page 591–600, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605587998. doi: 10.1145/1772690.1772751. URL https://doi.org/10.1145/1772690.1772751.

[17] Giovanni Brena, Marco Brambilla, Stefano Ceri, Marco Di Giovanni, Francesco Pierri, and Giorgia Ramponi. News Sharing User Behaviour on Twitter: A Comprehensive Data Collection of News Articles and Social Interactions, 2019. URL https://doi.org/10.7910/DVN/5XRZLH.

[18] Owen Phelan, Kevin McCarthy, and Barry Smyth. Using twitter to recommend real-time topical news. In *Proceedings of the Third ACM Conference on Recommender Systems*, RecSys '09, page 385–388, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605584355. doi: 10.1145/1639714.1639794. URL https://doi.org/10.1145/1639714.1639794.