ADAPTIVE HYBRID COLLABORATIVE FILTERING RECOMMENDATION SYSTEM (AHCF)

BY

ROBERT AGBOYI

(10402888)

THIS THESIS IS SUBMITTED TO THE UNIVERSITY OF GHANA, LEGON IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD OF MPHIL IN COMPUTER ENGINEERING DEGREE

DEPARTMENT OF COMPUTER ENGINEERING

SCHOOL OF ENGINEERING SCIENCES

UNIVERSITY OF GHANA, LEGON

JULY, 2019

DECLARATION

I, Robert Agboyi, hereby declare that, this thesis document except where indicated by referencing, is my own work carried out under supervision in the Department of Computer Engineering, Faculty of Engineering Sciences, University of Ghana, Legon. I further declare that, this thesis, either in whole or in part, has not been presented for another degree in this University or elsewhere.

…….…… ……

Robert Agboyi Date

(Student)

…….. ……..…….

Dr. Robert A. Sowah Date

(Principal Supervisor)

ADAPTIVE HYBRID COLLABORATIVE FILTERING RECOMMENDATION SYSTEM (AHCF)

ABSTRACT

Recommendation systems play a vital role in boosting the organization's profit, especially for e-commerce platforms such as Amazon. These systems focus on targeting specific products to users and predicting user preferences and interests. However, recommendation systems are plagued with many challenges, such as adapting them to changes in user preferences and taste, and the effectiveness of recommendations made also determines the ability to retain and engage new users, as new user conversion to clients. This thesis proposes to use an adaptive hybrid collaborative approach to making recommendations to users. Four algorithms are combined: the Alternate Least Squares (ALS), KMeans clustering, Latent Dirichlet Allocation (LDA) and KMeans streaming. The recommender engine developed is in itself a multi-hybrid system as it not only combines four (4) algorithms but also combines the collaborative technique and content-based techniques of making a recommendation. Thus, the approach adopted can be used on datasets that contain rating information, textual descriptions or both. Three servers are leveraged in the implementation, consisting of the Scala server, PHP and Angular JS server and the MySQL database server for the storage of the results from the recommender engine. Various industry-standard metrics are adopted for the individual algorithms in addition to their computational times. These metrics include Root Mean Square Error(RMSE) for the ALS, Within Cluster Sum of Squares(WCSS) for KMeans, Log Perplexity and Log-Likelihood in the LDA. The memory estimates footprints and computational time on retraining the model are recorded for the KMeans streaming. The recommender engine is tested primarily on the 100K and 1M movieLens datasets and some portions of the 20M dataset are used.

The implementation is compared with benchmark recommender algorithms via GitHub and existing offline implementations. In terms of retraining, the Adaptive Hybrid Collaborative Filtering Recommendation System(AHCF) developed improves a recommendation's computational time concerning the offline model by 50%. The AHCF has an accuracy measure of 0.88-3.0 on RMSE values for the chosen datasets on increasing rank but less than 8 for 5 other datasets adopted. The other datasets range from restaurant datasets, anime, dating datasets, books and e-commerce. These results are taken for the 1M and 100K datasets.

The unique contributions made in this research include combining multiple algorithms into one recommender engine that leverages textual and rating information at the same time. Improvements in computational efficiency as against offline models that are designed for a real-time update of recommendations by half on retraining. The generic nature of the algorithm also makes it useful to be used in many domains that leverage informative text and rating information. The model is also open source and available to all users.

In a nutshell, the research embraces the efficiency of updating user preferences in real-time and making personalized recommendations by adapting to user preferences over short time intervals.

ACKNOWLEDGEMENT

This work is dedicated to GOD ALMIGHTY, my friends and family for their support and encouragement, to Bernard Kuditchar for the tremendous input, suggestions, and encouragement and Dr. Robert Sowah for his invaluable guidance, contributions, and encouragements towards the successful completion of this research as well as members of the Department of Computer Engineering for the significant inputs to the writing of this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ALS- Alternate Least Squares

LDA- Latent Dirichlet Allocation

WCSS- Within Cluster Sum of Squares

MLE- Maximum Log-likelihood Estimation

RMSE-Root Mean Square Error

RS- Recommendation System

ML- Machine Learning

CF- Collaborative filtering

CT- Computational Time

CTU-Computational Time for User clustering

CTM-Computational Time for Movie Clustering

HC-Hybrid Computational Time

LL- Logarithmic Likelihood

LP- Log Perplexity

NMF- Non-negative matrix factorization

SVD- Singular Value Decomposition

K-NN- K Nearest Neighbors

TF-IDF-Term Frequency-Inverse Document Frequency

CHAPTER 1

INTRODUCTION

1.0 Introduction and Background

Recommender systems have gradually become an approach to leverage the product and user relationship gap. Day in day out, clients or customers use online services to find new items or to purchase items based on several variables of interest. User interests may span what is popular, trending, or preferences of friends or people with similar tastes, as well as other implicit and explicit characteristics. The interest of users is not always predictable and may change abruptly with very little room for an explanation; hence, the concept of recommender systems. Recommender systems employ algorithms that are primarily dependent on machine learning to provide items (such as books, music, transportation, places, etc.) for users [1]. Recommendations for users are usually based on user or recommended item information.

Amazon, Netflix, Facebook, Yelp, and many others are increasingly dependent on recommender systems to target specific products and content to users. Movies and music vendors usually gain a significant advantage in enhancing user interaction and engagement on their systems. This is made easy through monitoring user ratings given, click-through rate, the relevance of the recommended item to the user and the accuracy of the prediction among others.

In systems with a large number of users and a wide range of items, recommendation systems become an invaluable tool considering the heterogeneous nature of user preferences and the many choices available. Coupled with that is the regular evolving nature of the system with regards to adding new items or users. Businesses have tremendously benefitted from implementing and analyzing recommender system results to enhance their profitability[2] better.

Recommender systems fall into three categories: collaborative, content-based, and hybrid filtering.

The collaborative approach makes recommendations based on some computed similarity measures among users or items. The content-based approach depends on item metadata to establish the relationship between items for recommendation. The third, hybrid filtering approach, is a diverse combination of collaborative and content-based approaches.

While a hybrid approach to collaborative filtering focuses on combining the varied approaches to collaborative filtering, an adaptive approach to collaborative filtering focuses on changing the dynamics of user preference. Due to the unpredictable nature of user preferences on a system and the abrupt nature with which the preferences change, there is the need for these systems to adjust properly using some algorithms and the need for diverse combinations of the collaborative and content-based approaches (hybrid), thus introducing the concept of adaptive hybrid collaborative recommender systems.

Recommender systems are plagued with computational and modelling problems with the advent of making successful recommendations. The Netflix (DVD Rental Company) prize challenge set the perfect tone for the advancement of research in recommendation systems, having the collaborative filtering(CF) approach gaining much attention and use in many domains[3].

CF approaches include model-based (using data mining techniques in making a recommendation), memory-based (using explicit data such as ratings to make recommendations), and hybrid approaches (combining the model and memory-based approaches).

Many approaches have been proposed to address these issues, ranging from clustering techniques, feature-based recommendations, and hybrid techniques, but the issues persist.

In the feature-based models, feature representation is scarcely or partly available due to the manual method of providing descriptions. Most recommender systems automate the feature extraction

based on tags, reviews, comments, and ratings, which introduces a curse of dimensionality which requires dimensionality reduction techniques.

In practice, most recommender systems are based on offline implementations, which give a slow time scale of updating the system, irrespective of the velocity of data coming in from time to time. This thesis proposes a novel Hybrid CF model susceptible to user preference changes that addresses scalability, sparsity, and extensively tackles the cold-start problem. Specifically, the ALS(Alternate Least Square) model, KMeans models and LDA (Latent Dirichlet Allocation) topic model take advantage of ratings, tags and textual descriptions for generating a recommendation. On the dimensionality reduction front, the use of the hidden topics automatically reduces the vastness of vocabulary of textual descriptions. This harnesses the preference heterogeneity against rich user-generated content representation in handling cold-start scenarios.

The model is also, an extension of literature on a hybrid distributed collaborative filtering recommender engine using Apache Spark[4] and a later improvement on it [5]. The use of the LDA topic model is distinct in its merging of all available textual contexts in the movie domain used for generating K hidden topics. The infusion of adaptation to changing dynamics of user preference by introducing the KMeans streaming further broadens the efficiency of the proposed model.

The model amplifies many insights on the preferences of users. A further demonstration of the relevance of the model to recommender systems is in generating recommendations relying on varied information sets to support a personalized recommendation.

1.1 Problem Statement

The significance of recommendation systems is drawn in its ability to help in such situations where it is difficult to determine user choices. Based on observations made on human behavior, especially

in situations where multiple choices and options are available, the difficulty associated with making such choices requires the use of adaptive and hybrid techniques to making recommendations for a user [6]. Thus Recommender systems in all domains are intended to make accurate suggestions of items or products to users/clients on demand and to provide new users with readily available suggestions of products based on specific criteria such as preferences, geographical location, demographics and social data among others. In the online movie domain, recommender systems are intended to enhance user engagement and user experience. The usefulness of recommender systems in marketing and management cannot be undermined as it has enhanced many organization's profits, especially in the e-commerce industry.

These systems are plagued with many challenges, such as scalability, sparsity, and cold-start. Challenges peculiar to CF techniques are the coverage problem (the percentage of items for which recommendations can be made) [7] and the following underlying problems:

1. Sparsity- Most databases are sparse, having few users who rate a few items and many users selecting very few items. The scantiness of users also requires that inferences from other user's profile be used in predicting a particular user's interest[2][8].
2. Scalability- Which is usually a big data problem that defines the recommender system's capability to produce suggestions in real time[4].
3. Cold start – a challenge resulting from efficiently predicting for a new user or item with little or no history.

Scalability is as a result of the challenges associated with big data, making it challenging to deploy many applications to provide regular updates in response to the changing dynamics of user preferences. The sparsity of the data is the true reflection of the effect of the choice of approach in

dealing with the huge gap between user rating and product and is a subject of discussion as the selected data representation may not, necessarily reflect the population at hand.

Now, a more dominant challenge, peculiar to the collaborative filtering approach, are the various scenarios of the cold-start problem, which includes, but not limited to:

1. Enhancing the experience of newly added users,
2. Making recommendation of new unpopular items or newly added products and
3. Making recommendations for users (both new and existing) with little or no personal information.

Another subtle challenge is the tradeoff between these three significant challenges (scalability, sparsity and cold start) as the accuracy of predictions and computational time are closely linked with leveraging these three challenges of collaborative filtering recommender systems. The cold-start problem impacts Key performance indicators as it may define a user's first impression on items and affect their conversion to clients[9].

The peculiar problems fundamental to this research include the computational time lapses in responding to user choices instantaneously, the slow time scale of updating the recommender system, the various phases of the cold start problem outlined in this section and efficiently managing the tradeoffs between scalability, sparsity and the cold start problem.

In response to these problems, the thesis proposes to add an adaptive approach to enhance the offline model of existing systems. Irrespective of the approach adopted, there has been increasing effort to utilize user reviews to enhance the description of items or users [2]. Reviews constitute a great wealth of information alongside the personal preference of users and items. Researches into reviews are focused on enriching and adequately describing items, users, or features that are also central to this research.

1.2 Research Objectives

The research described in this thesis seeks to achieve the following objectives:

1. Design, develop and deploy an adaptive multi-hybrid recommender system
2. Evaluate the computational efficiency in terms of speed and memory footprints
3. Evaluate accuracy metrics for the multi-hybrid recommender system
4. Implement multiple algorithms for making recommendations
5. Design and implement a movie web portal for testing and evaluation
6. Apply the AHCF to other datasets making it useful in other domains
7. Compare the AHCF implementation with other baseline recommendation algorithms.

1.3 Relevance of Research

In summary, this research makes methodological and managerial contributions. In terms of methodology, a novel hybrid collaborative model combining an offline phase and adaptive phase, content, and collaborative approaches and the use of the LDA model to address core issues affecting recommendation systems while enhancing personalized recommendation for users in the movie domain is developed. The model has improved performance on retraining to make recommendations subject to user preference changes.

Managerially, the model is useful for acquiring an understanding of consumer preferences on user-product recommendations such as targeting, personalization, and segmentation, which are major marketing activities. The research enhances business profits, considering the first impression of users, user retention, or promotion of unpopular or unknown items boosts business revenue[10].

Subtly, recommender systems reduce user/customer frustration as they aim to help them get exactly what they are hoping to acquire on using the various platforms. The engine developed is also made available for open-source use, thus, helping to meet the business needs of an organization.

The model presented represents an entirely different approach to making a recommendation that leverages ratings and textual information jointly.

1.4 Thesis Outline

The rest of the thesis continues as follows:

Chapter 2 consists of a literature review on recommender systems, collaborative filtering, topic models, feature extraction and dimensionality reduction. Chapter 3 discusses system requirements, analysis and specifications, design considerations, and the selection and development tools for the adaptive hybrid model, while Chapter 4 focuses on the development of the model and discussions on accuracy metrics. Results and findings are also interpreted. Chapter 5, provides summaries on the discussion and results and extensive data analysis is performed on computational time. Chapter 6 focuses on conclusion and recommendations.Contribution to knowledge, observations and recommendation are also highlighted in chapter 6.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

Machine learning, statistics, and marketing are pertinent to this thesis. The literature on the LDA topic model, natural language processing, recommendation systems, and pervasive work in collaborative filtering and the cold-start problem are included. These areas are concisely studied. The chapter is organized into 12 sections; Recommendation Systems, Collaborative Filtering, Cold start problem, Adaptive Recommender Systems, Natural Language Processing (NLP), LDA Topic model, Feature extraction, Similarity measures Performance Metrics, Existing Systems and Summary.

2.2 Recommendation System (RS)

Many studies implement the memory and model-based approaches for making a recommendation of items. These systems differ algorithmically from data search algorithms in their ability to produce results without an explicit request from the user[11]. Prevalent among these are the content-based, collaborative, and hybrid approaches. They constitute 35% of sales for Amazon,2/3 of Netflix movies viewed, and 38% of Google's News click-through[9].

The hybrid approach to recommender systems, as of 2016, suffers from minimal research efforts, although some studies indicate its high accuracy level compared to the other methods. Some ML techniques, like KMeans, despite its popularity, have not been researched enough, hence, present an opportunity for future work[1].

2.3 Collaborative Filtering

Collaborative filtering is geared towards making recommendations that are personalized based on the needs of the user[12]. Some side limitations of CF are the assumption that most popular

items add extra value to all user recommendations. L. Esperanza, [13] proposed the use of content-based approaches to address this assumption; nonetheless, the popular approach is useful in addressing the cold start problem.

In the real world, the velocity and volume of data arriving poses challenges such as real-time updating, unknown size, and concept shift. S. Chang et al. [14] proposed to use streaming recommender systems based on a recursive mean-field approximation to address these challenges.

2.4 Cold start problem

The cold start problem is defined in several ways and appears to be very broad, with some researches touching on Complete Cold-Start (CCS), no rating information available and Incomplete Cold Start(ICS); little rating information available[15]. The authors in [9] distinctively tackled the pure cold-start, which deals with completely new users navigating a system which is a narrowing of the CCS. In addressing the CCS and ICS problem, [15], proposed two models that utilize time-aware model and timeSVD++ with SDAE deep learning architecture for content feature extraction, the timeSVD++ predicts unknown ratings with considerations for temporal dynamics of user preferences and item features. The results of the prediction error, RMSE comparative to Netflix dataset, proved its superiority over baseline implementations. The time and item content information impacted positively on the implementation. However, the ICS item-based model performed poorly for items with few ratings (e.g., 3 ratings).

Silva 2019,[9] argues that only 4% of consumer visits to e-commerce platforms end up in users buying items due to privacy issues among others and users usually arrive on a platform via incognito searches, thus, aggravating the cold-start problem. In reference to [9], the author asserts that there exists no singular straightforward approach that readily addresses the demands of first-time users, hence the need for a non-personalized complementary mixture for RSS. They evaluate

four different non-personalized recommendations; thus, the Popularity (k most popular items), Best-Rated (k best-estimated items), Recent Items (k last items consumed) and Random Popularity (k random popular items). They test 3 hypotheses which invalidate the assumption made by e-commerce owners that items biased by popularity are potentially suitable for first-time users. P. Shinde [16], attributes the cold start problem with the inability to get feedback in feedback-based systems.

2.5 Adaptive Recommender Systems

Neighborhood-based models represent one of the popular CF approaches of which scalability is a significant challenge. This makes adapting to user preference dynamics complicated due to long intervals of training. To address this, a scalable and adaptive CF for item-item recommendation suited for personalization with the neighborhood model updated instantaneously to give user feedback in a stream is proposed. The algorithm has no offline phase[17]. The algorithm was compared with the Incremental stochastic gradient descent (ISGD) [18]. The streaming algorithm is seen to be scalable in that it is space-efficient and highly adaptive to changes.

Zadeh 2015, [44] introduced KMeans Cloning (KMC) as an adaptive clustering approach based on KMeans clustering comprising of two novel merging and splitting procedures.

Collaborative filtering recommendation systems that implement single algorithms have many drawbacks. K. Haruna, M. A. Ismail, D. Damiasih, J. Sutopo, and T. Herawan [20] assert that the user-based approach, for example, based on the behavior of other users; thus a new behavior does not lead to an update of items recommended. The item-based approach on the other hand, focuses on item metadata which presents many challenges, especially when there is a large number of items. These problems ushered in the mixed recommendation algorithms, thus, the hybrid collaborative filtering techniques. Y. Song, W. Ji, and S. Liu, [19] use user and item combination

as a hybrid approach. This makes recommendations by weighted summing of the recommended results of two algorithms. Others also merged the user-based CF with the content-based.

While a hybrid approach to collaborative filtering is a common practice among many recommender systems and following the success of the collaborative filtering techniques, their adaptivity hasn't gained much attention as these systems in themselves have a chunk of challenges as indicated in section 1.1 of this thesis[20]. This thesis considers the adaptive nature of the hybrid collaborative filtering technique as a key component of the research especially when it comes to leveraging it with the offline model.

2.6 Natural Language Processing (NLP)

In natural language processing, automatic representation of content concerning textual data in probabilistic topic models from different kinds of literature such as in [21], is introduced. As described in the introduction, this thesis relies on the LDA model to exemplify the peculiar demands of RSS. The model employs more affluent latent factors to accommodate multiple descriptions of varied movies by gleaning through available textual data to summarize the recommendation by topics as against traditional supervised topic models that are not adaptive for a personalized recommendation.

2.7 LDA Topic model

Adopting a mixed CF and content-based approach is problematic as it is dependent on meta-data such as title, description, year, etc., of the item. This challenge limits the chances of exploring new content. P. Chu and S. Lee, [22], proposed using word2vec to analyze the semantics of the words represented in a unique vector on user comments available online. They allude it is a true reflection of user opinions on purchased items. J. Wei, J. He, K. Chen, Y. Zhou, and Z. Tang, [15], asserts

that the topic model is particularly useful for dealing with implicit rating prediction but not effective for latent representations with sparse content information for item attributes.

Topic modeling in other business domains include how emerging markets are researched [23], the structure of approaches to organizational research methods [24], and in real business settings, such as, in determining patent topics [25]. The paper referenced in [26], used it in understanding online brand discussions. In finance, [27] grouped 219 articles on the application of business intelligence to the banking sector into topics, and 150 topics determined in US 10-K annual reports [28] [29].

LDA is an unsupervised statistical model based on probabilities that reveal words by using Bayes estimation. T. Rajasundari, S. Palaniappan, and P. Kumar[30] compared the Latent Dirichlet Allocation (LDA), Latent Semantic Analysis (LSA) of topic models with three different learning approaches (K-NN, Naïve Bayes and KMeans) using BBC news corpus. The recall and precision metric were employed in the analysis, for which it was found that the topic model outperformed these algorithms.

2.7.1 Caveats of Topic Modelling

G. Brookes and T. McEnery, [31], used the LDA topic model to group a volume of online comments from patients from the National Health Service more successfully compared to traditional approaches. The evaluations carried out were to check the feasibility of combining topic models with well-known methods in linguistics, such as discourse analysis for a more discerning explanation of textual data.

Some of the drawbacks outlined include naïve modelling of text. Thus, the order of words in text is immaterial, as long as they co-occur. This decouples words from their grammatical and syntactic context usage. The use of the stop-word list to remove closed classed words is problematic as there is the tendency to exclude relevant information. Stemming or lemmatization[32], thus, shrinking

of words morphologically to common base forms, though, it addresses the data sparsity problem, overlooks established conclusions in corpus linguistics; for instance, T. Mcenery, M. McGlashan, and R. Love[33] realized the words Muslim and Muslims constantly indexed different discourses about Muslims.

Replicability is another challenging area because researchers deduce topics from the word-lists manually by 'eye-balling' without resorting to any standards or frameworks, making the results of any research challenging to be verified and falsified or replicated. Also, [34] stated that in comparing models, evaluations are subjective and beyond the iterative approach, there is no easy alternative for acquiring a suitable number of topics.

2.8 Feature extraction

The usefulness of feature extraction extends many fields and areas, especially in machine learning. In the preprocessing of data, the selection, transformation, and extraction of features must be carefully designed as a primary concern. User-item rating data are usually noisy for real-world recommender system implementations due to the unfitting granularity of rating scales, subjectivity in ratings, and the difficulty associated with quantifying preferences, etc. [38]. Another challenge is that the representation of features in most kinds of literature focuses on the selection of weighting features. In [39], the authors proposed to combine two approaches based on statistics and the quality of features. D. Sisiaridis and O. Markowitch, [35] proposed to reduce the time needed for feature extraction by automating the feature extraction process for security analytics of heterogeneous data derived from different network sensors. The primary question is to optimally define a way to handle the complex input either by using the full number of dimensions or decomposition of the initial logs into distinct baseline structures. By this classification, it becomes easy to transform the data to apply machine learning techniques. This technique was contrasted

with the data frame approach employed in apache-spark, which proved to be simple and straight forward by just examining the schemas and retrieving only one of the records.

As a feature extraction technique, [7] used a matrix factorization model to make use of reviews to rank product features most useful for the preferences of users. The rating based recommendation is dependent on ranking and latent factors coupled with opinion mining of the quality of item features [47].

2.9 Similarity measures

In computing the user similarities, some of the popular metrics include the cosine and Jaccard measures. Reference [36] used a combination of fuzzy cosine and Jaccard similarities to enhance the total similarity between users/movies. This is to increase the reliability and robustness of the final similarity measure in addressing the cold-start problem in particular. An improved similarity measure method is proposed, which reduces the Mean Absolute Error (MAE) by focusing on commonly rated items and the rating difference of common item [37], using implicit and explicit data in establishing similarity amongst users or items. By explicit rating, the user provides ratings that can determine what a user likes and dislikes for recommendations. For the implicit data, a typical example of music recommendation is when the number of times a user played a particular song is used to represent user interest. For explicit ratings, it is self-evident what a user likes and dislikes but implicit data are usually noisy, hence the need to use some metric, such as, repetitive usage to establish preference and similarity relationships [38].

2.10 Performance Metrics

When it comes to metrics for performance evaluation and accuracy of predictions, the most commonly used are the Mean Square Error (MSE), Root Mean Square Error (RMSE), Mean Absolute Error (MAE), R-squared coefficient and many others. These represent accuracy metrics,

but there exist other equally useful metrics such as precision and recall that are dependent on the task at hand and more useful for classification techniques. It is argued that MAE is a robust evaluation model. It is the deviation of the predicted values from the actual values [39].

2.11 Existing Systems

Comparative studies on two of Africa's e-commerce giants(Jumia and Konga) indicate their dependence on the history of purchases made, browser history, user behavior on the platform in making personalized recommendations [40].

Netflix serves as a subscription model for personalizing recommendation on movies and shows a user may be interested in. Fundamentally, estimations of the chances of watching a particular title in the catalogue based on ratings and viewing history, similarity among users, title, genre, etc. information are taken into account. The time of the day, device type, and length of viewing are also employed in the recommendation process. On adding a new profile or registering as a new user, suggestions are made and as a user selects these optional suggestions, similar recommendations are made; otherwise, popular movies are recommended. For the titles in the rows on the home page, ranking is created within and between rows to enhance personalized recommendation[6][41].

Amazon Machine Learning, Strand, Peeruis, Azure ML and IBM Watson are some proprietory recommender models that are employed in the industry, while, The Universal Recommender, Raccoon Recommendation Engine, HapiGER and easyRec are open source models for the recommendation. However, Surprise benchmark algorithms are employed in this research.

2.12 Summary

In this dissertation, a unique contribution was made by realizing a novel hybrid collaborative filtering recommender engine, highly adaptive to changes in user preferences almost

instantaneously. It addresses scalability, sparsity, and extensively tackles the cold-start problem. Other side issues tackled include the overspecialization and over-fitting issues associated with recommender systems. Also, a topic model (LDA) with user-generated product tags and textual description is implemented. It is evident that a one size fits all is impossible when it comes to predicting user preferences or to correctly decide what to present and recommend to a user and thus the need for a combined multi-hybrid system that can meet the varied and unique demands of consumers.

CHAPTER 3

METHODOLOGY

3.1 Introduction

This chapter illustrates the methods used in data collection and preprocessing, feature extraction, dimensionality reduction techniques, and clustering techniques. System requirements and analysis and design implementations are discussed. The functional and non-functional requirements of the AHCF are also highlighted. The hybrid of the offline and adaptive models, machine learning techniques and the hybrid of the user and item-based collaborative filtering techniques are emphasized in this chapter.

3.2 System Design and Process

A vast majority of researches in recommender systems present models that perform their operations offline. Offline model design and implementation mean, results are fixed and can only be updated by retraining the entire recommender model. This may lead to overspecialization especially in situations where a user has exhausted the recommendations. The offline models in most cases are impractical with few real-world implementations. These systems primarily forfeit the major concept of recommender systems to make recommendations to a user in real-time or within short time intervals. In relation to that, this thesis leverages a recommender model that capitalizes on real-time results for a user over short time intervals. While we cannot do away entirely with the offline model as it contains extremely useful information, it is worthy to note that user preferences online are influenced by many intrinsic and extrinsic factors that the system in the strict offline mode cannot account for. There is the need for a system as in Figure 0.1 below that account for such dynamics.

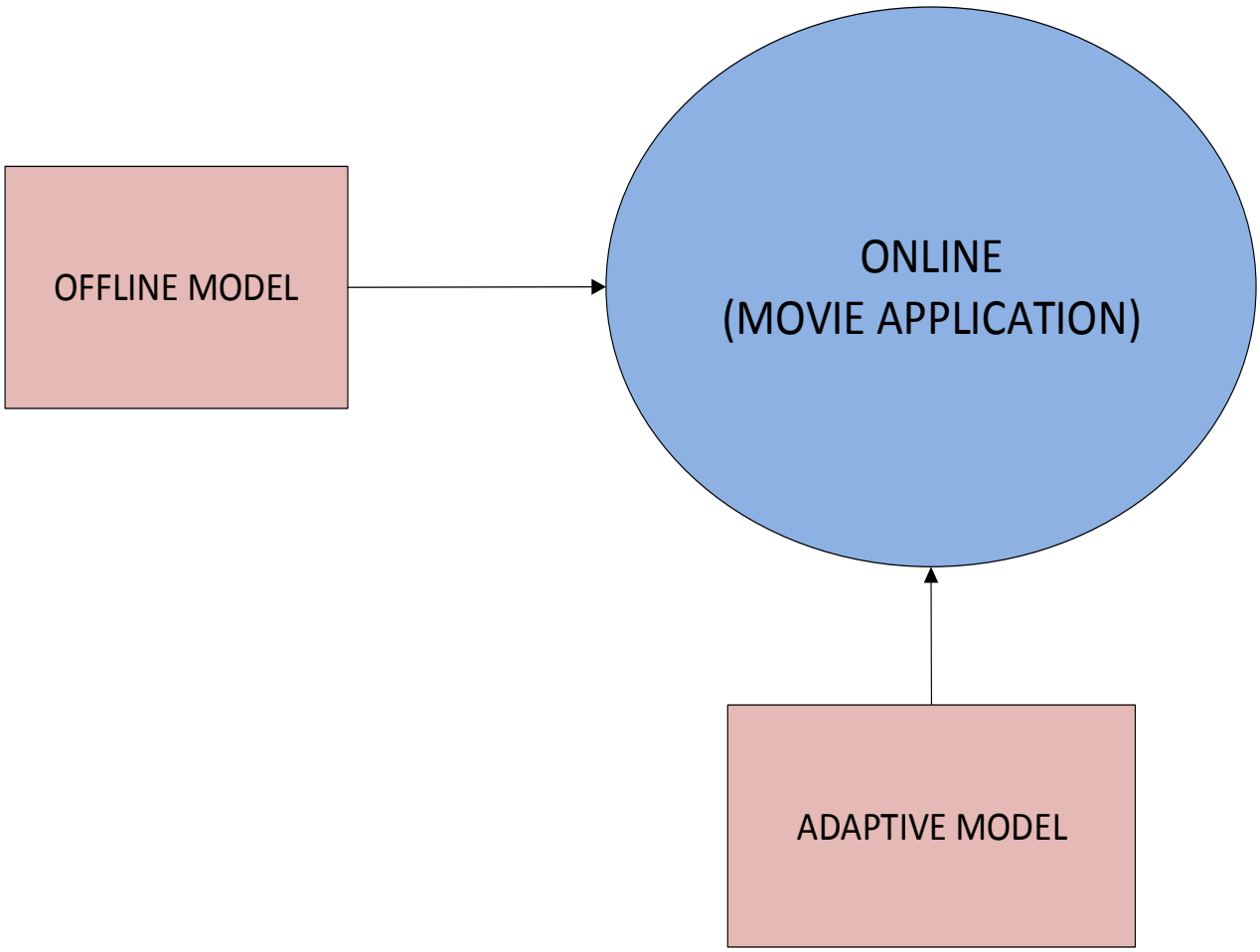


Figure 0.1 Basic model

On the other hand, having a complete real-time model is almost impossible since recommendations require many actions, and it is impossible to predict all likely scenarios correctly. The system design proposed in Figure 0.2 represents a hybrid model that takes advantage of the offline and adaptive model for real-world implementation in the movie domain. The movie data is passed through the ALS algorithm described in Figure 0.3 for cleaning and dimensionality reduction before clustering in Figure 0.5. LDA cluster (Figure 0.4) plays a vital role in mining the text data as user preferences can also be captured in text, such as reviews. The KMeans streaming (Figure

0.6) is a variant of the KMeans with an update rule for clustering singular input in batches to mimic real-world scenarios.

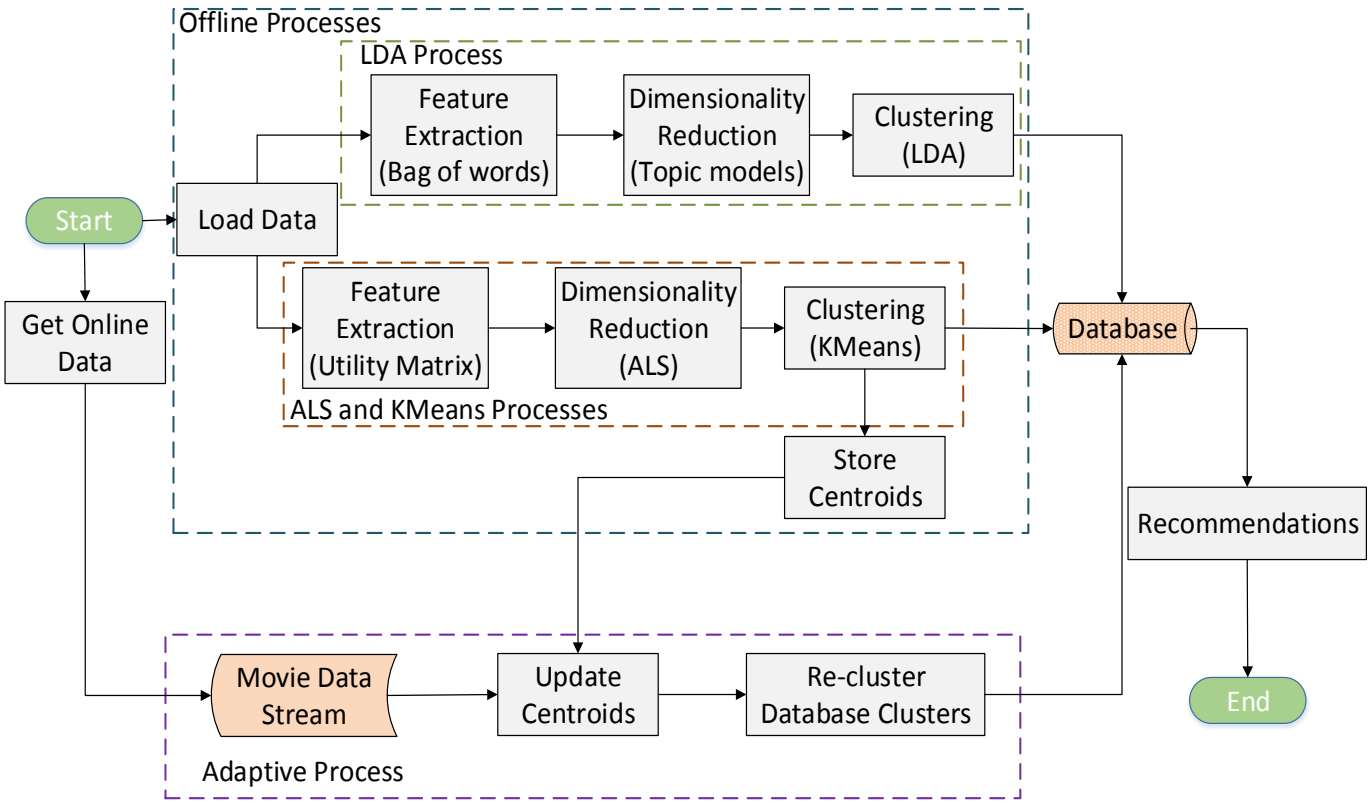


Figure 0.2 Hybrid model system design

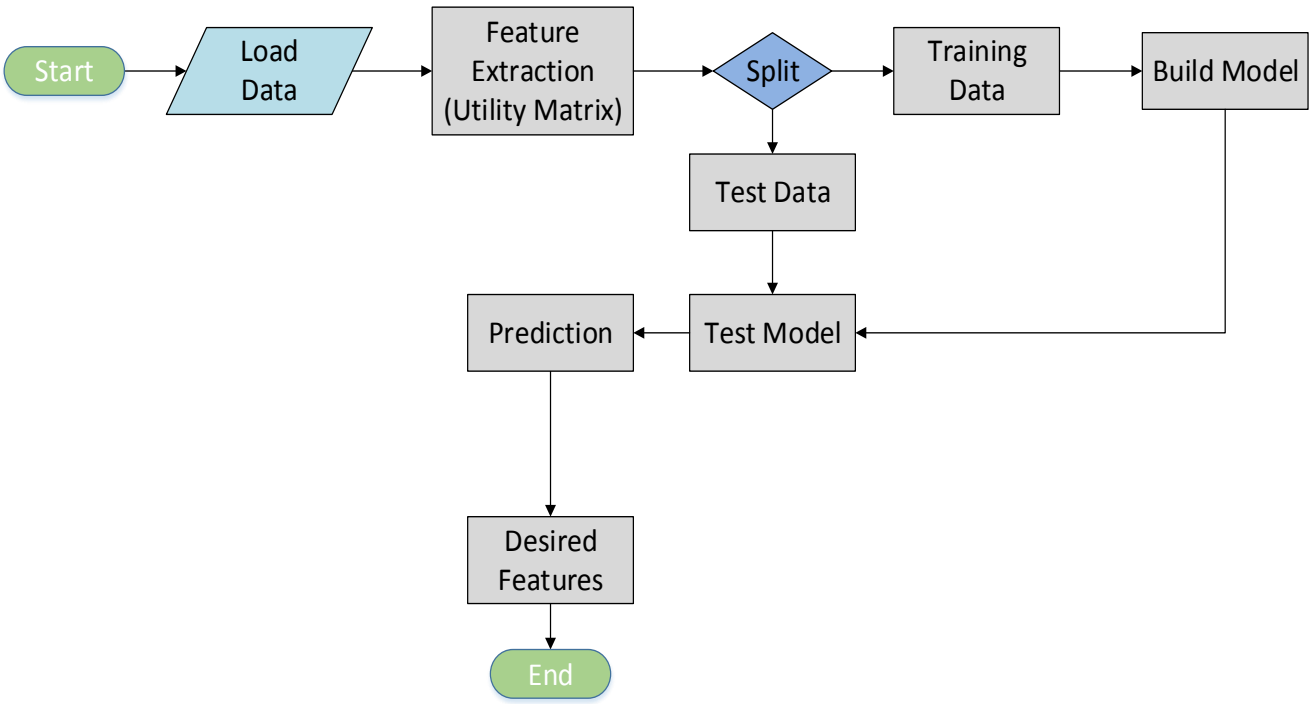


Figure 0.3 ALS Model

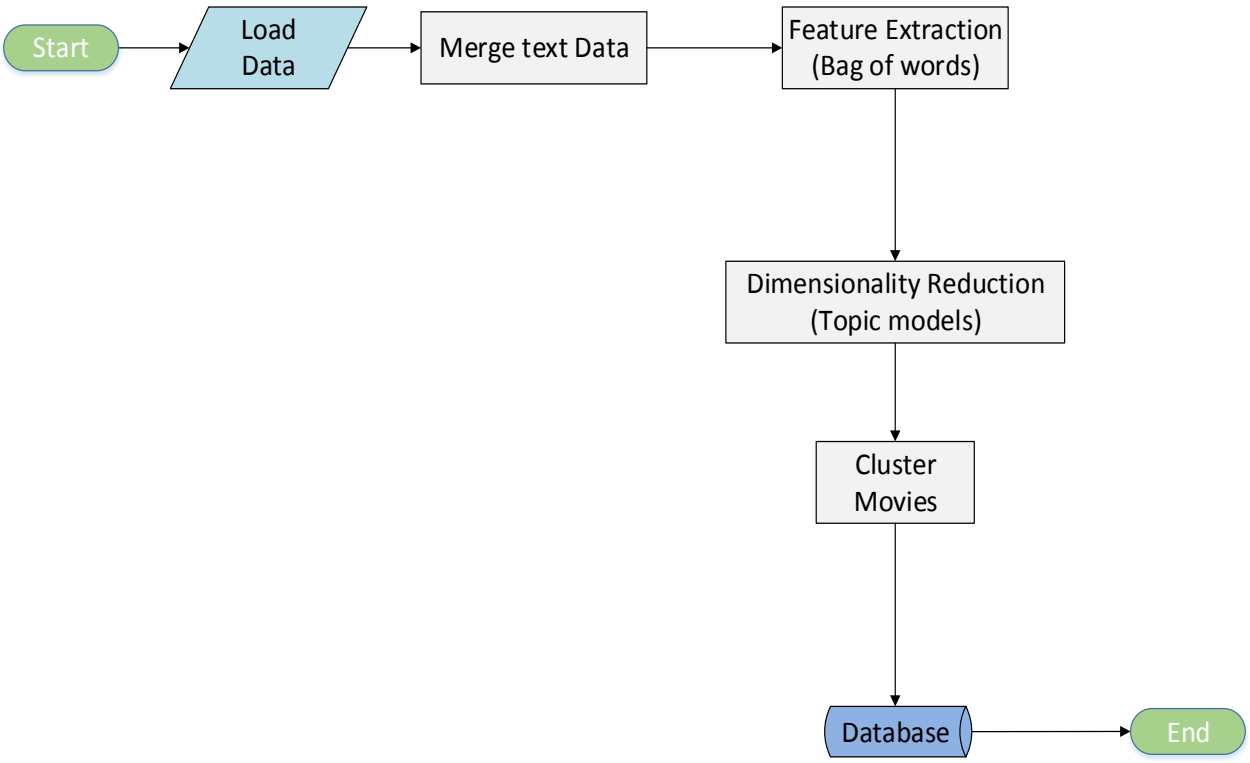


Figure 0.4 LDA Algorithm implementation

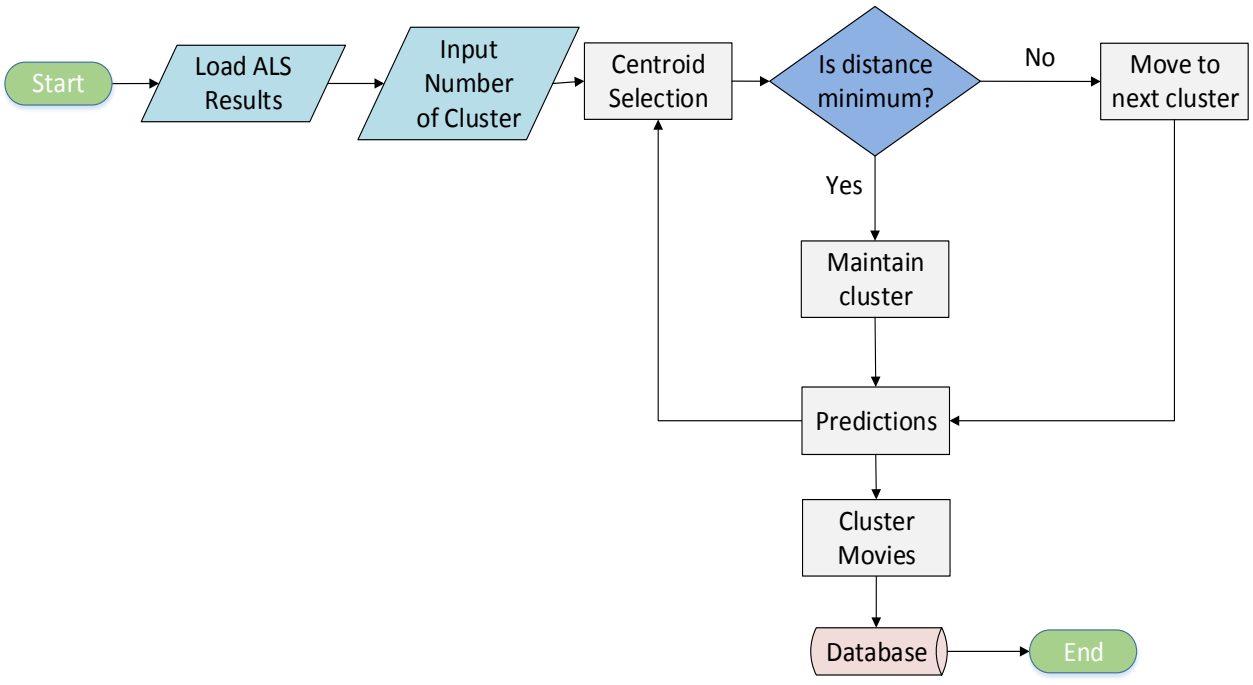


Figure 0.5 KMeans Algorithm implementation

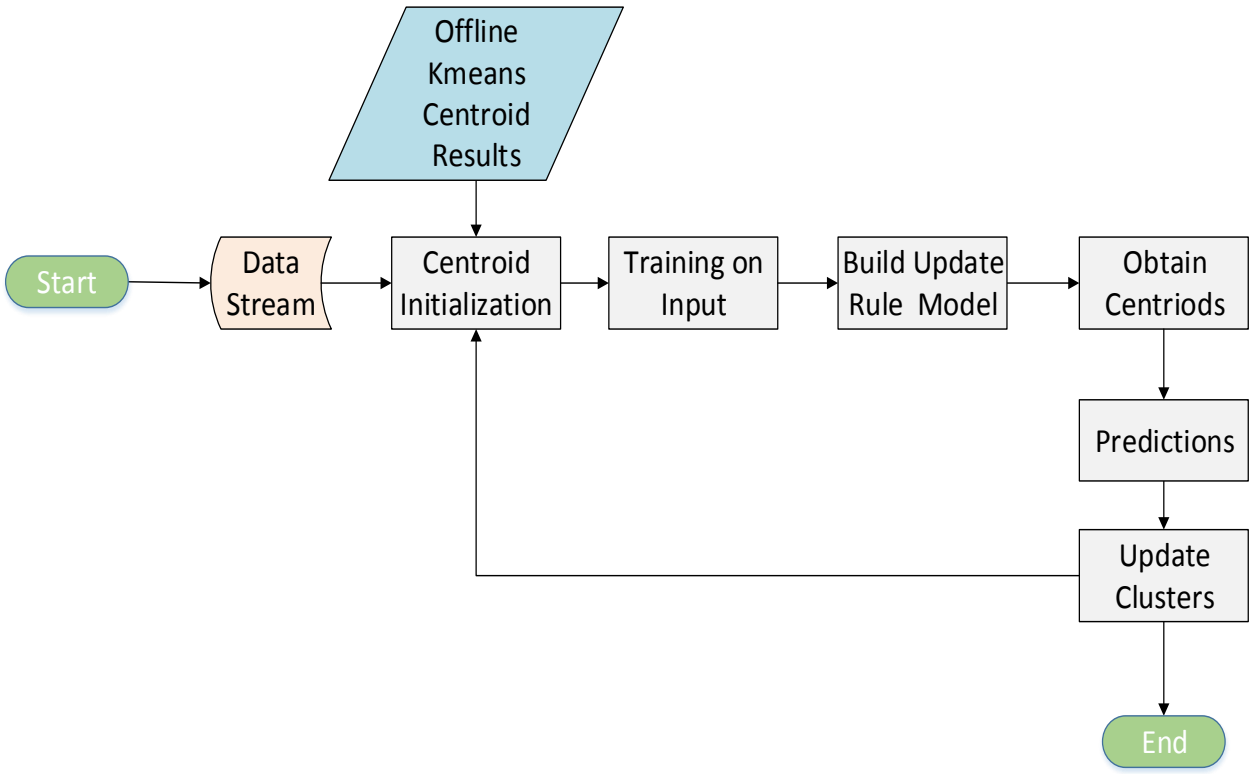


Figure 0.6 Streaming KMeans Implementation

3.3 Data Description

The 100K and 1M benchmark datasets from MovieLens will be adopted in this research. MovieLens is a movie recommendation service. To verify the validity of the model implemented, scalability test is performed on the datasets [42]. A minimum of 20 rated movies is captured; the dataset consists of genome-scores, genome tags, links, movies, ratings, and tags[43][44]. The MovieLens dataset is a popular dataset used in many types of research, such as in [37 - 43]. This may be attributed primarily to the up to date nature of the dataset. The dataset has 19 genres, with movies belonging to more than one genre.

3.4 System Requirements, Analysis, and Specifications

The purpose of the design process is to produce an efficient recommendation system. To achieve the design process, the following requirements must be satisfied:

- Offline model
- Adaptive model
- Movie recommender system

Combining these models in a movie recommendation system aims at:

- Recommending movies to avoid overspecialization after the user has exhausted proposed recommendations.
- Providing room for alternative recommendations that increases the chances of meeting user preference.
- Updating the recommender model as user interest change or user actions are difficult to account for per the pre-trained model by using an adaptive model.

While the offline model establishes the relevance of previously existing data in pre-trained models, the adaptive model accounts for current data. There is the need for not only theoretical proof but also a real-world implementation as theoretical models are not a guarantee that business demands and requirements will be met, hence the development of a movie recommender portal to test the engine.

3.4.1 Functional Requirements

The system to be developed must be able to:

1. Make recommendation on the first visit through (Popularity, best-rated, recent items and random popularity) for all users

2. Allow a new user to rate movies but not get personalized recommendation until they register
3. Allow a user to get mixed recommendation consisting of items from different categories or based on some computed matrix for selection
4. Allow a user to view movies without registering to be a user
5. Allow a user to get personalized recommendation after successful registration and login
6. Allow an existing and new user to rate a movie and watch trailers
7. Provide personalized recommendations based on user activities and rating information.
8. Must have items organized under 4 categories (Popularity, best-rated, recent items and random popularity) for all users
9. Must allow the user to search for movies based on textual descriptions or categories
10. Make recommendations based on user selection of movies (not necessarily by rating but by textual information)

3.4.2 Non- Functional Requirements

The following non-functional requirements should be satisfied;

1. The AHCF model should be able to efficiently and rapidly make a recommendation of items to users
2. It must be able to make a personalized recommendation within very short time intervals
3. Must be generic to serve as a plug into other recommendation models
4. Must have items well organized to attract and make user navigation simple
5. Must be engaging to enhance user retention by enhancing user rating experience.
6. It must be highly scalable and reliable

The system developed satisfies the following implementation requirements:

Table 0.1 System Specifications

OS	Windows 10
OS type	64bits
RAM	8GB
Hard Drive	1TB
Processor	AMD A10-8700P Radeon R6, 10 Compute Cores 4C+6G 1.80GHz

3.5 System Design

An adaptive system takes into account unexpected actions that prior to the system deployment were unknown. One principal aim of machine learning is to learn and adapt as human beings do. However, implementing an adaptive system only invariable assumes a lack of value for already existing data that is available. There exist some patterns in user actions that may be repetitive or common between members of a particular group, and other common factors that will influence the choice of a product may also exist. For instance, it is prevalent for students in a Computer Engineering department to purchase computer-related items, and such recommendations are usually straightforward. However, user interests can change for a particular product based on other users' opinions; that is why user opinion represented not only in ratings is particularly useful in making recommendations. The figures below represent the design for the implementation of the two phases of the model. Figure 0.7 depicts the general overview of the entire recommender model, which is subdivided into the offline model (Figure 0.8) and the adaptive model (Figure 0.9). The offline phase consists of three main algorithms the ALS for dimensionality reduction, KMeans for categorizing the movie results and LDA for text dimensionality reduction and categorization. The

adaptive phase consists of a simple streaming KMeans specialized with update rule to process data arriving in a stream.

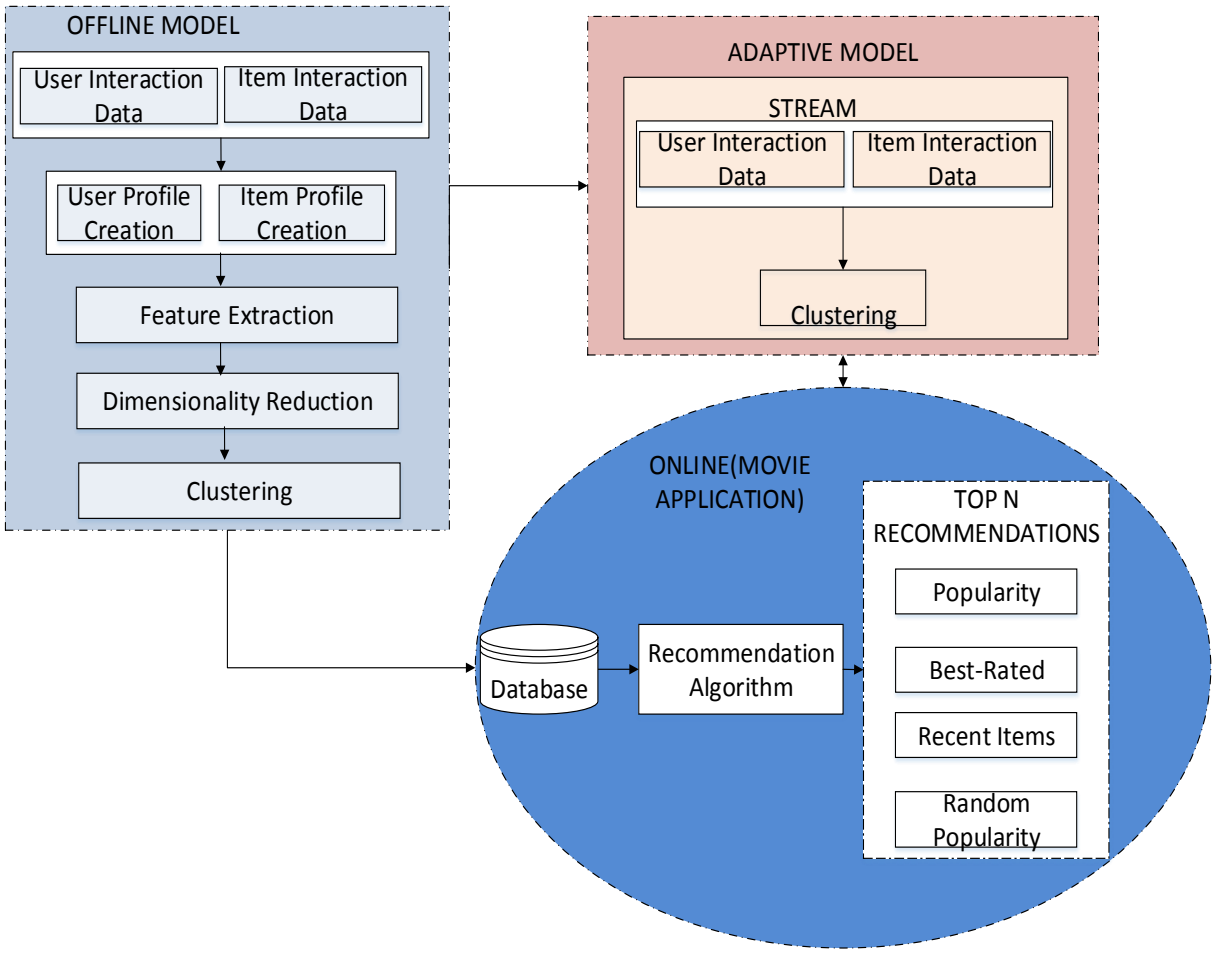


Figure 0.7 General recommender model architecture

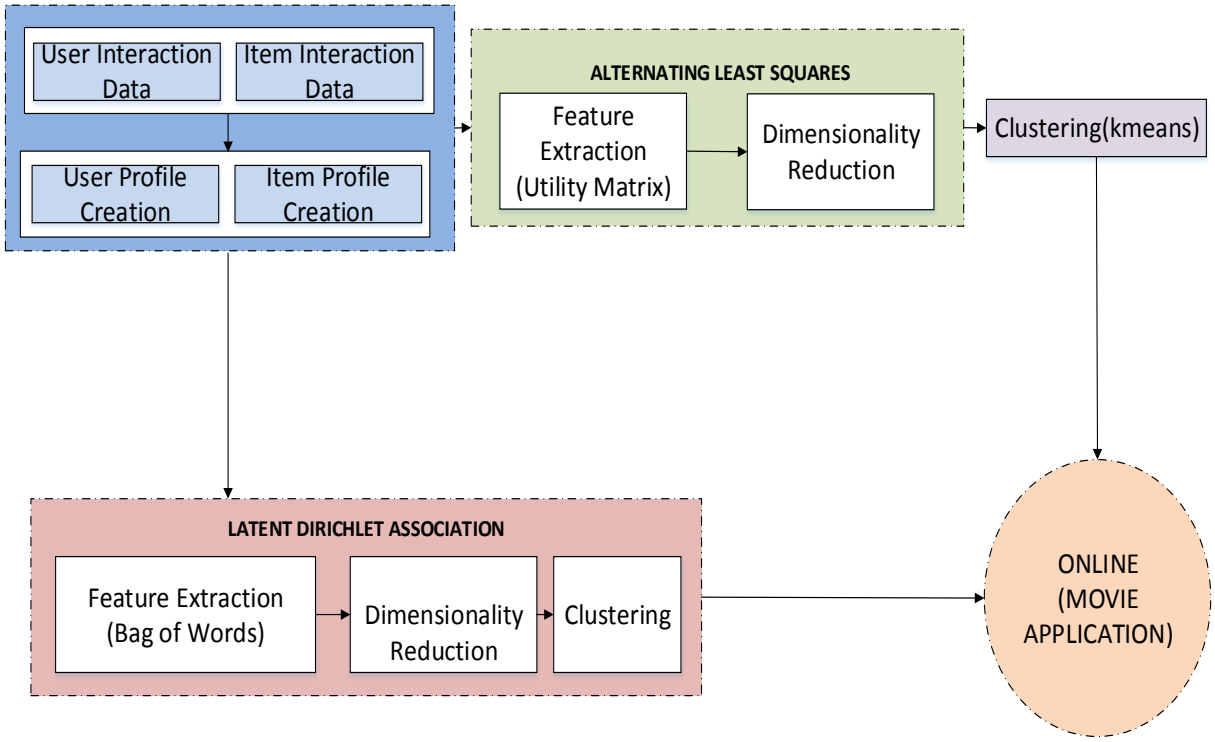


Figure 0.8 Offline model

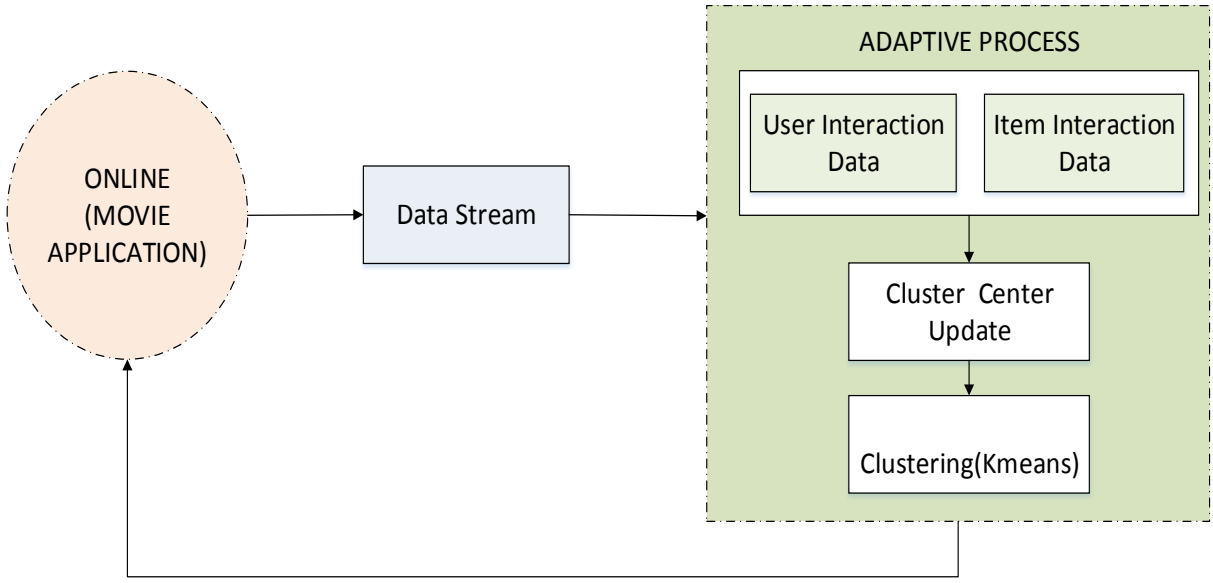


Figure 0.9 Adaptive model

In the online phase of the system, the user (new or existing) is allowed to carry out actions indicated in Fig 3.10 and the associated use cases for users and system interaction are

represented in Fig 3.11 and Fig 3.12 respectively.

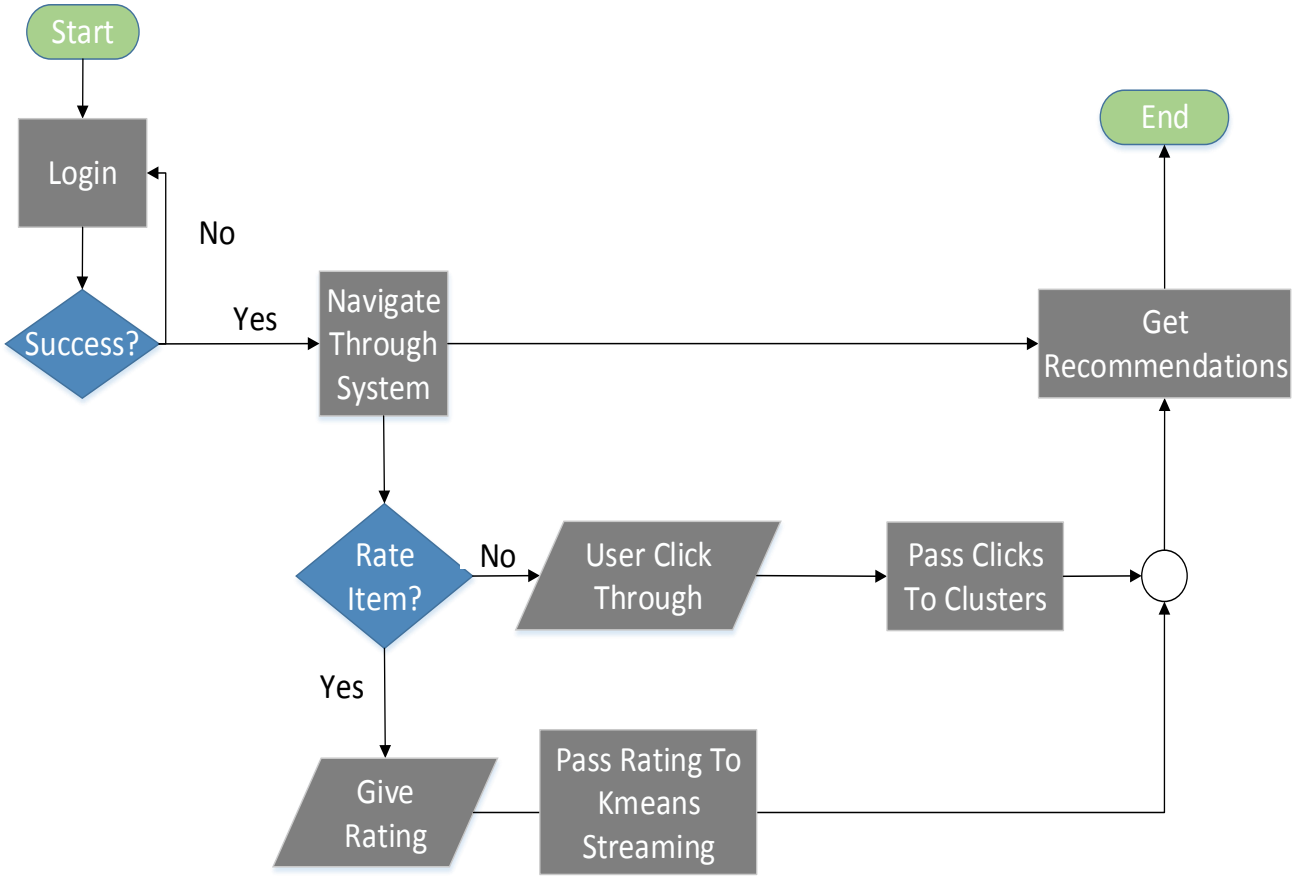


Figure 0.10 Online movie recommender (process flow)

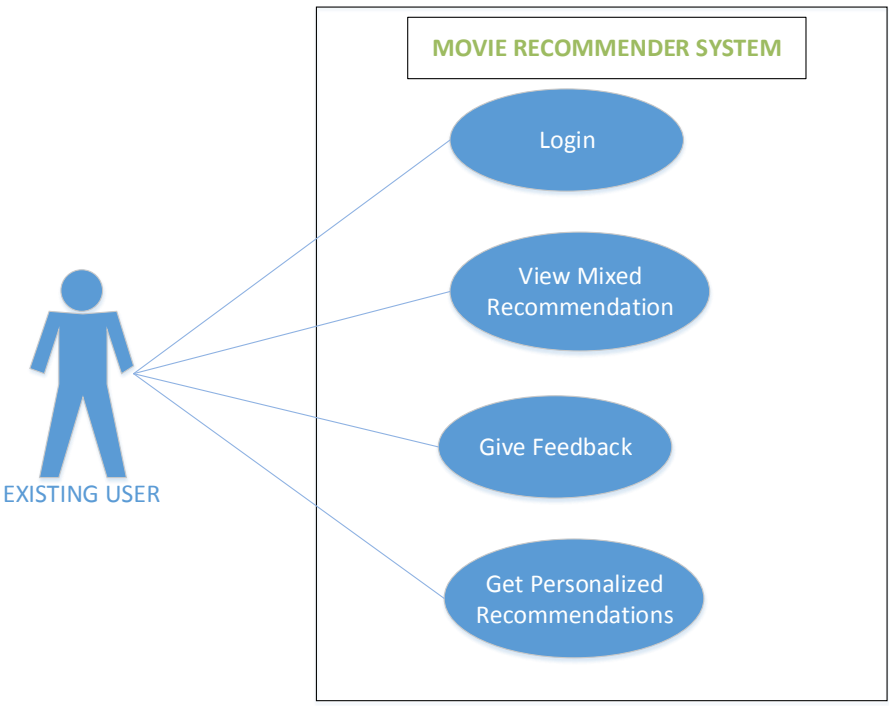


Figure 0.11 Use case (existing user)

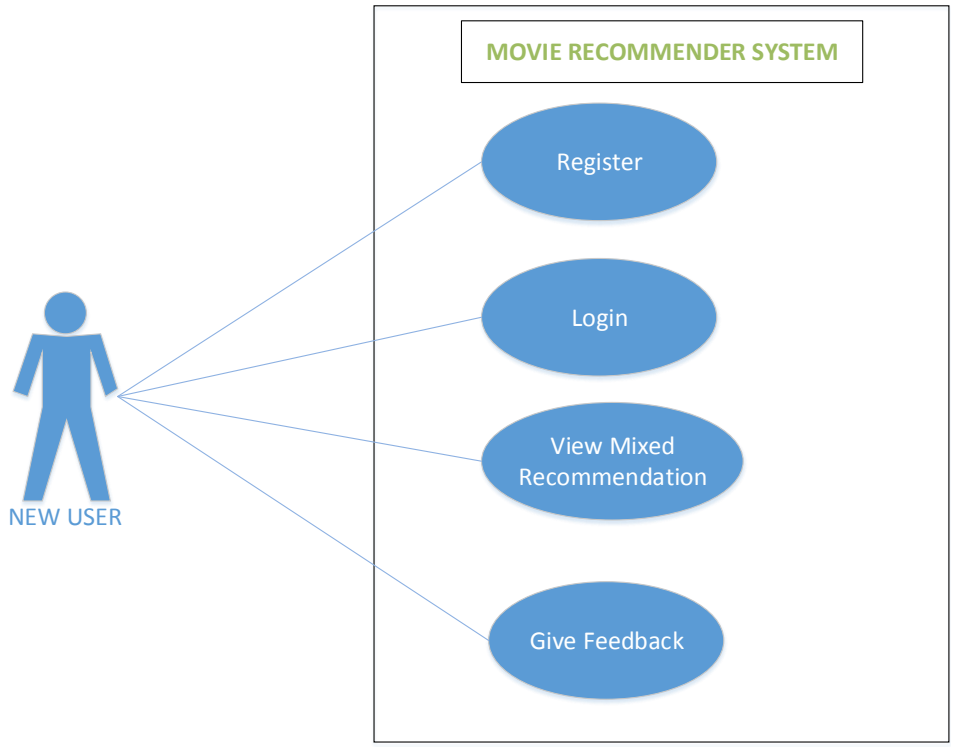


Figure 0.12 Use case (new user)

3.6 Design Considerations and Selection

The system design approach in this thesis is focused on enhancing overall recommendation taking into consideration every available information in movie recommendation. Despite the time lag in hybrid algorithms and training offline, the model represents an ideal approach for addressing the dynamic nature of user taste and preferences in movie recommendation. These concepts and design approaches, in future work, can be extended to accommodate other recommendation domains such as YouTube among others.

As much as it is herculean to predict precisely user tastes and preferences, the model proposed takes into consideration the time needed to make accurate predictions within a very short time interval. The quality of recommendation is based on people's opinion and the product information for which a user does not need to be overwhelmed or waste much time in finding what suits them best by the provision of an organized search.

3.7 Development Tools

Table 0.3 and Table 0.4 shows a list of tools employed in achieving the objectives of the thesis.

Table 0.2 Tools for Recommender Algorithms and web portal

Tools For Recommender Algorithms:	Tools For Web Portal:
Ubuntu 18.04 and Windows 10 OS	MySQL Database
IntelliJ Idea	Angular JS
Scala 2.11.12	TMdb Database API
Sbt 2.0	HTML
Apache Spark 2.4.0	CSS

Oracle JDK 8.0, JDK 11	Bootstrap
Akka Server	PHP 7.0

Table 0.3 Graphical tools

MATLAB 2018b
Microsoft Visio 2013

CHAPTER 4

SYSTEM IMPLEMENTATION AND TESTING

4.1 Introduction

The focus of this chapter revolves around the offline and adaptive model and a real-world implementation in the movie domain as well as tests carried out with other datasets. Results and performance evaluations are also discussed. The chapter is organized into 2 sections; System Implementation with its subsection, testing, and results with its subsection. The system implementation discusses the machine learning techniques employed while interpretations are drawn from their usage in the testing and results section.

4.2 System Implementation

The movie application domain will be used in this research due to the ease of data acquisition. MovieLens dataset by the University of Minnesota is widely used and easy to access. The Movie Database (TMDb) represents another primary source of movie ratings and titles, useful for building online tests and queries [1]. The dataset is loaded into Spark dataframe as presented below:

Table 0.1 MovieLens Datasets

Dataset	Ratings	User ID	Movie ID
1M	1000209	6040	3952
100K	100054	71567	65133

4.2.1 Dimensionality Reduction (Alternate Least Squares-ALS)

From the above representation of the data in (Table 0.1 MovieLens Datasets), the movie to user rating is sparse and considering the ratio of movies to users, a user-movie rating matrix (U) is created with M users and N movies of size |M| x |N| to address the sparsity problem. The Alternate Least Square (ALS) algorithm is employed to reduce the sparsity and dimensions of

the dataset as well as extract the needed features or latent factors from the dataset by summarizing each user and item on k-dimensional vectors x_u *and* y_i , respectively and predict user-item rating by $r \approx x^T y$. The primary concern is to estimate the complete rating matrix $R \approx X^T Y$ represented in a matrix form $x_1, ..., x_n \in \Re^k$ for user factors and $y_1, ..., y_n \in \Re^k$ for movie factors. By formulating it as an optimization problem and minimizing the objective function below, the sparsity problem is being addressed,

$$\min_{X,Y} = \sum_{r_{ui}\ \text{observation}} \left(r_{ui} - x_u^T y_i\right)^2 + \lambda\left(\| x_u \|^2 + \sum_i \| y_i \|^2\right) \tag{1.1}$$

$y_i =$ Item feature vector,

$x_u =$ User feature vector,

$\lambda =$ Regularization parameter,

$r_{ui} =$ Rating given by the user,

$u =$ User,

$i =$ item and the dot product $x_u^T y_i$ represent the interaction between the user, u, and the item i which is non-convex and NP-hard to optimize.

However, by holding the X set of variables constant, then the objective is a convex function of Y and vice versa. In practice, we hold X and optimize Y and then hold Y and optimize, X, as presented below:

Initialize X (user matrix), Y (movie matrix)

Given, $u = 1...n$ do

$$x_u = \left(\sum y_i y_i^T + \lambda I_k \right)^{-1} \sum_{r_{ui} \in r_{u*}} r_{ui} y_i$$

end

Given, i=1...m do

$$y_i = \left(\sum x_u x_u^T + \lambda I_k \right)^{-1} \sum_{r_{ui} \in r_{i*}} r_{ui} x_u \qquad (1.2)$$

end

till convergence

Updating x_u is $O(n_u k^2 + k^3)$, n_u represent movies rated by user u , and

y_i is $O(n_i k^2 + k^3)$, n_i represent users who have rated movies [46].

4.2.2 Clustering (KMeans)

Data clustering is known to be NP-hard for finding groups in heterogeneous data [19]. KMeans is notable for its efficiency and usage with large datasets, although its operations require fore-knowledge of the cluster [8]. It is the process of grouping similar data items together based on some measure of similarity (or dissimilarity) between items. In this case, K values are randomly selected between 5 and 100. There are various variants of the KMeans clustering algorithm. However, Lloyd's algorithm approach is employed to cluster similar users based on the features obtained. The focus will be to reduce the squared error function (Within Cluster Sum of Squared Error-WCSS), given by:

$$\sum_{j=1}^{k=1} \sum_{i=1}^{n} \| x_i^{(j)} - c_j \|^2 \qquad (1.3)$$

Where $|| x_i - c_j ||^2$ is the Euclidean distance between x_i and c_j . x_i , equals the number of data points in the i^{th} cluster, c shows the number of cluster centers. Users closest to their cluster centers are the one's representative of that cluster (most relevant users).

4.2.3 Latent Dirichlet Allocation (LDA)

To sum up the LDA technique of clustering text into relevant headings or topics. The algorithm assumes there exists a fixed hidden number of topics in a corpus when a word co-occurs. It uses it to determine what the topics are and determines the presence of the corpus probabilistically. In carrying out the topic generation:

1. Tokenizing is done to each document to form a collection of their words in no particular order (called “bag of words”).
2. Stemming is done to decrease the number of unique words and estimate unique term usage accurately. E.g.‘ please’ and ‘pleases’ may be reduced to the common ‘pleas’ stem
3. TF-DF experiment is done on the relative importance of the remaining words in the corpus.
    a. First, the percentage of instances of a term in a document compared to all terms in that document is calculated.
    b. It is then multiplied by the log of all documents in the corpus divided by the number of documents in a corpus that contain the term.

Very low and high TD-IDF terms are discarded because of their very rare usage in the corpus to describe a topic and for being overly used to describe a particular topic, respectively.

The number of topics is extended contrary to most literature in which the choice of 20 topics is a common practice because of the large, varied sizes of the datasets, purposely to explore the possibilities of getting fine topics by eyeballing process [47]. Also, for short messages, some literature, using around 20 topics were effective. The log-likelihood and log-perplexity estimation

are used aside from the eyeballing technique to determine the appropriate number of topics in this research. The count vectorizer approach is used to convert text to vectors which is much easier for the LDA technique employed to work with.

4.2.4 KMeans Streaming

The intuition behind the streaming KMeans clustering is the introduction of parameters to control the decay (or "forgetfulness") of the estimates using a generalization of the mini-batch KMeans update rule. For each batch of data, all points are assigned to their nearest cluster, and new cluster centers are computed to update each cluster using:

$$update\ cluster,\ c_{t+1} = \frac{c_t n_t \alpha + x_t m_t}{n_t \alpha + m_t} \tag{1.4}$$

$n_{t+1} = n_t + m_t$

c_t represent previous cluster center

n_t represent the number of points assigned to the cluster

α represent the decay factor

m_t represent the number of points added to the cluster center from the current batch

The decay factor α can be used to ignore the past: with α=1, all data will be used from the beginning; with α=0, only the most recent data will be used. This is analogous to an exponentially weighted moving average.

4.3 Testing and Results

To make the experimental results comparable and reproducible, the RMSE (Root Mean Squared Error) is adopted in the ALS to measure the accuracy of the predictions. Small RMSE values are preferred. The Within Cluster Sum of Squares (WCSS) is used in the KMeans as presented in section 4.2.2, for which minimum values are desirable. Log-likelihood and log-perplexity estimations are used to estimate the correctness of topic distribution in the LDA. Lower values indicate good predictions as well as the quality of the LDA clustering. We apply cross-validation

on the datasets to select the optimal value of k and rank for the KMeans and ALS algorithms respectively. This is useful in selecting the optimal dataset and the percentage split on training and test data that will make our web portal implementation efficient. The computational time complexities are also evaluated. These evaluation metrics are carried out for all the 3 datasets represented below.

4.3.1 ALS Computational Time and RMSE results

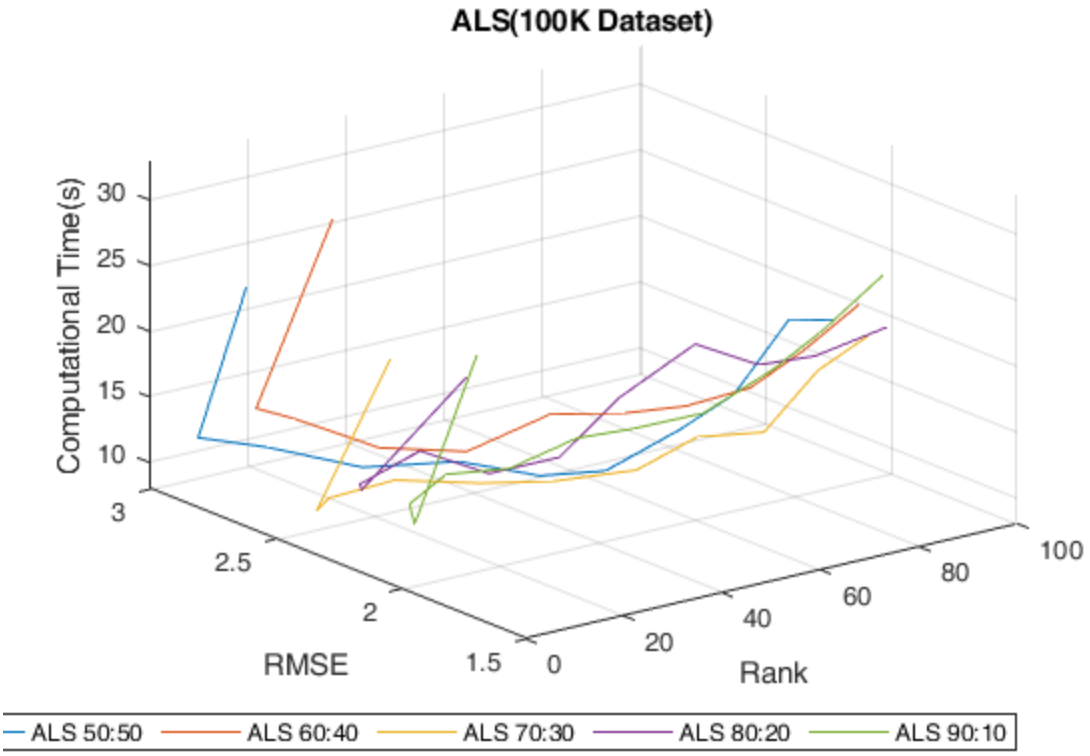


Figure 0.1 3D ALS result plot(100K Dataset)

The ALS result for the 100K dataset in Figure 0.1 has the 80:20 and 90:10 with cross-validation results having consistently lower values for the RMSE compared to the others and low computational time. Computational times are high with very low-rank values, such as when the rank is 5 for all cross-validations. For the 1M dataset represented as Figure 0.2 considering the

size of the dataset, the computational cost increases significantly between 10 and 60. The RMSE results also decrease significantly as compared to the 100K. 80:20 and 90:10 cross-validation results have lower points when the rank is less than 20. The computational times, however, increase with increasing rank value.

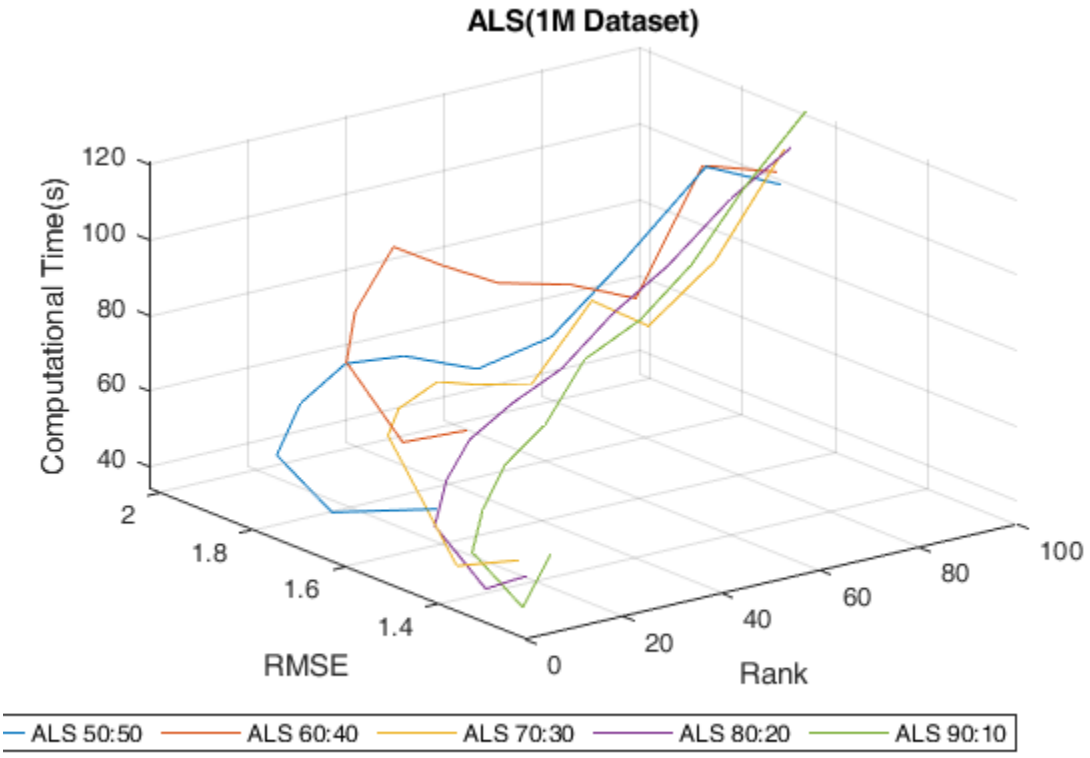


Figure 0.2 3D ALS result plot (1M Dataset)

4.3.2 KMeans computational time and WCSS results

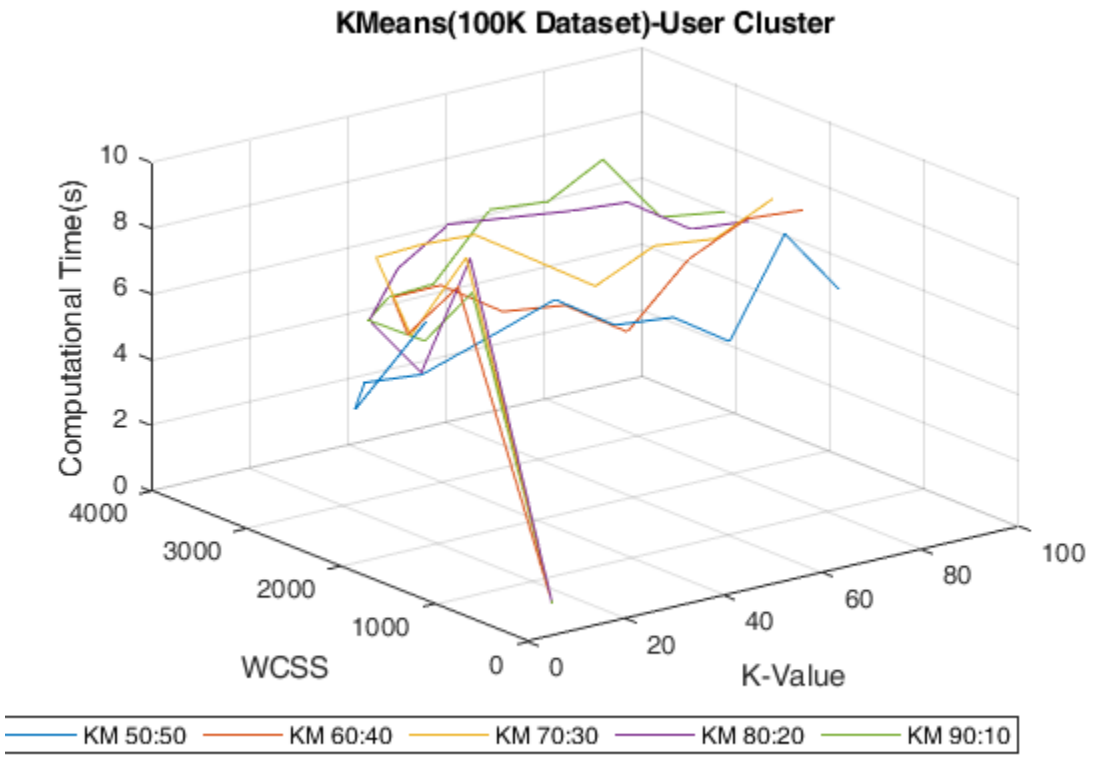


Figure 0.3 3D KMeans-User cluster result plot (100K Dataset)

The KMeans result for clustering movies based on similar users for the 100K dataset in Figure 0.3 has lower WCSS when K value is less than 40 but evens off after 3000 WCSS. It also has the 50:50 with the least computational time. The result has very close computational time differences in all results.

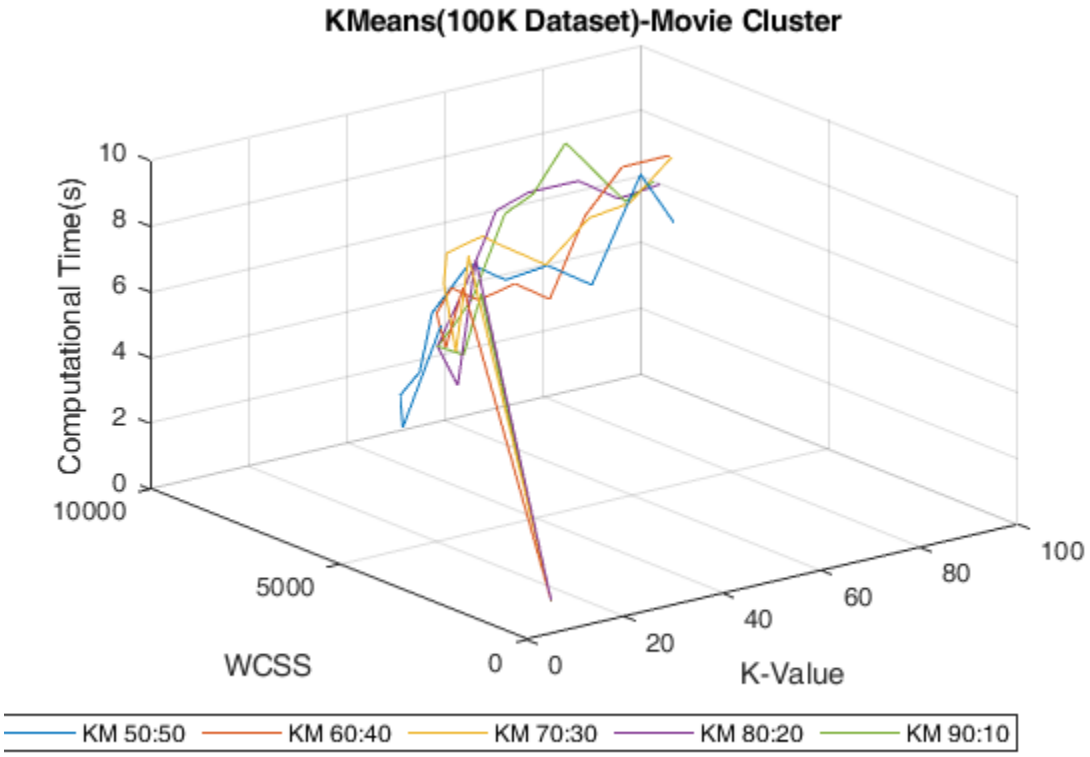


Figure 0.4 3D KMeans movie cluster result plot (100K Dataset)

The KMeans result for clustering movies based on similar movies indicates that as K- Value increases, the WCSS values increase and are accompanied by increasing computational time in Figure 0.4. A close range of computational value of 4s is seen for some K values

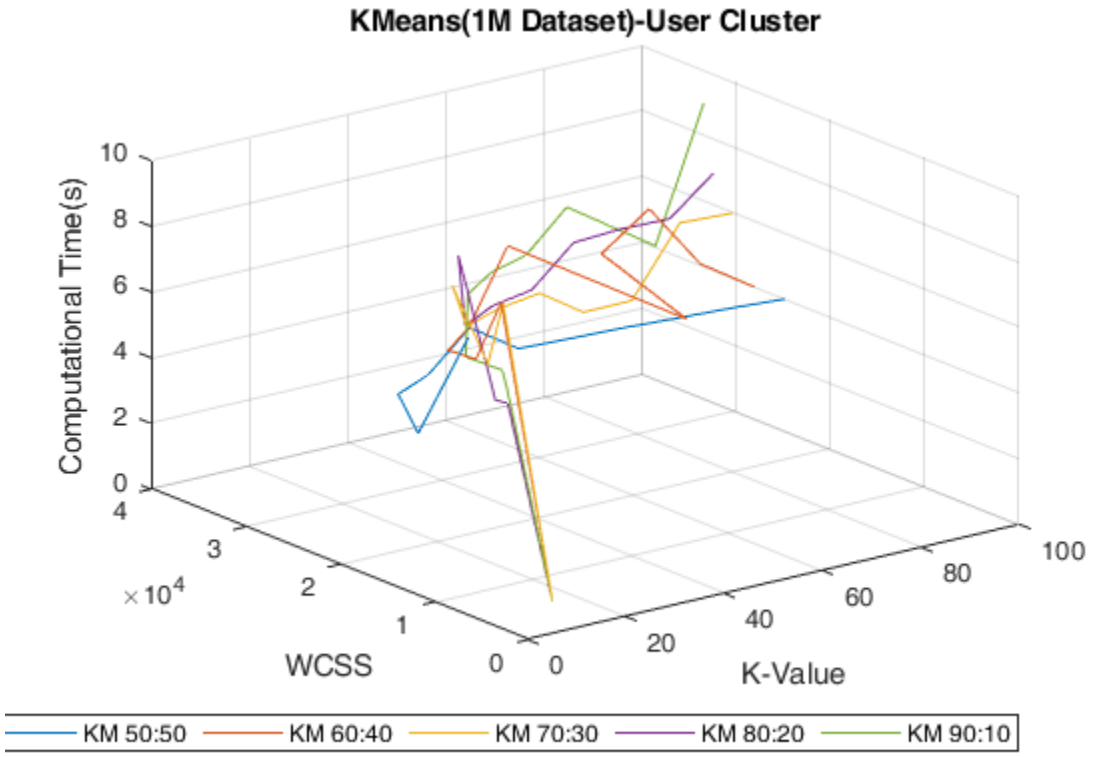


Figure 0.5 3D KMeans user cluster result plot (1M Dataset)

The KMeans result for clustering movies based on similar users for the 1M dataset in Figure 0.5 has decreasing WCSS effect for increasing K value and increasing computational time. The 50:50 cross-validation result has the least computational time, less than 4s throughout.

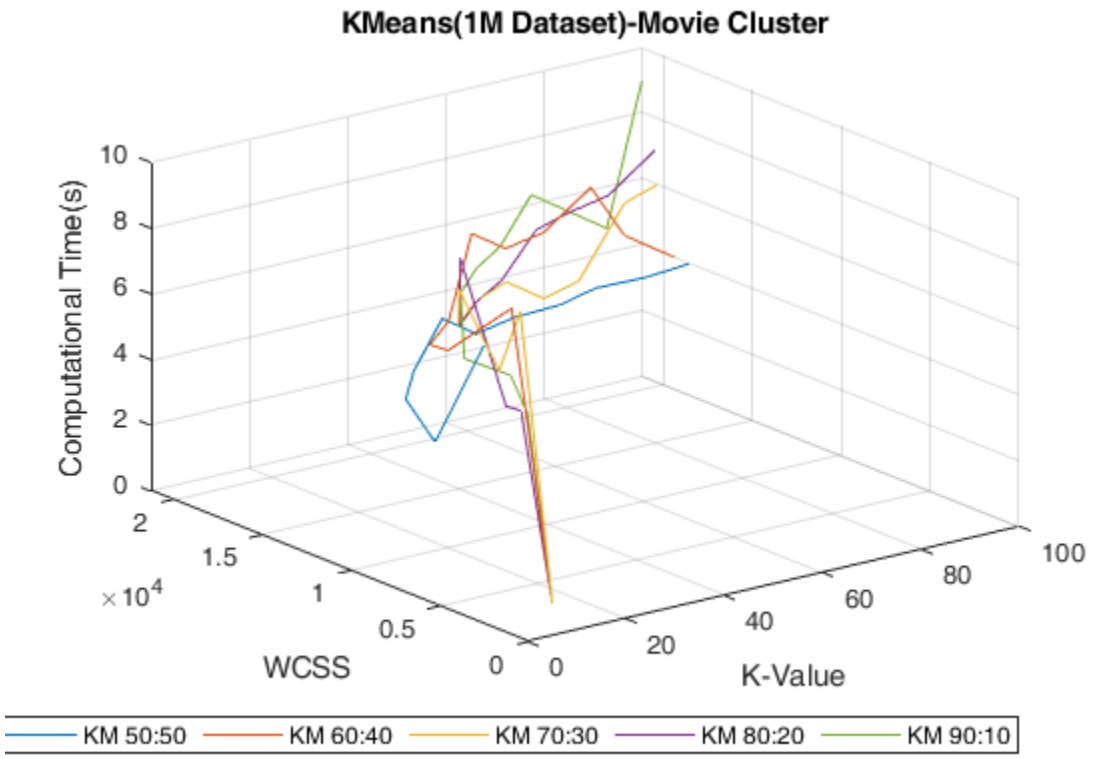


Figure 0.6 3D KMeans movie cluster result plot (1M Dataset)

The KMeans result for clustering movies based on similar movies for the 1M dataset in Figure 0.6 has very low computational results for K- Values less than 40 and high WCSS values as rank increases with the 50:50 cross-validation results recording the lowest results on WCSS and computational time.

4.3.3 LDA

Textual analysis is performed with the LDA topic model on a dataset that has high relevant textual information, in this case, the movieLens 20M dataset which comes along with tags and reviews. The effectiveness of the model is as presented in Figure 0.7 and Figure 0.8 below.

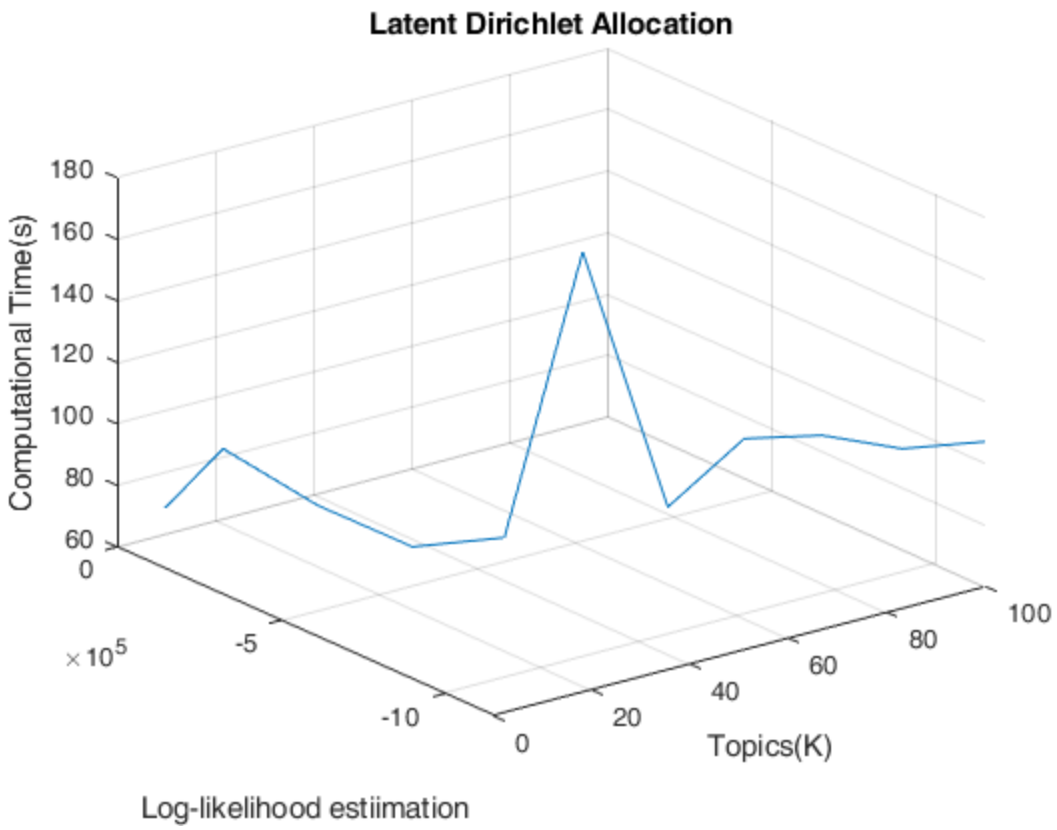


Figure 0.7 LDA log-likelihood plot and computational time

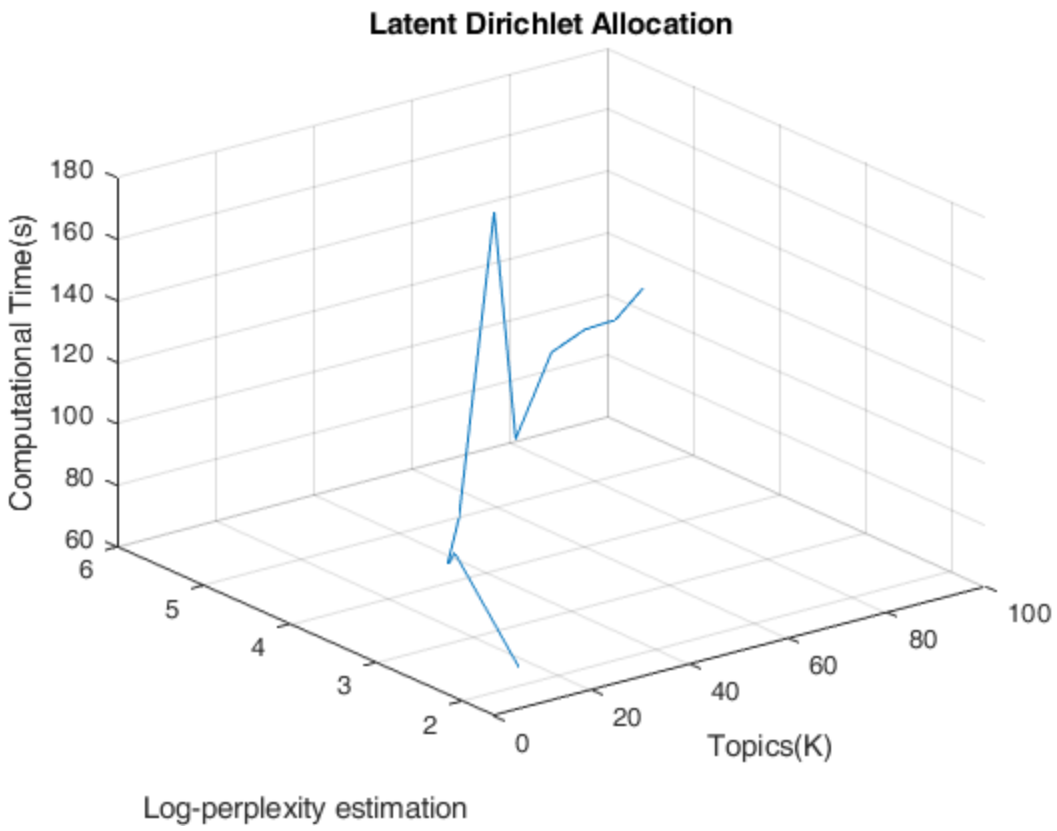


Figure 0.8 LDA log perplexity plot and computational time

From the above log estimates, increasing the number of topics significantly increases the Log perplexity (Figure 0.8) and decreases the Log-likelihood estimates in the dataset (Figure 0.7). Between 20 and 80 k topics generated, the results have high computational times and drops as k reaches 100. The larger the number of the topics, the better the computational times, log-likelihood and log-perplexity results but from the eye-balling technique, it is evident that words that are not representative or useful end up becoming relevant. Hence the recommendation in most literature to use 20 proves helpful although it comes with a computational cost.

4.3.4. Computational Time Comparison (ALS, KMEANS and Offline Model of AHCF)

The offline model of AHCF represents a combination of the ALS and KMeans algorithm. A computational time comparison is made of the three implementations. The 90:10 and 80:20

cross-validation results are used, together with the 1M and 100K datasets. The tradeoff here is that, even though the 50:50 cross-validation results for the KMeans are the best, the ALS result efficiency is preferred above the KMeans efficiency in this research because the ALS results are used for making recommendation alongside the AHCF model and the KMeans is an enhancement to the ALS, therefore the need to rely more on the ALS as a base algorithm and its best results. When the model is successfully trained, it produces the outputs presented in Figure 0.9 and Figure 0.10 below.

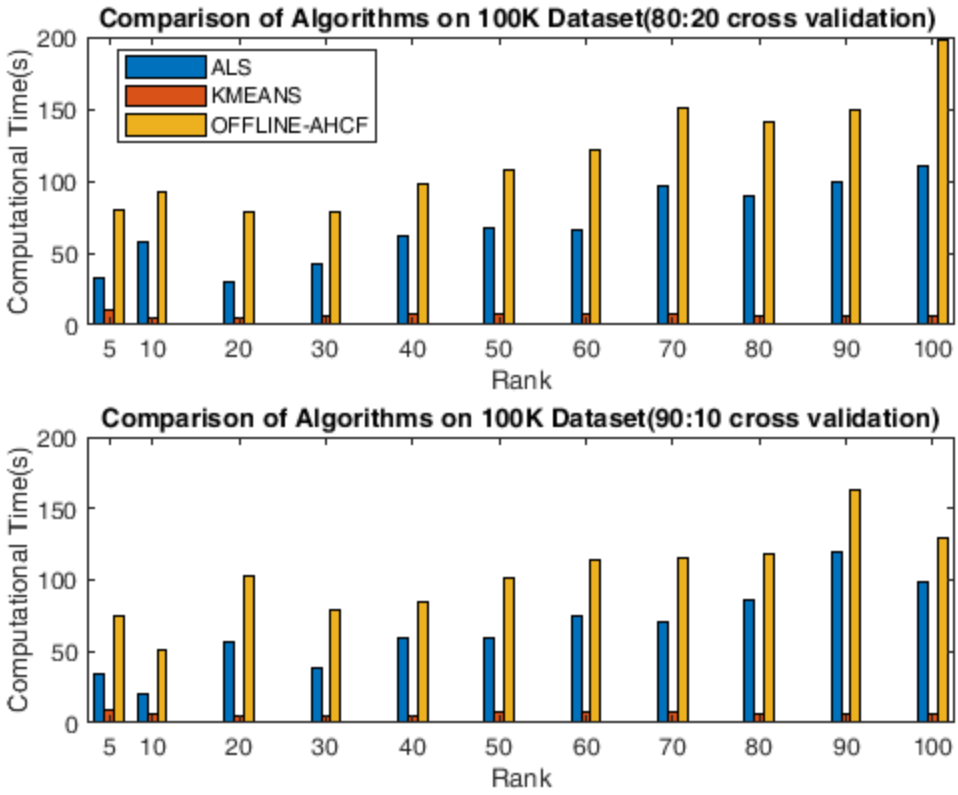


Figure 0.9 Comparison (ALS, KMEANS and Offline Model of AHCF)

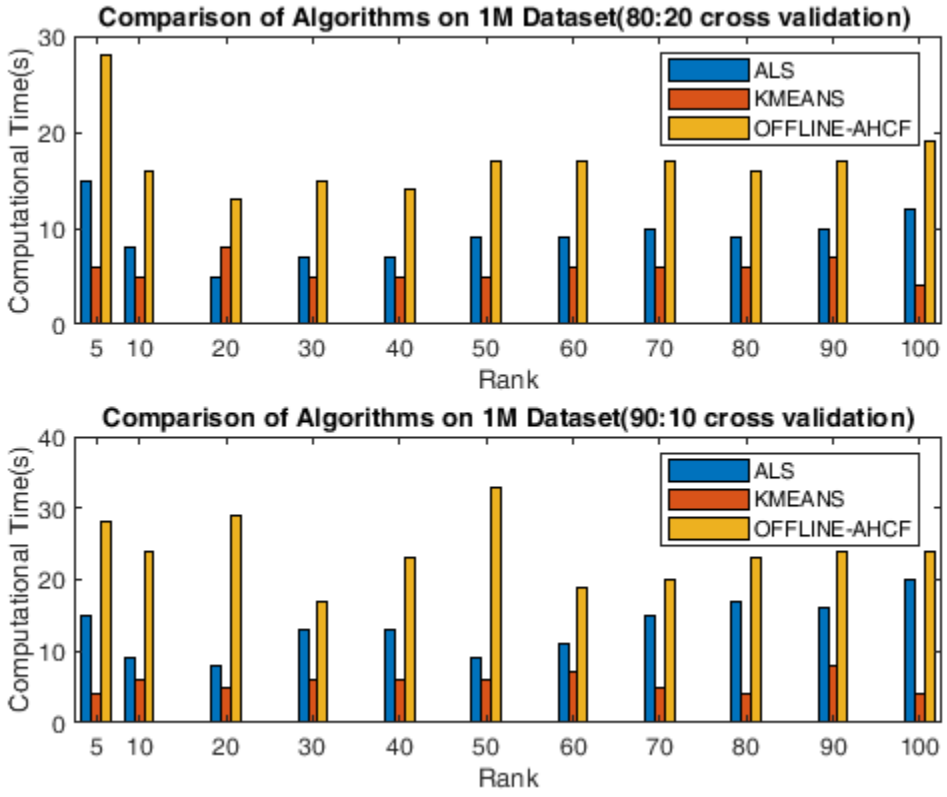


Figure 0.10 Comparison (ALS, KMEANS and Offline Model of AHCF)

Both results are closely matched with the 80:20 results-producing slightly higher discrepancies at rank 5 and 100. However, the 80:20 result is used in the implementation phase of the movie web portal. Choosing a rank value between 10 and 60 is most useful as the others further reduce the computational time. The offline model, however, performs some transformations, such as normalization in its computation aside from the ALS and KMeans algorithms.

Aside from the notion of combining multiple models in this research, a significant catch is the ability of the AHCF to adapt to user preference changes. The implementation adopted reduces the computational time significantly as compared to the individual algorithms and existing offline model implementation. (Figure 0.11below).

4.3.5 Comparison (Offline Model and Adaptive model of AHCF)

The existing offline models can be employed for adapting to new user preferences but may be computationally expensive. Below(Figure 4.11), the computational time required in making recommendations for a new user in real-time is evaluated, thus the time needed to retrain the model to make a recommendation is studied based on user ratings.

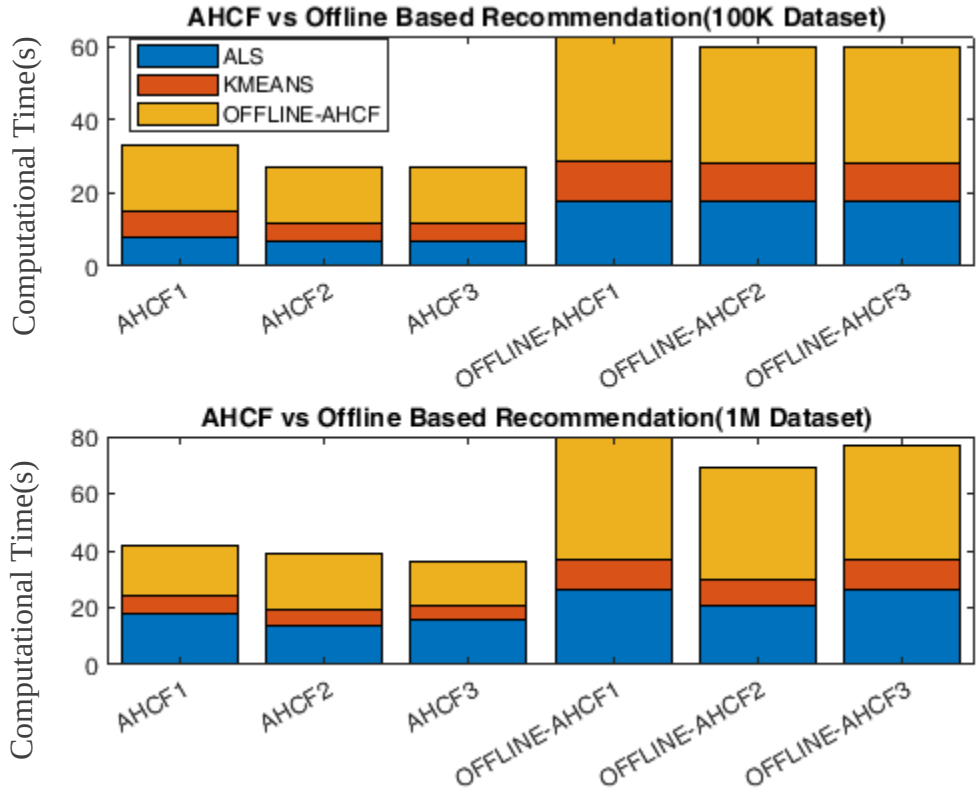


Figure 0.11 Comparison (Offline Model and Adaptive model of AHCF)

The use of AHCF proves advantageous over the base-line offline model of literature in [48] when the model is retrained to make a recommendation for a new user.

4.3.6 KMeans Streaming

The memory footprints, the number of tasks on spark Jobs on the successful execution of the KMeans streaming, is recorded at http://laptop-hcv0fien:4040/jobs/. This information changes

from time to time as computations are being carried out and once the recommender engine successfully performs computations. The memory footprints highlighted in Appendix C are taken instead of taking the individual memory footprints of the implemented algorithms because it is at this stage; all the algorithms become fully functional. From the results, it can be seen that the ALS uses more memory for its computation as against the other algorithms, for the reasons stated in Section 4.2.1 (particularly, its iterative nature).

4.3.7 Making Recommendations (The Adaptive System)

The methodology emphasizes six requirements for the implementation, which are:

1. Making recommendations for a first time user or users who arrive on the platform by incognito (thus addressing the cold start challenges)
2. Making more personalized recommendation for users with few or no ratings
3. Updating user preference or recommendations list as user preferences change over time via user ratings
4. Recommend similar movies in the neighborhood of movies selected by the user
5. Recommending movies based on randomly selected multiple or single genres or titles.
6. Ensuring that the hybridization is efficient and working well in sync

These requirements are addressed as presented below:

The system developed has 3 main components, a recommendation server (Recommender engine) written in the Scala programming language, a Websocket Server and an Apache Server. The Recommendation server is responsible for computing and re-computing the user's movie preference group based on some form of similarity. The Websocket Server establishes connection between the Recommender Server and the Apache Server to enable the recommender engine accept and respond to streaming requests from the user. The Apache Server runs a PHP

backend that provide basic services to the web portal. Finally the User Interface (UI) of the web portal is powered by Angular framwework running on a NodeJS Server. The recommendation algorithms consist of the offline algorithms (ALS, KMeans, and LDA) and the adaptive algorithm (KMeans streaming); however, both the online and offline algorithms are computed once the server start. The offline model is computed once after which the ALS results are used in some form of recommendation, the KMeans streaming is recomputed from time to time to accommodate the user rating changes via the web portal for updating the recommendations upon arrival of the batch. When the engine is run successfully, it establishes a connection with the Apache server (serving the PHP backend which interfaces between the recommender system and the user actions) through the Websocket Server as displayed in the blue square marker in (Figure 0.12).

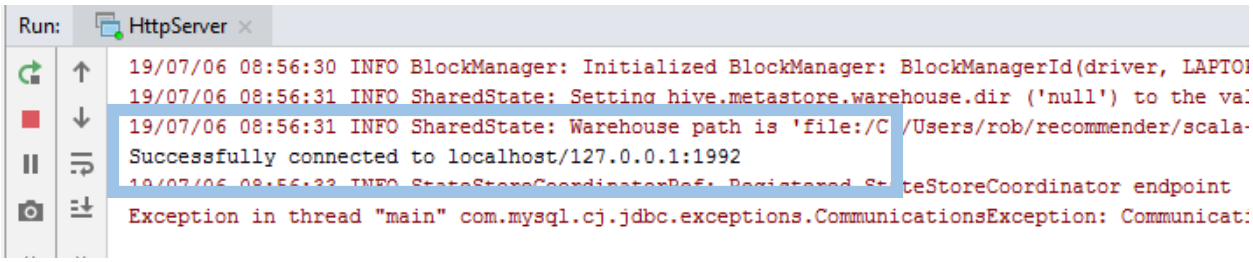


Figure 0.12 Recommender Engine establishing a connection

The KMeans streaming batch is set to 5 (for the sake of brevity) as shown in Appendix B, which means that upon a successful rating of 5 movies, the adaptive model is recomputed. As the user rates a movie, the user, movie Id and the rating value is passed to the engine as seen in Figure 0.13 below.

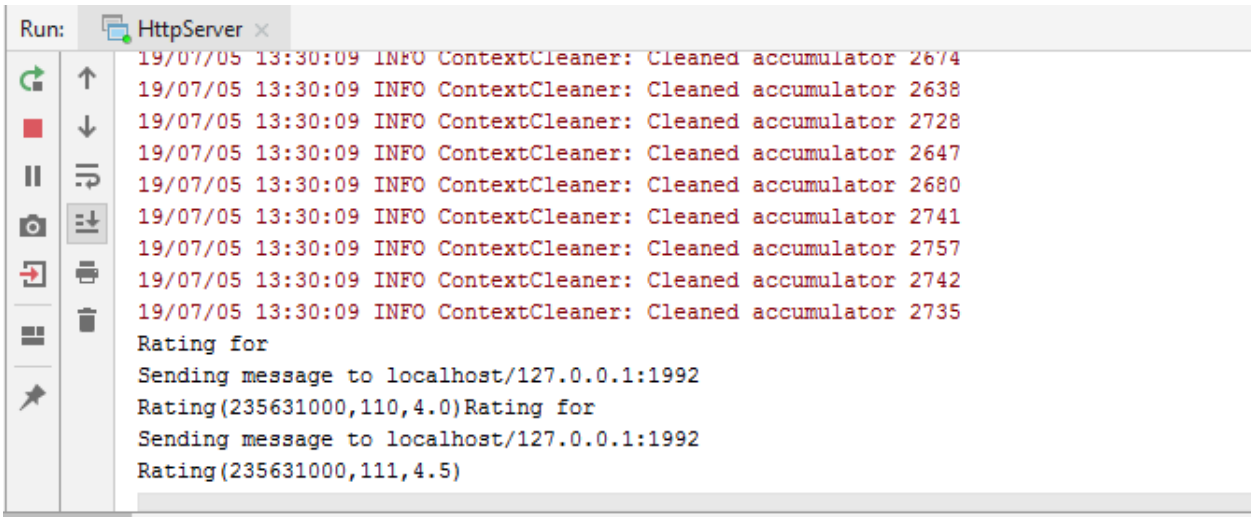


Figure 0.13 Recommender Engine receiving user input

The WebSocket Server (Figure 0.14), NodeJS Server serving the Angular frontend web application (Figure 0.15) and the Apache Server serving the PHP backend work together to manage the web portal (frontend), specifically displaying movie trailers, rating options and querying the database server as well as sending streaming requests to the recommender engine and querying for movie information from the TMDb database which is cached into the MySQL database to enhance query operations .

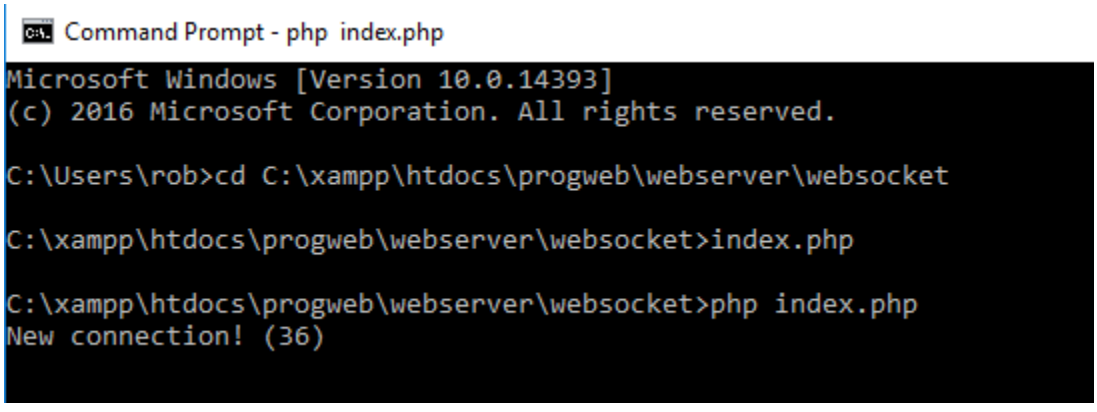


Figure 0.14 Websocket connection received from AHCF

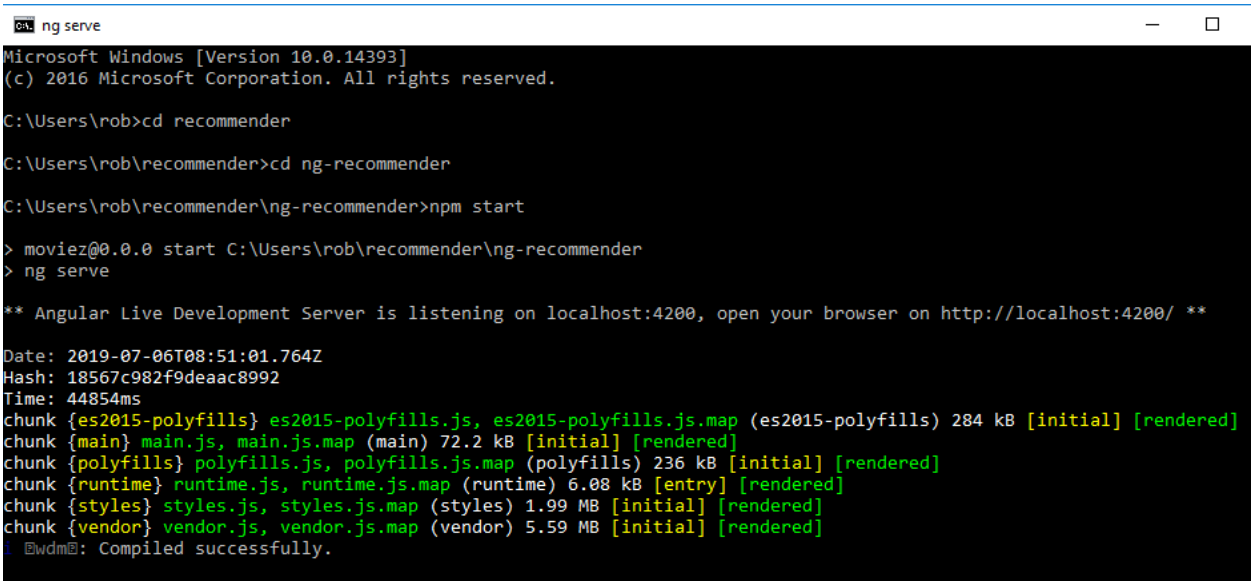


Figure 0.15 AngularJS front-end web framework

The final server is the database server (MySQL) from which the AngularJS and PHP backend display information. The output of the recommendation engine is sent to the database, and some aspects of the various datasets such as the rating information of users are loaded into dataframes that are inserted into the database once the engine is started.

When these servers are started successfully, the system begins to meet the design requirements outlined in the thesis.

The top-rated movies, which represent movies with the highest ratings, are displayed first for all users. The rating scale is reduced to 1-10 with 10 being the highest rating. This information is obtained by running queries on the database after the data is inserted from the recommender engine. The top 10 is displayed in 4 columns (Figure 0.16) with the intention of attracting and not overwhelming a new user on first-time encounters.

Figure 0.16 New user: Top rated movies

The idea behind addressing the cold start is to guess the possible choices a new user will subscribe to, which is difficult, so the addition of the best rated (Figure 0.17), which is slightly different from the top-rated but follows the same approach is implemented. In this regard, an average of all high individual ratings are taken and the movies which score well are displayed.

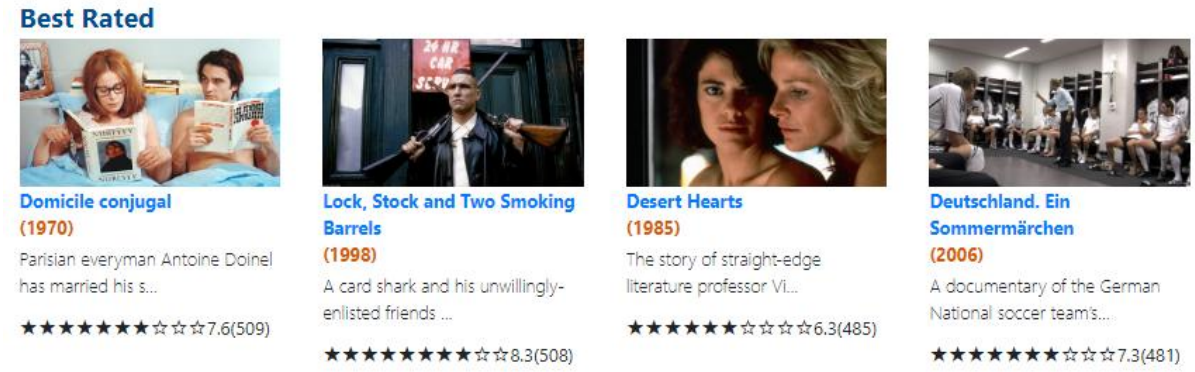


Figure 0.17 Best rated movies

The featured movies represent movies from the clustering result. Movies are picked randomly and displayed from the various clusters as displayed in Figure 0.18

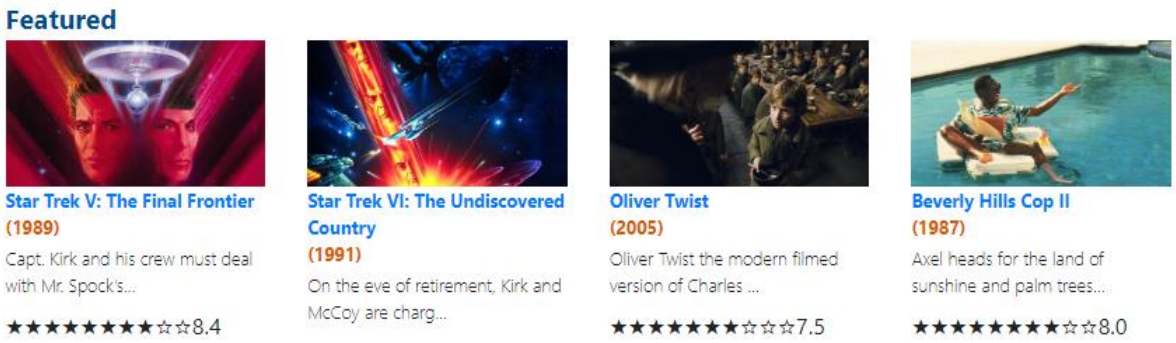


Figure 0.18 Featured movies (movies from the different clusters in the database)

A new user to the system or users who arrive by incognito is presented with the views above. This simplified organization and presentation is bound to enhance user retention on the platform. In addition to that, such a user can rate movies they are interested in but will not receive personalized recommendation until they register or become users. This feature is intended to enhance user curiosity. To the right-hand side of the movie banner (Appendix A), our LDA implementation for grouping words or genres unrestricted to a single choice is implemented. The primary purpose is for example, helping a user to select movies that to a large extent, fall in more than one category. This is depicted below in Figure 0.19.

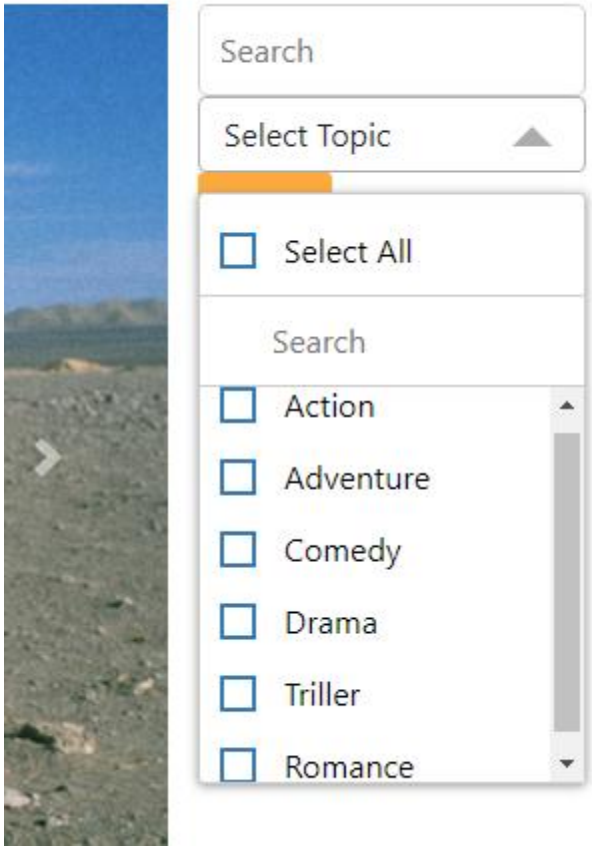


Figure 0.19 LDA movie search

Upon successful registration and login, as presented in Figure 0.20, the user gets non-personalized similar movie recommendations as in Figure 0.21, can watch movie trailers as in Figure 0.22 and select movies. As the new user select moves at random or by preference, the similar movies are re-clustered and updated from the database. The user can rate movies (Figure 0.23) to enhance predicting user choices or to generate personalized recommendations in the "recommended for you" section in Figure 0.24.

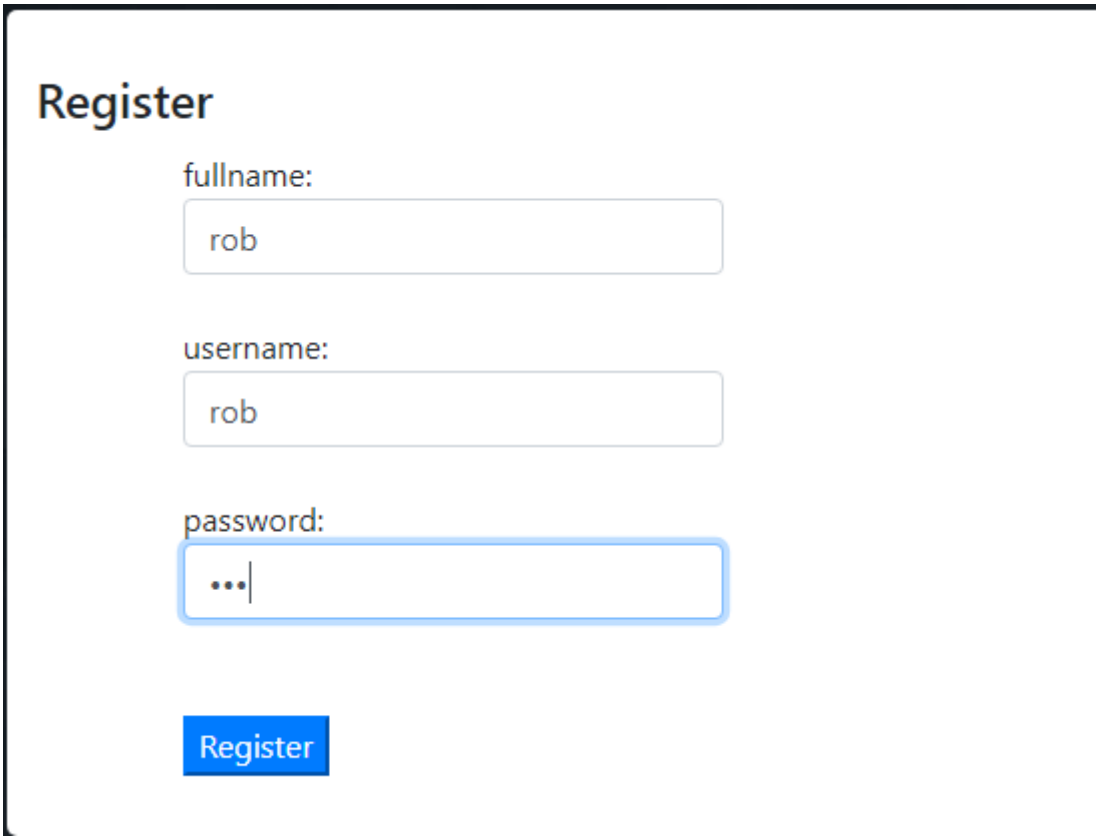


Figure 0.20 Registration page

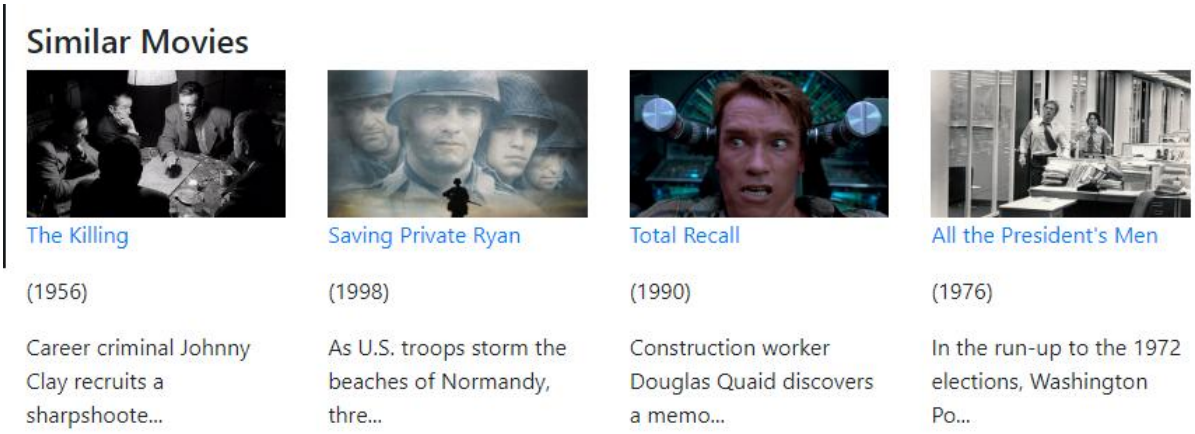


Figure 0.21 Similar movies

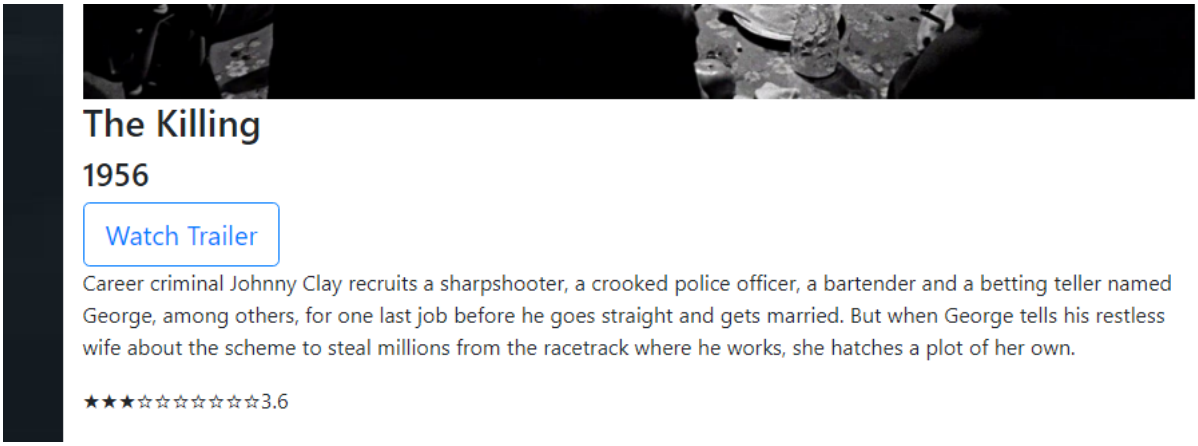


Figure 0.22 Watch Trailer

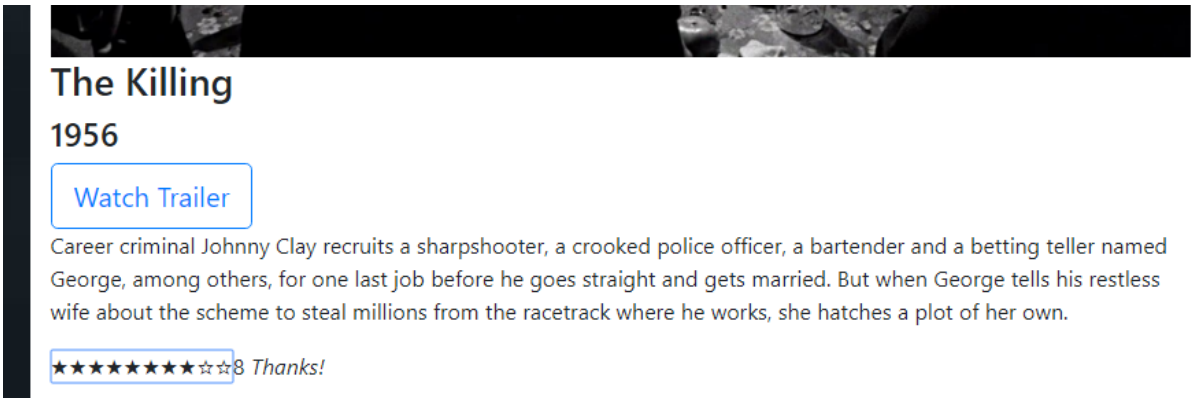


Figure 0.23 Rate a movie

A new user does not get very personalized movie recommendations in the "Recommended for your" (Figure 0.24) until ratings are giving but gets non-personalized similar movie recommendations.

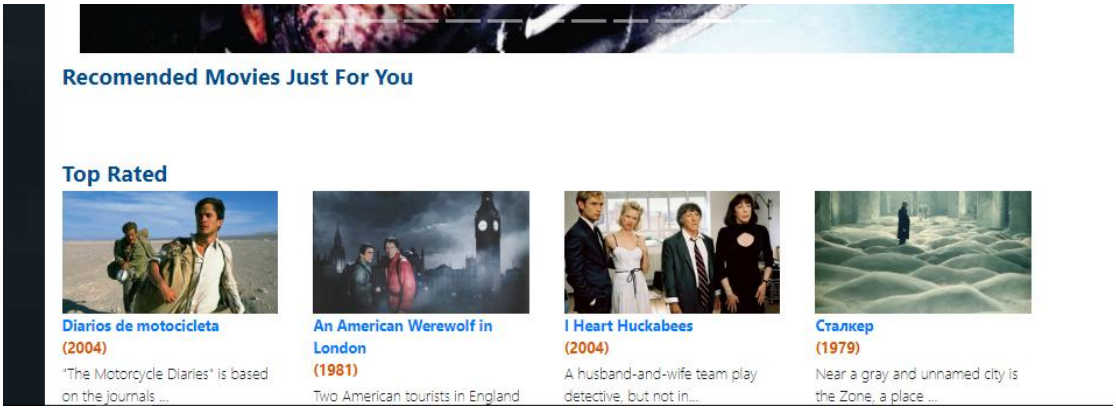
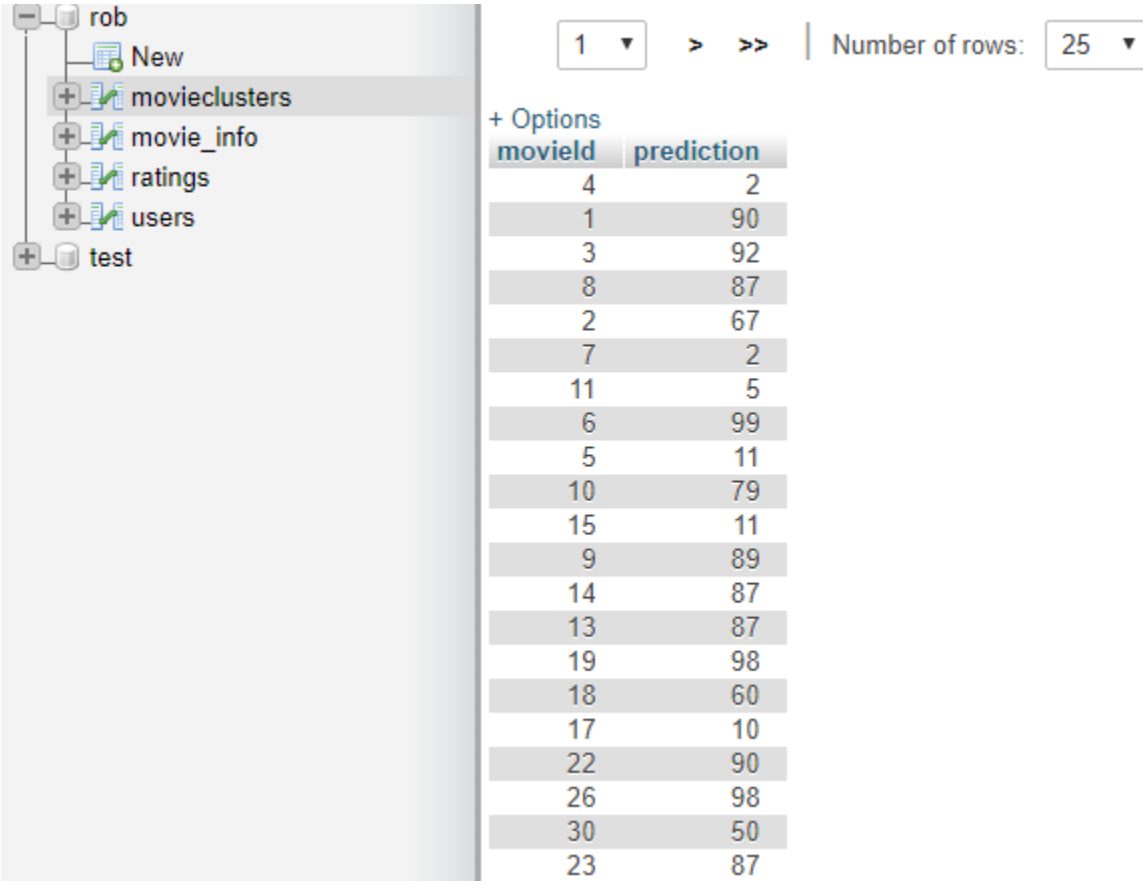


Figure 0.24 No result for a personalized recommendation for new users

Displayed below is the database for the movie clusters. Upon successful running of the recommender engine, the KMeans algorithm predicts the cluster every movie belongs to and assigns cluster number or Id (prediction value) to that movie. Figure 0.25.



movieId	prediction
4	2
1	90
3	92
8	87
2	67
7	2
11	5
6	99
5	11
10	79
15	11
9	89
14	87
13	87
19	98
18	60
17	10
22	90
26	98
30	50
23	87

Figure 0.25 Database movie cluster setup

The significant role of the KMeans streaming is to update the cluster centers, which affects the predicted group value, thus re-clustering the movies in the database. For example, a query in our database for movies in cluster group 90 is displayed in Figure 0.26 below. It can be seen here that this cluster contains only 10 movies. The value of k is 100 meaning there are 100 cluster groups.

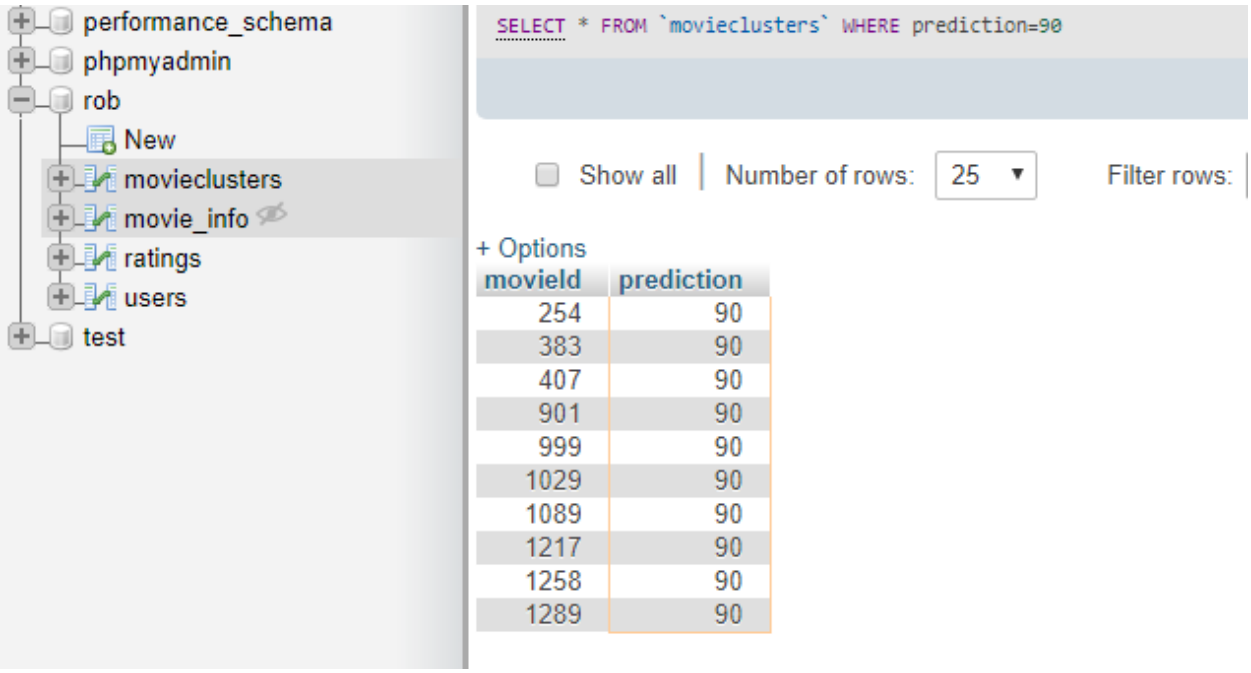


Figure 0.26 Selecting Cluster group 90

Now, as a user begins to rate movies, the cluster centers are updated by the KMeans streaming, and a user is required to rate 5 movies, which represent the movie batch number used by the KMeans streaming to perform the cluster computations. As the user rates movies, the rating, user

and movie Id is passed to the recommender engine presented in Figure 0.13(previous). After the 5th rating is made on movies, the cluster computation begins until completion in Figure 0.27.

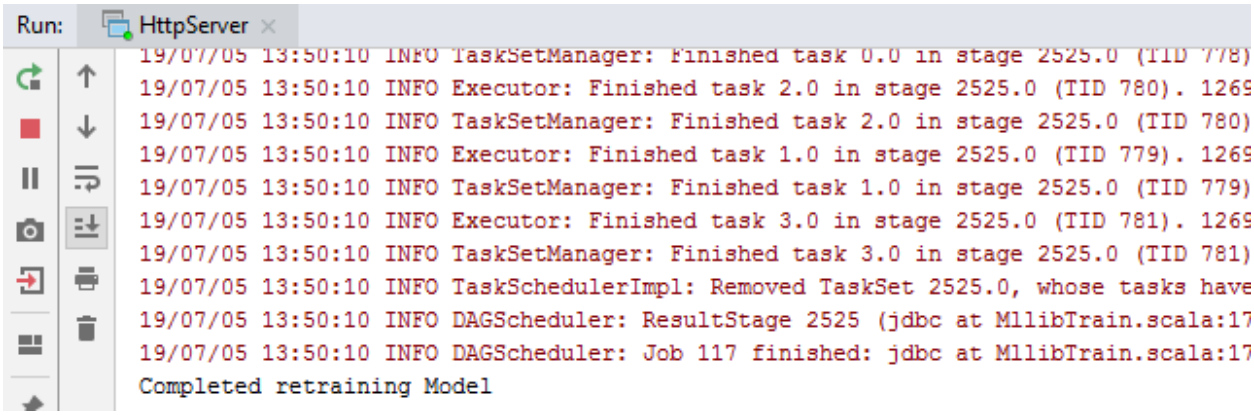


Figure 0.27 Retraining the Model with batch ratings

The movies are then re-clustered in the database. In the previous example, cluster group 90(Figure 0.26 -previous) had 10 movies but after re-clustering, it has only 4 movies in Figure 0.29 in this case, totally different from the previous 90th cluster group.

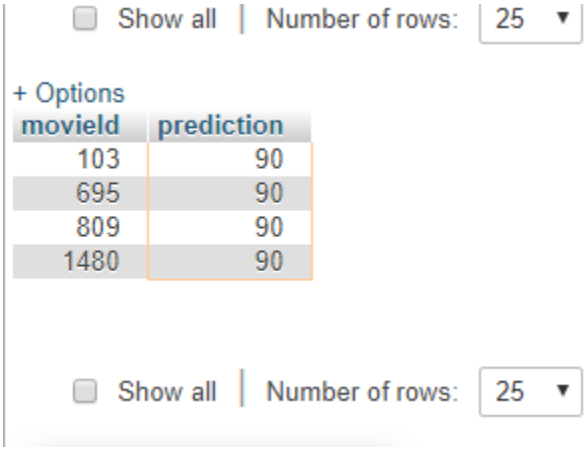


Figure 0.28 Cluster group 90 Output after retraining

Figure 0.32 also follows in the example of recommending movies for users in cluster group 90. Another random cluster 71 is selected to check the movies in the cluster group. Upon re-clustering, the results in Figure 0.31 is obtained.

+ Options

movieId	prediction
72	71
80	71
90	71
102	71
105	71
277	71
384	71
363	71
395	71
401	71
455	71
590	71
585	71
768	71
743	71
822	71
831	71
940	71
944	71
931	71
1027	71
1162	71
1321	71
1492	71
1459	71

Figure 0.29 Selecting Cluster group 71

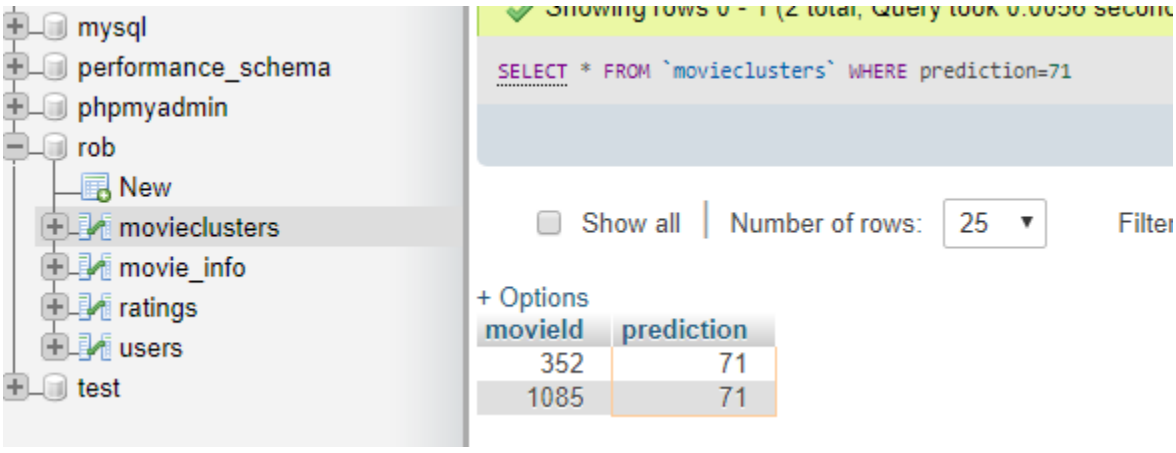


Figure 0.30 Cluster group 71 after retraining

Figure 0.31 Recommended for you (personalized recommendation)

Using the scenario for an already existing user, the user receives personalized recommendations (Figure 0.31) on successful login, can get topic related searches (Figure 0.32), watch trailers, and get Figure 0.33similar movie recommendations based on movies previously (Figure 0.34).

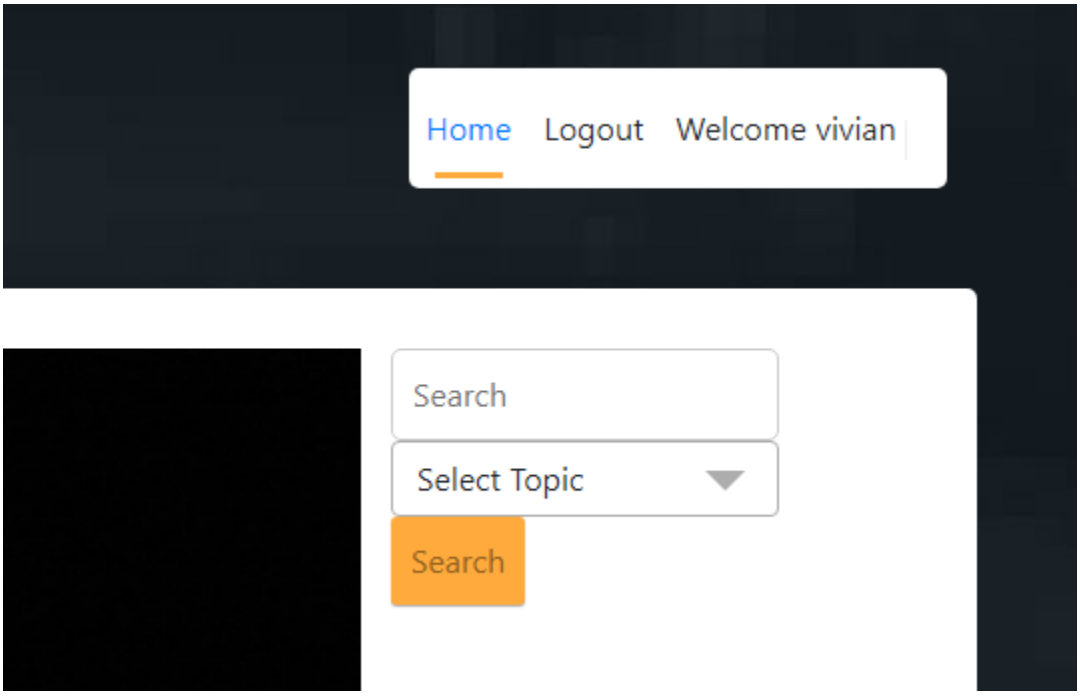


Figure 0.32 LDA search for existing user

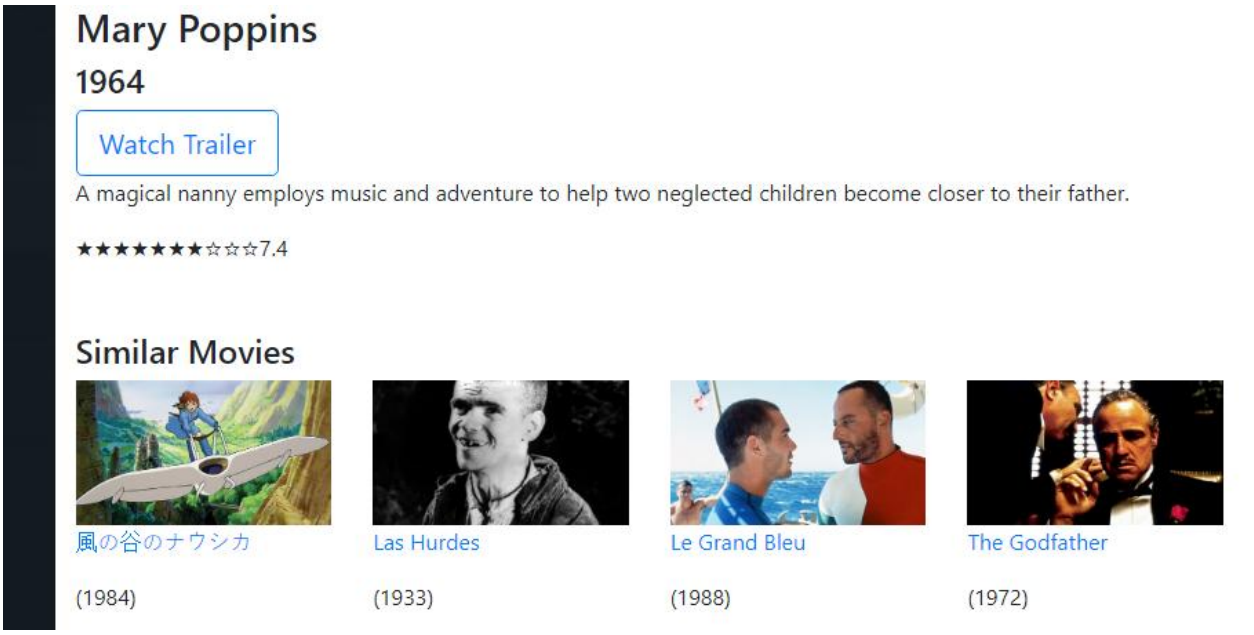


Figure 0.33 Similar movies for existing user

As the existing user rates more movies, after every 5 ratings, the movies are updated to generate a more personalized recommendation for the user. The rating information is sent to the recommender engine (Figure 0.34) until completion Figure 0.35

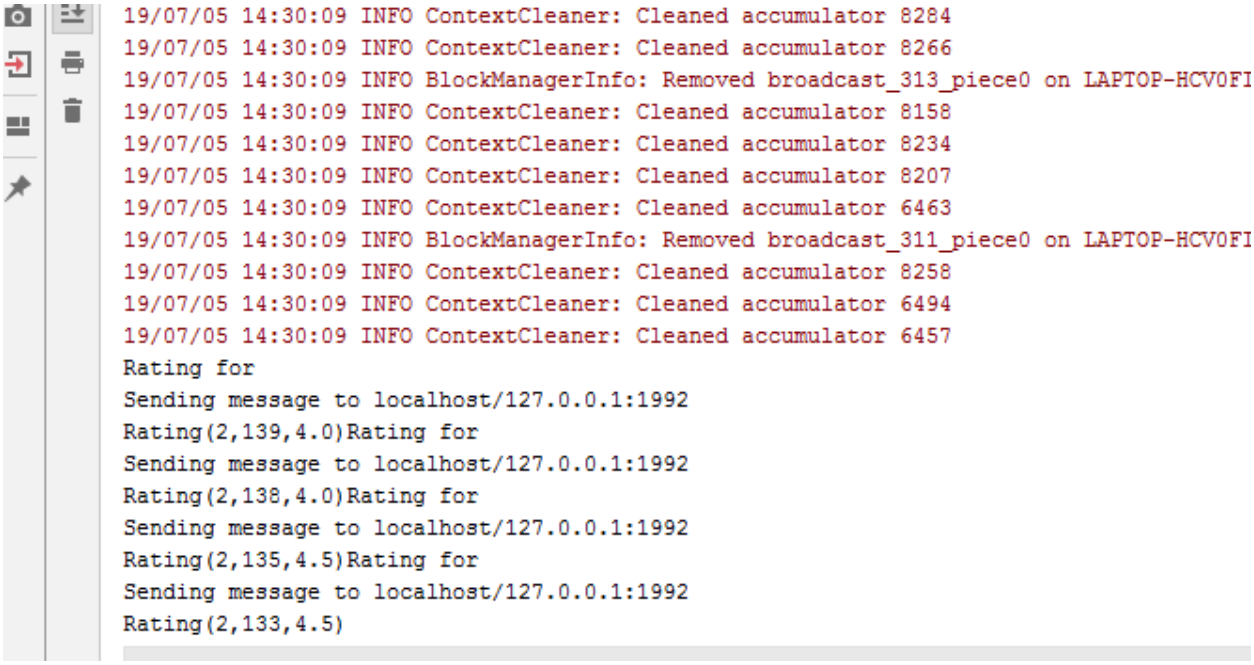


Figure 0.34 Recommender Engine receives user rating

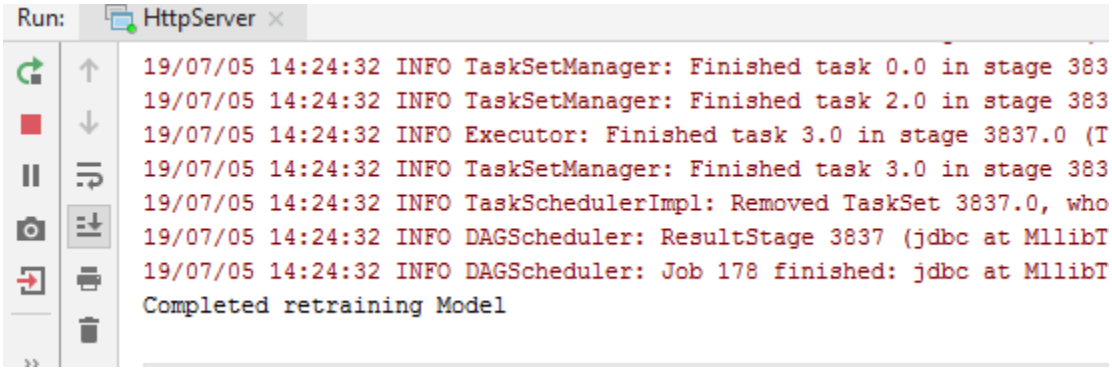


Figure 0.35 Retraining the model

Cluster group 21(has only one movie) and 0(has more movies) in Figure 0.36 and Figure 0.38, respectively, are randomly selected to view movies in that cluster. After re-clustering, the results are displayed in Figure 0.37and Figure 0.39. This shows that the cluster sizes can change from a large cluster to small or small to big depending on user preferences.

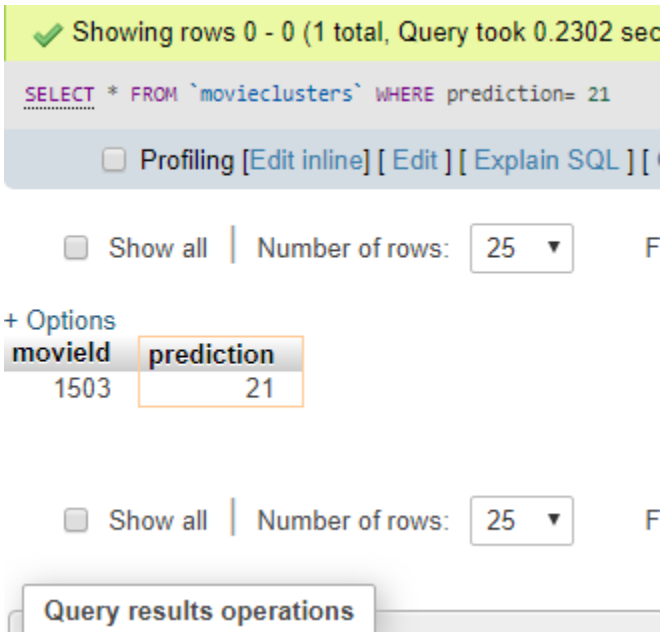


Figure 0.36 Cluster group 21

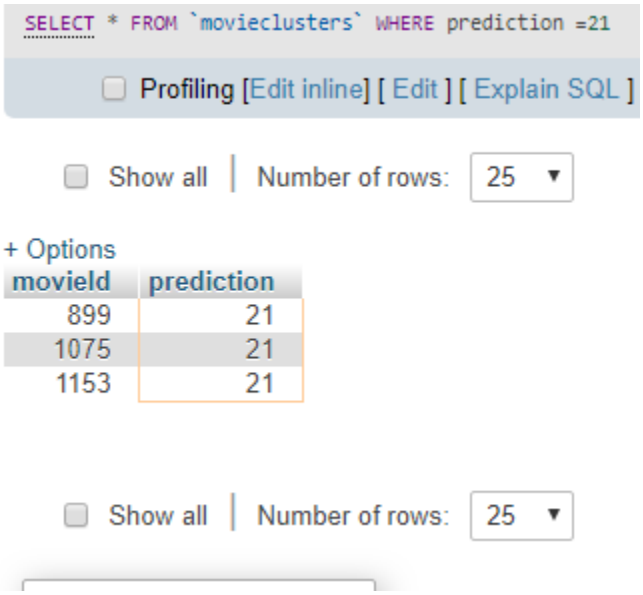


Figure 0.37 Cluster group 21 after retraining

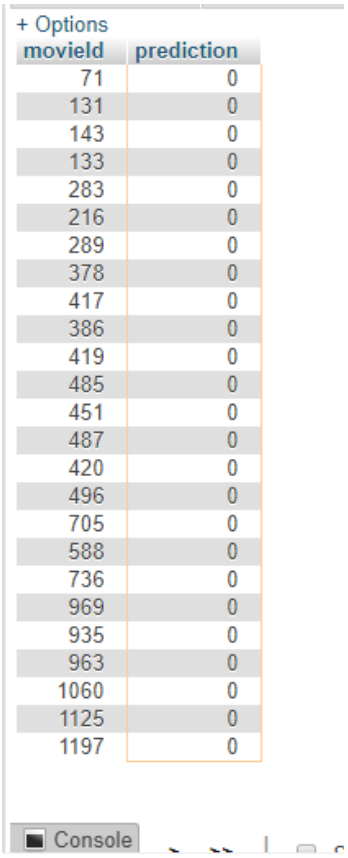


Figure 0.38 Cluster group 0

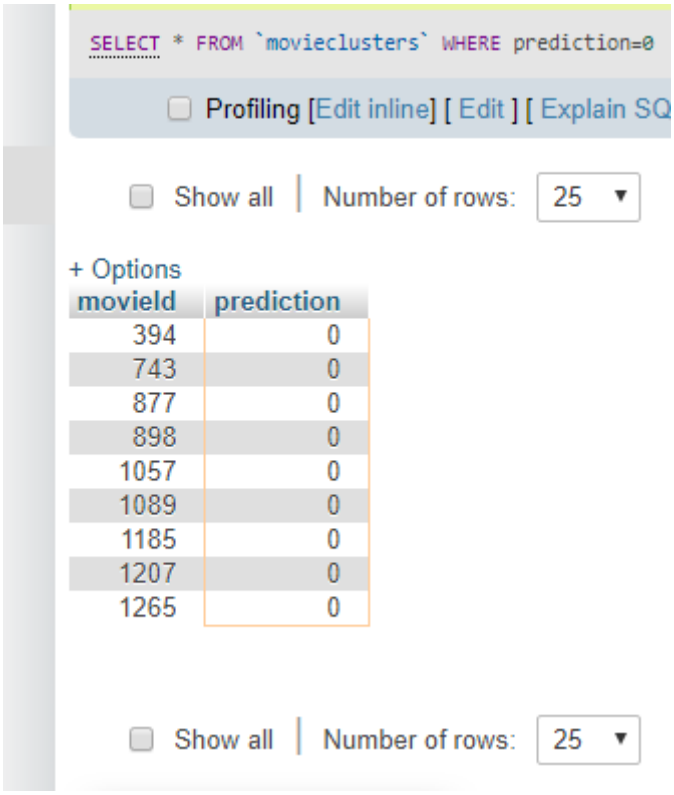


Figure 0.39 Cluster group 0 after retraining

4.3.8 Generic Nature of the Proposed AHCF

After drawing inferences on the movieLens datasets, further investigations are carried out on various datasets to show that the proposed implementation is not restricted to the movie domain only, but, equally useful in any domain that accommodates textually rich datasets and/ or ratings from platform users. Hence, the proposed AHCF is applied to book, restaurant, anime, dating agency, and Amazon datasets[49]. The results were compared with the 1M results of the MovieLens dataset.

The Bookcrossing dataset contains 272,679 interactions from 2,946 users on 17,384 books with 62,657 ratings on 14,684 books by 1295 users(ratings are between 1-10). It also contains historical information and demographics(age, gender, occupation, zip). The dataset was organized by Cai-

Nicolas Ziegler in a 4-week crawl (August / September 2004). Preprocessing steps similar to MovieLens 100K treatment was performed on the dataset [2],

The anime dataset consists of ratings on 12,294 anime from 73,516 users reduced in Python with a random seed of 123. It contains 520,610 interactions (play / purchase) from 5,000 users on 7,718 anime, history information and ratings from 4,714 users on 7,157 anime (419,944 interactions).

Libimseti dating agency dataset consists of 17,359,346 anonymous ratings of 168,791 profiles by 135,359 Libimseti users, gathered on April 4, 2006, with a rating scale of 1-10. Users with more than 20 ratings were used, and users who gave constant ratings were removed.

Amazon dataset contains more than 2M reviews and ratings concerning Beauty products (ratings range from 1-5). Product reviews (ratings, text, helpfulness votes) that have 142.8 million reviews spanning May 1996 - July 2014 and Amazon metadata (descriptions, category information, price, brand, and image features) are made available online.

The Zomato dataset provides a vast array of textual data ranging from reviews, cuisines, votes, names, rate, menu items, and many others. There exist no metadata about the dataset currently.

The datasets were passed through the AHCF recommender engine. Standard and preferred choice values inferred from the previous experimentations were used. For ALS, a rank of 10 and an RMSE metric was used. For KMeans, the WCSS metric was used and a k value of 100.

The bookcrossing, Amazon beauty, Libimseti dating, and anime datasets were passed through the offline and adaptive model because they are based on ratings and produced the following results for the KMeans, ALS and computation front:

4.3.8.1. The ALS results comparison for Various Datasets

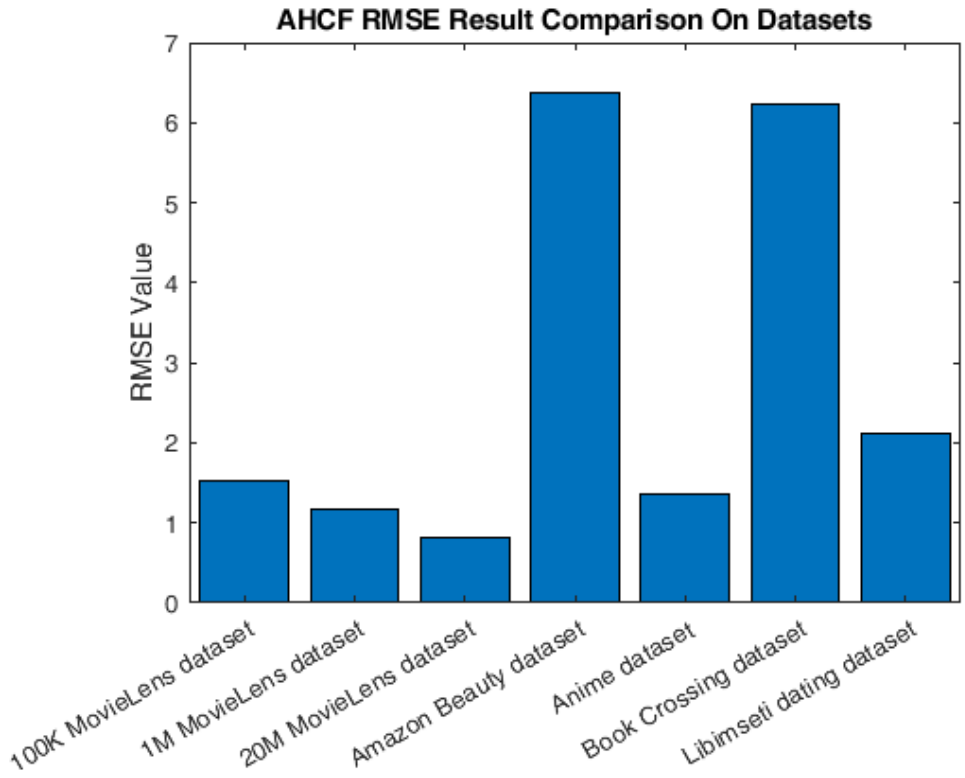


Figure 0.40 AHCF RMSE result comparison on Dataset

From Figure 0.40 the RMSE values are best with the huge size of the dataset, which suggests that the algorithm developed is mainly dependent on the size of the dataset.

4.3.8.2. Dataset LDA Comparison

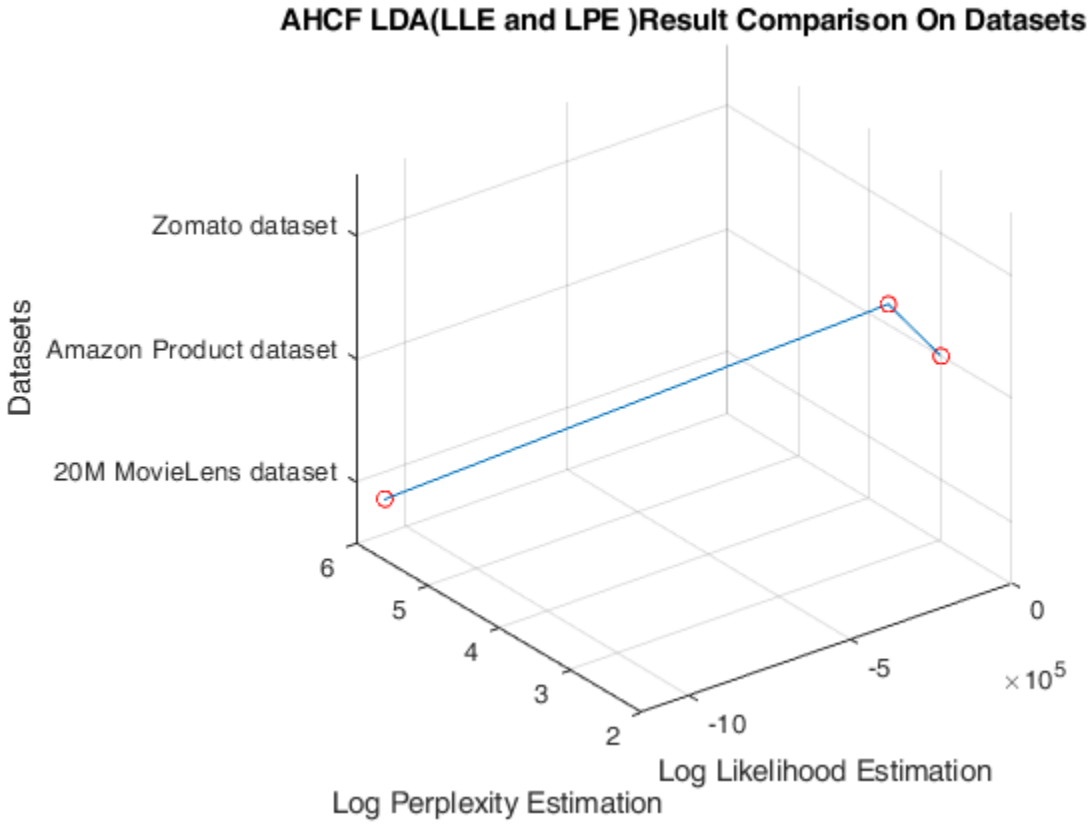


Figure 0.41 LDA Dataset Comparison

The lowest log perplexity is recorded by the Amazon products dataset represented by the red marker at 3. The 20M dataset, however, has the highest LLP and LLE results in Figure 4.41.

4.3.8.3. Datasets computational time comparison

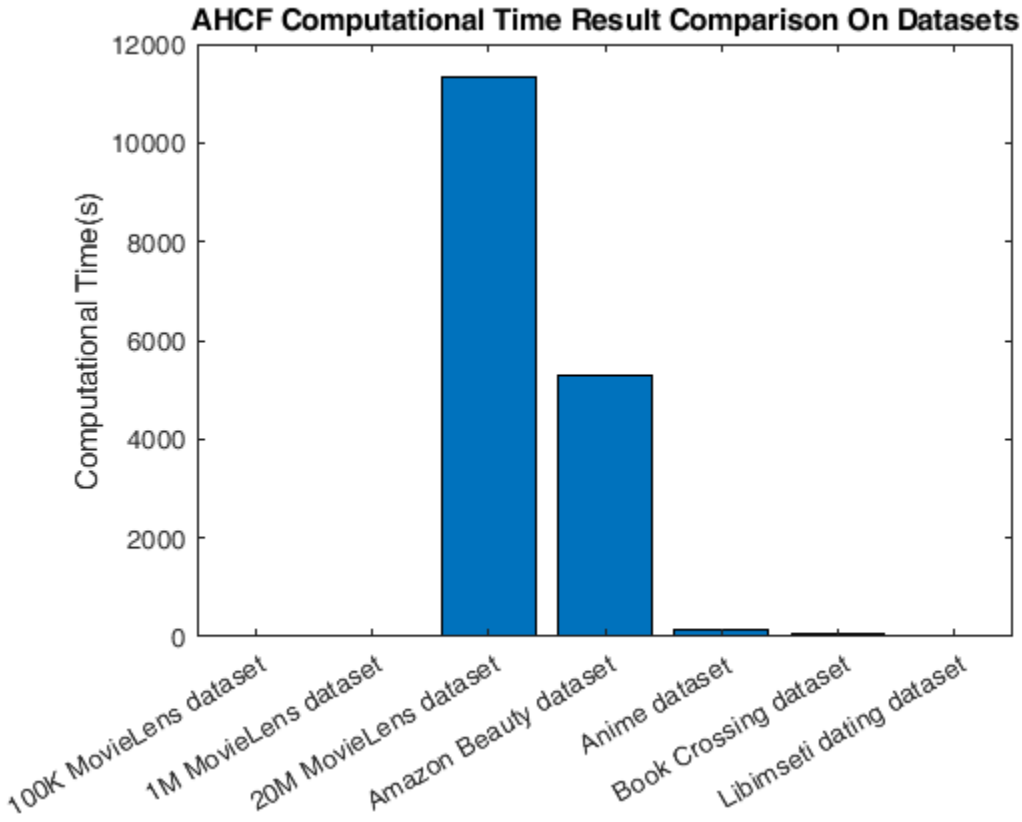


Figure 0.42 Datasets computational time comparison

In terms of the computational time, it can be seen from Figure 0.422 that the lesser datasets have very small computational time cost as compared to the 20M movieLens and Amazon Beauty results huge amount of training time is required to predict unknown ratings and cluster movies, alongside the intermediate transformations and cleansing of the data.

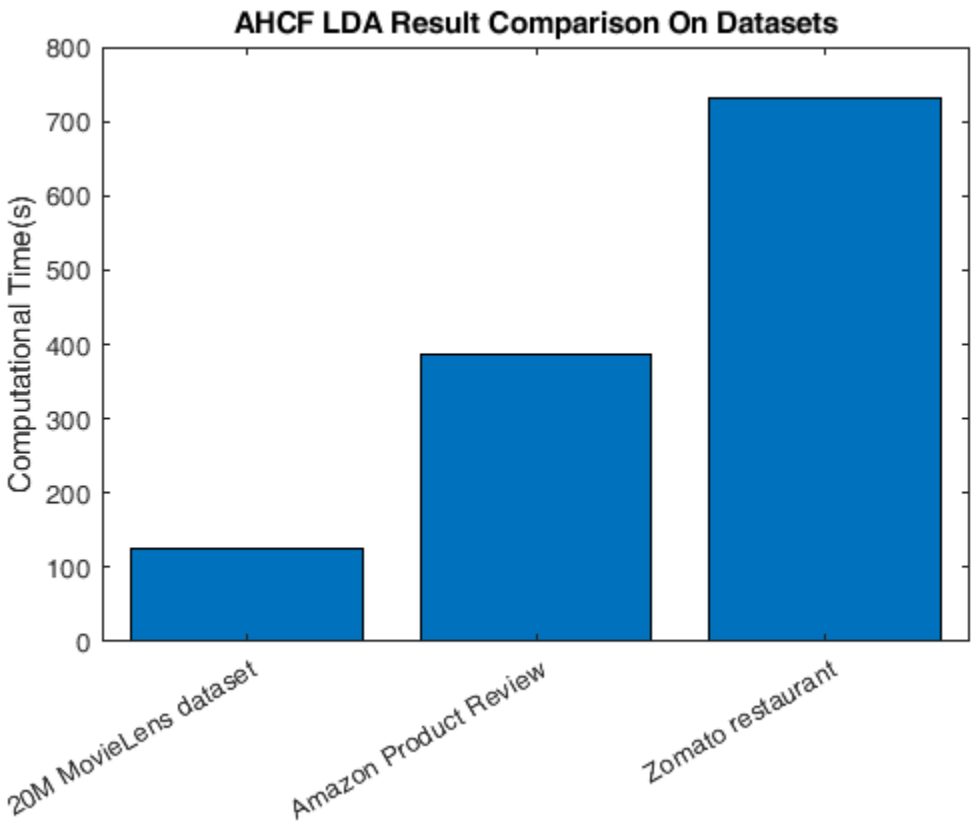


Figure 0.43LDA results comparison on various Datasets

The LDA computational time (Figure 0.43) results indicate that the 20M dataset although huge has very little textual information, in this case, just 2 to 3-word descriptions or tags from users

4.3.9 Bench Mark Comparison

The AHCF developed is further compared with other open-source benchmark results. The surprise benchmark specifically provides information for RMSE and computational time for 1M and 100K movieLens datasets on various algorithms which can be useful in comparing the efficiency of the AHCF model developed (Figure 0.44)

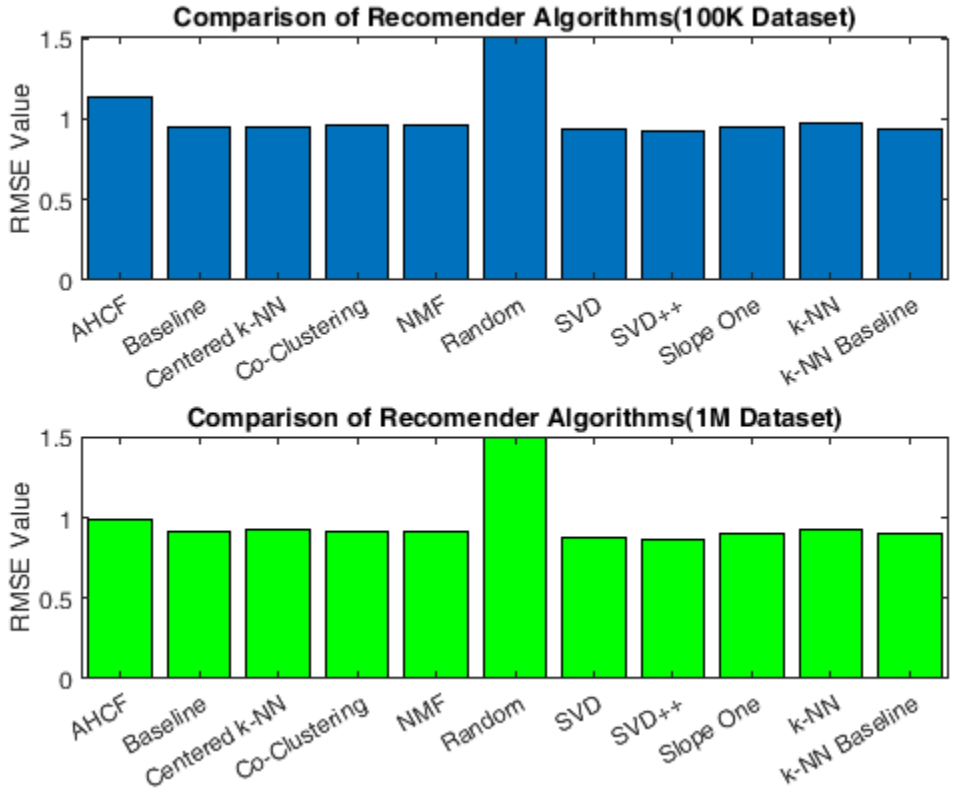


Figure 0.44 Comparison of AHCF with other recommender algorithms (RMSE)

The SVD++ result is taken out in the comparison (Figure 0.44) because it has an extremely substantial computational time of 543s and 9945s on 100K and 1M MovieLens dataset respectively, to enhance the careful considerations made for Algorithms within a measurable margin. Thus, it can be concluded that in terms of RMSE values, the AHCF is reasonably accurate in its prediction and within the range of values for standard RMSE results in Figure 0.44 above.

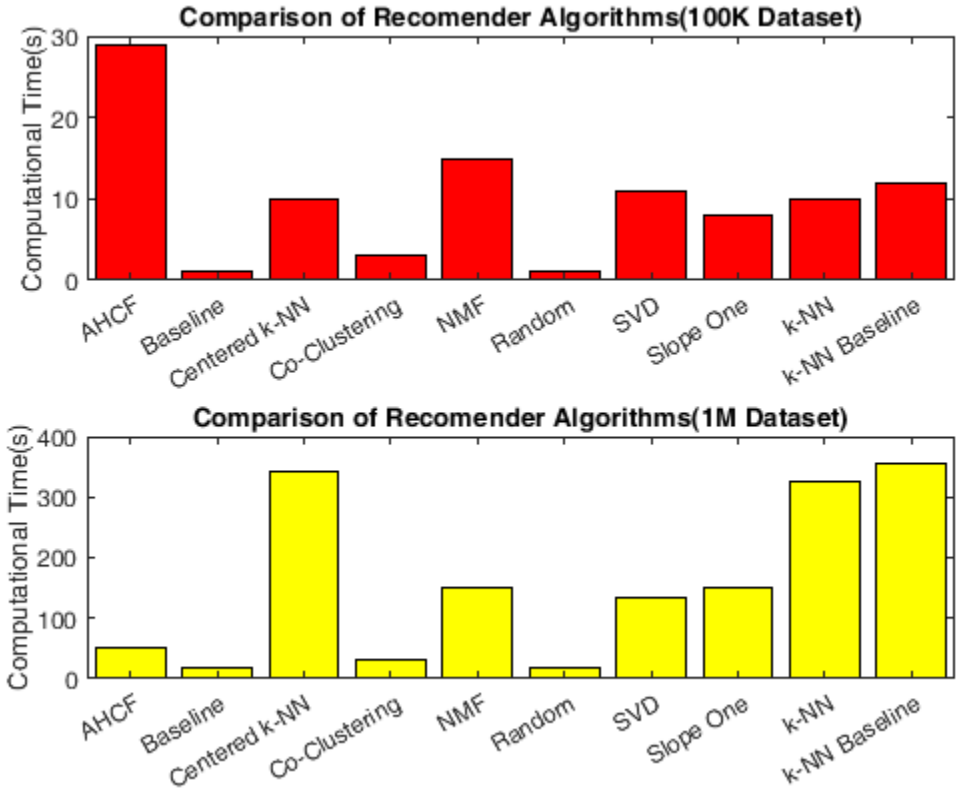


Figure 0.45 Comparison of AHCF with other recommender algorithms (Computational Times)

From Figure 0.45, the AHCF implementation works very efficiently with an increased dataset but performs poorly with fewer data. This makes it even more useful since data is ever increasing.

CHAPTER 5

RESULTS AND DISCUSSION

5.1 Introduction

The chapter emphasizes findings of the thesis research work and further outlines some important observations, further statistical inferences or data analysis, as well as discussion of results. This discussion will proceed as follows: The analysis of data with its conclusions and discussion of results in line with the thesis objectives, relevance and problem statement. The interpretation of the result from section 4.3 will also be highlighted. Other interesting discoveries will also be brought to light in the discussion segment.

5.2 Data Analysis

The 90:10 and 80:20 cross-validation results are used, with the 1M and 100K datasets for further data analysis. This is to justify and validate the choices in section 4.3.3.1. A one-way analysis of variance (ANOVA) is used to determine whether the computational times for the 90:10 and 80:20 cross-validation results and the 1M and 100K datasets have conventional means respectively. This enables the research to find out whether the different groups of independent variables have different effects on the response variable(computational time). The one-way analysis of variance (ANOVA) is a simple case of the linear model. It tests the hypothesis that all group means are equal versus the alternative hypothesis that at least one group is different from the others. The median represented by the red notch in Figures 5.1, 5.2 and 5.3 explains the conclusions drawn in section 4.3.3.1.It can be seen from Figure 5.3 that the larger the dataset, in this case, the 1M dataset, the less the computational time this is mainly because the ALS algorithm works more efficiently with more data. In Figure 5.1, the computational times are almost close but with the 90:10

dataset having a lower median. However, as the dataset increases in Figure 5.2, the 80:20 performs better.

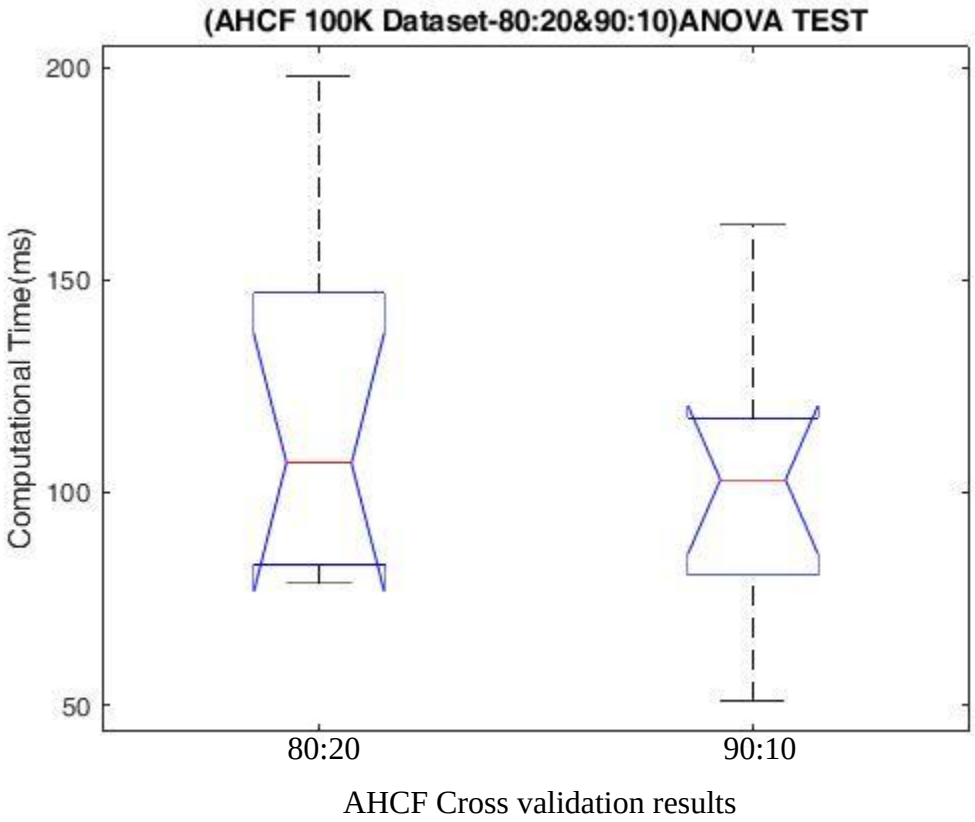
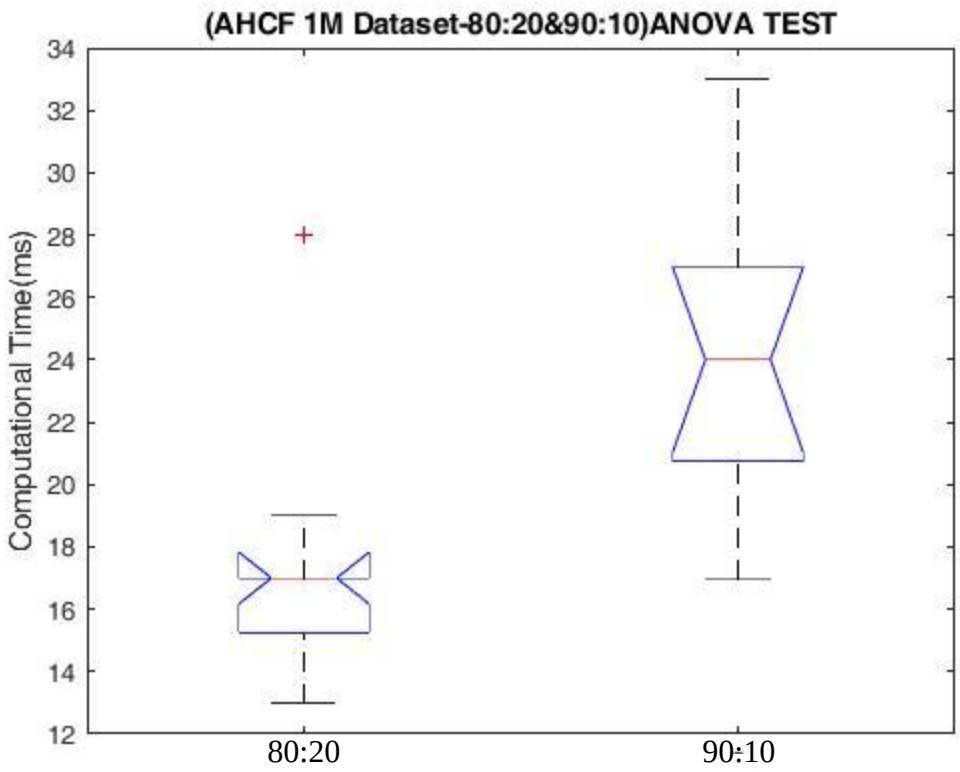
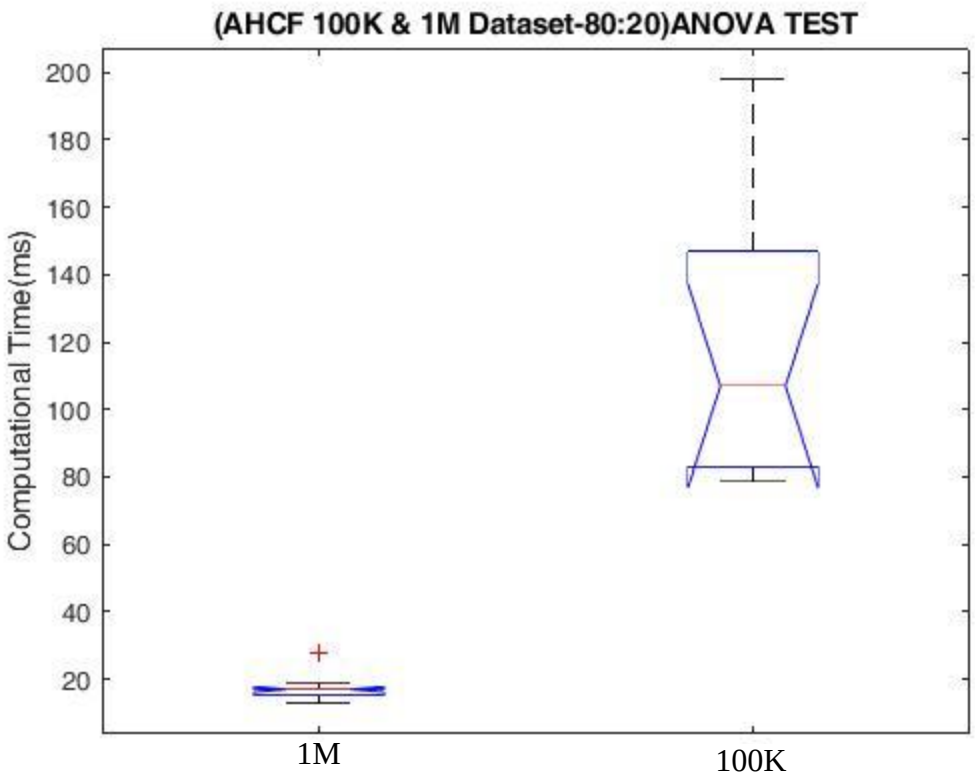


Figure 5.1 AHCF 100K Dataset Cross-validation results ANOVA test



AHCF Cross validation results

Figure 5.2 AHCF 1M Cross validation results ANOVA test



AHCF 1M and 100K datasets

Figure 5.3 AHCF 1M and 100K datasets ANOVA test

5.3 Discussion of Results

A unique contribution has been made by realizing a novel hybrid collaborative filtering recommender engine, highly adaptive to changes in user preferences almost instantaneously. It addresses scalability, sparsity, and extensively tackles the cold-start problem. Other side issues tackled include the overspecialization and over-fitting issues associated with recommender systems. A one size fits all is impossible when it comes to predicting user preferences or to decide what to present and recommend to a user correctly and thus the need for a combined multi-hybrid system that can meet the varied and unique demands of consumers has been implemented primarily via the incorporation of the topic model (LDA). This has leveraged both textual and rating information that proves informative for making a recommendation

The ALS results clearly are affected by the cross-validation results, while it is evident that the KMeans algorithm is not significantly affected by the cross-validation but primarily by the number of clusters represented by k.

In terms of performance, the 90:10 cross-validation result is superior in the ALS followed closely by the 80:10 and in terms of computation, they perform reasonably well. For the KMeans, the WCSS values indicate that it is helpful to have more number of clusters (K) that will give a desirably low WCSS. Nonetheless, the tradeoff between the times needed to compute these clusters are also of significant importance as a short time interval for making recommendations is a desired result hence the need to cap our results somewhere between 50 and 100, thus improving the computational time of the hybrid approach while ensuring quality of recommendations made to a user. The user- movie clustering seems to provide larger values for the WCSS on user-movie as compared to the movie-movie clustering, which means that it is more discriminatory of features and more efficient in clustering compared to the movie-movie clustering.

The AHCF implementation performs equally well as compared to the other algorithms in terms of prediction accuracy. It reduces the time needed for retraining, which is key to updating models by half as compared to the existing offline model. This suggests that streaming efficiency and real-time update of recommendations for users is drastically reduced. It can also be concluded that the AHCF developed leverages implicit information (text mined by the LDA) and explicit information (ratings) in making recommendations.

The simplicity and organization of the web portal attract users while encouraging them to be active in rating movies to get a personalized recommendation. This helps draw customers to such platforms. The liberality and the combination of many approaches enhances the chances of getting a user to like at least an item from the list, as recommendations come in 4 ways, either from similar

movies from clustering, best rated and top-rated from normal database queries, LDA text search options or adapted clustering peculiar to user preference ratings. This makes it evident that the entire system is user-centered and customer-focused, as is expected in the business world.

The results from the various dataset also indicate clearly that the AHCF can be applied in many useful domains that leverage ratings and textual data in the form of reviews, among others.

It is also worthy to note that the AHCF was experimented on using 2 different operating systems (OS); Ubuntu 18.4 and Windows 10, which make it platform-independent, once properly configured. However, windows are extensively used in this research. Due to its generic nature as proven by testing on various datasets, the AHCF developed can serve as an add-on or plug into many recommender models, especially in recommendation systems that employ similar dataset characteristics to the ones depicted in section 4.3.7. Hence, any recommendation system implementation that possesses dataset features identical to the ones used in this thesis can either solely depend on the AHCF or be used side by side with others.

CHAPTER 6

CONCLUSION AND RECOMMENDATION

6.1 Introduction

The chapter summarizes the thesis research work and some challenges, observations, and room for future work in the domain. The Contribution to knowledge, Observation, and Recommendations will be preceded the Conclusion in organizing the chapter. These will generate inferences, in line with the thesis objectives, the relevance of the thesis, and the problem statement of this thesis. The chapter will sum up the very essence and justification for this thesis research.

6.2 Conclusion

In summary, this research has made methodological and managerial contributions. In terms of methodology, a novel hybrid collaborative model combining an offline phase and adaptive phase, content and collaborative approaches and the use of LDA model to address core issues affecting recommendation systems while enhancing personalized recommendation for users in the movie domain is developed to solve the peculiar problems fundamental to this research that include computational time lapses in responding t o user choices instantaneously, the slow time scale of updating the recommender system, the various phases of the cold start problem outlined and efficiently managing the tradeoffs between scalability, sparsity and the cold start problem.

The model has improved performance on retraining recommender models by reducing the computational time by 50% compared to baseline offline implementations, thus addressing the problems of computational time lapses in responding to user choices instantaneously, the slow time scale of updating the recommender system. Managerially, the model is useful for acquiring an understanding of consumer preferences on user-product recommendations such as targeting, personalization, and segmentation, which are major marketing activities. The research enhances

business profits since it impacts positively on the first impression of users, user retention, or promotion of unpopular or unknown items due to the organizational structure adopted in the online phase of the implementation.

Subtly, recommender systems reduce user/customer frustration as they aim to help them get exactly what they are hoping to acquire on using the various platforms. The engine developed is also made available for open-source use, thus, helping to meet the business needs of an organization.

The model presented represents an entirely different approach to making recommendation that leverages ratings and textual information jointly by its multi-hybrid nature; thus, the various phases of the cold start problem outlined in the problem statement are managed and the efficiency desired in managing the tradeoffs between scalability, sparsity and the implementation and methodological framework adequately handle the cold start problem.

In this dissertation, a unique contribution was made by realizing a novel hybrid collaborative filtering recommender engine, highly adaptive to changes in user preferences almost instantaneously. It addresses scalability, sparsity, and extensively tackles the cold-start problem. Other side issues tackled include the overspecialization and over-fitting issues associated with recommender systems. Also, a topic model (LDA) with user-generated product tags and textual description is implemented. It is evident that a one size fits all is impossible when it comes to predicting user preferences or to correctly decide what to present and recommend to a user and thus the need for a combined multi-hybrid system that can meet the varied and unique demands of consumers.

In a nutshell, the AHCF developed satisfies the initial objective to design, develop, and deploy an adaptive multi-hybrid recommender system. The objective to evaluate the computational

efficiency in terms of speed and memory footprints has been realized and proven by the ability of the AHCF to reduce computational time by 50% compared to baseline offline implementations. The evaluated accuracy metrics and implementation of multiple algorithms (Root Mean Square Error for the ALS, Within Cluster Sum of Squares for KMeans, Log Perplexity and Log-Likelihood in the LDA) and the memory footprints and on retraining the model were recorded for the KMeans streaming in line with project objectives were realized with the AHCF producing an accuracy measure of 0.88-3.0 on RMSE values for chosen datasets on increasing rank but less than 8 for the 5 other datasets adopted. A movie web portal was designed for testing and evaluation as depicted in section 4.3.5. The AHCF was also applied to 5 other datasets (on restaurant datasets, anime, dating datasets, books and e-commerce) as suggested in the thesis objectives. However, the 100K and 1M movieLens datasets and some portion of the 20M dataset constitute the primary or anchor dataset employed in the research. The AHCF implementation is compared with Centered k-NN, Co-Clustering NMF, SVD, Slope One, k-NN, k-NN Baseline approaches in section 4.3.8 in line with the thesis objectives.

The research has demonstrated the potentials that can be harnessed in bringing together algorithms that help create a complete recommender system, that models, to a large extent, the dynamic user needs and preferences for a real-world domain it addresses the cold start problem, sparsity and scalability issues associated with making recommendations extensively.

The results in line with the thesis objectives enhance the confidence level in concluding on the accuracy of the proposed model. The ability to respond to real-world data input by implementing the streaming KMeans enhances the relevance of this research to machine learning. The movie search and recommendation domain serve as the application domain for this model. And the generic nature of the algorithm developed makes it more useful in many business recommendation

domains. Real-time update of user recommendation for specific batches of ratings while reducing the computational time of retraining by 50% makes this implementation efficient.

6.3 Contribution to the body of knowledge in recommender systems

The research contributes to harnessing multiple machine learning techniques in recommender systems using an adaptive hybrid approach. The coupling of the offline and online models in the adaptive collaborative filtering technique adopted as well as the efficient merger of machine learning techniques for making recommendations based on textual data and rating information constitute significant contributions of the research.

The AHCF implementation performs equally well as compared to the other algorithms in terms of prediction accuracy. It reduces the time needed for retraining, which is key to updating models by half as compared to the existing offline model. This suggests that streaming efficiency and real-time update of recommendations for users is drastically reduced. It can also be concluded that the AHCF developed leverages implicit information (text mined by the LDA) and explicit information (ratings) in making recommendations.

Due to its generic nature as proven by testing on various datasets, the AHCF developed can serve as an add-on or plugin to many recommender models, especially in recommendation systems that employ similar datasets that are characteristic to the ones depicted in section 4.3.7. Hence, any recommendation system implementation that posses dataset features similar to the ones used in this thesis can either solely depend on the AHCF or be used side by side with others.

6.4 Observations

The Scala programing language is a relatively new paradigm that requires extensive work to promote the efficiency of machine learning algorithms. The algorithms in themselves are computationally expensive making it difficult to have complete online deployments or take

advantage of their online models. The use of intuitive design decisions of these algorithms makes them difficult to standardize, considering, eyeballing of topics, prior selection of k and rank values are all dependent on subjective human reasoning which hampers the system's ability to be formalized as perceptions differ among individual researchers. The system developed does not take advantage of demographic data as this can be useful especially in dealing with privacy issues and what content is valuable, especially in dealing with children or cultures. As compared with baseline recommenders, the AHCF depends on more data hence performs significantly better on 1M datasets as compared to the 100K dataset. The AHCF performs well with a large dataset, which is desirable for real-world implementations which usually have increasing data sizes and users.

6.5 Recommendations

Further research can be carried out to produce a model that is 100% online. The spark ML streaming algorithms need to be extended to accommodate the KMeans streaming for a much-structured presentation, and there is the need to take into account the semantic meaning of the textual data employed in the LDA clustering. Privacy-related issues, such as the appropriate content for specific ages, are not accounted for by the implemented system (demographic data).

REFERENCES

[1] I. Portugal, P. Alencar, and D. Cowan, "The use of machine learning algorithms in recommender systems: A systematic review," *Expert Systems with Applications*. 2018.

[2] A. Ansari, Y. Li, and J. Z. Zhang, "Probabilistic Topic Model for Hybrid Recommender Systems: A Stochastic Variational Bayesian Approach," 2017.

[3] O. Sar Shalom, N. Koenigstein, U. Paquet, and H. P. Vanchinathan, "Beyond Collaborative Filtering," in *Proceedings of the 25th International Conference on World Wide Web - WWW '16*, 2016, pp. 63–72.

[4] S. Panigrahi, R. K. Lenka, and A. Stitipragyan, "A Hybrid Distributed Collaborative Filtering Recommender Engine Using Apache Spark," *Procedia Comput. Sci.*, vol. 83, no. BigD2M, pp. 1000–1006, 2016.

[5] R. K. Lenka, R. K. Barik, S. Panigrahi, and S. S. Panda, "An improved hybrid distributed collaborative model for filtering recommender engine using Apache Spark," *Int. J. Intell. Syst. Appl.*, vol. 10, no. 7, 2018.

[6] C. A. Gomez-Uribe and N. Hunt, "The Netflix Recommender System_Algorithms, Business Value.pdf," *ACM Trans. Manag. Inf. Syst.*, vol. 6, no. 4, 2015.

[7] V. Ranjith, V. Rajaram, and P. Identification, "Mining User ' S Interest By Mining Interpersonal Interest Similarity And Polarity Identification," vol. 5, no. 3, pp. 115–119, 2018.

[8] A. Ansari, Y. Li, and J. Zhang, *Probabilistic Topic Model for Hybrid Recommender Systems: A Stochastic Variational Bayesian Approach*, vol. 37. 2018.

[9] N. Silva, D. Carvalho, A. C. M. Pereira, and F. Mourão, "The Pure Cold-Start Problem : A deep study about how to conquer first-time users in recommendations domains," vol. 80,

pp. 1–12, 2019.

[10] J. Lu, D. Wu, M. Mao, W. Wang, and G. Zhang, “Recommender system application developments: A survey,” *Decis. Support Syst.*, vol. 74, pp. 12–32, 2015.

[11] M. Shvarts, M. Lobur, and Y. Stekh, “Some Trends in Modern Recommender Systems,” *2017 XIIIth Int. Conf. Perspect. Technol. Methods MEMS Des.*, pp. 167–169, 2017.

[12] W. Xing, W. Zhou, and W. Wang, “A hybrid collaborative filtering algorithm on spark,” *ICIC Express Lett. Part B Appl.*, vol. 7, no. 11, pp. 2397–2403, 2016.

[13] L. Esperanza, “Recommendation System for Netflix,” 2018.

[14] S. Chang *et al.*, “Streaming Recommender Systems,” pp. 381–389, 2016.

[15] J. Wei, J. He, K. Chen, Y. Zhou, and Z. Tang, “Collaborative filtering and deep learning-based recommendation system for cold start items,” vol. 69, pp. 29–39, 2017.

[16] P. Shinde, “A Survey on Multimedia Information Retrieval System,” vol. 5, no. 6, pp. 1945–1949, 2014.

[17] A. M. Yagci, T. Aytekin, and F. S. Gurgen, “Scalable and adaptive collaborative filtering by mining frequent item co-occurrences in a user feedback stream,” *Eng. Appl. Artif. Intell.*, vol. 58, pp. 171–184, Feb. 2017.

[18] J. Vinagre, A. M. Jorge, and J. Gama, “Fast Incremental Matrix Factorization for Recommendation with Positive-Only Feedback,” 2014, pp. 459–470.

[19] Y. Song, W. Ji, and S. Liu, “Research on personalized hybrid recommendation system,” in *IEEE CITS 2017 - 2017 International Conference on Computer, Information and Telecommunication Systems*, 2017, pp. 133–137.

[20] K. Haruna, M. A. Ismail, D. Damiasih, J. Sutopo, and T. Herawan, “A collaborative approach for research paper recommender system,” *PLoS One*, vol. 12, no. 10, pp. 1–17,

2017.

[21] J. Büschken and G. Allenby, *Sentence-Based Text Analysis for Customer Reviews*, vol. 35. 2016.

[22] P. Chu and S. Lee, "A Novel Recommender System for E-Commerce," pp. 1–5, 2017.

[23] A. Piepenbrink and E. Nurmammadov, "Topics in the literature of transition economies and emerging markets," *Scientometrics*, 2015.

[24] A. Piepenbrink and A. S. Gaur, "Topic Models As a Novel Approach To Identify Themes in Content Analysis: the Example of Organizational Research Methods," *Acad. Manag. Proc.*, 2017.

[25] S. Kaplan and K. Vakili, "The double-edged sword of recombination in breakthrough innovation," *Strateg. Manag. J.*, 2015.

[26] S. Tirunillai and G. Tellis, "Mining Marketing Meaning from Online Chatter," *J. Mark. Res.*, 2014.

[27] S. Moro, P. Cortez, and P. Rita, "Literature Review-Business intelligence in banking: A literature analysis from 2002 to 2013 using text mining and latent Dirichlet allocation," *Expert Syst. Appl.*, 2015.

[28] G. S. Miller, "Discussion of 'the evolution of 10-K textual disclosure: Evidence from Latent Dirichlet Allocation,'" *Journal of Accounting and Economics*. 2017.

[29] M. Dowling, A. Piepenbrink, S. Aziz, and H. Hammami, "Machine learning in finance : A topic modeling approach Machine learning in finance : A topic modeling approach," no. February, 2019.

[30] T. Rajasundari, S. Palaniappan, and P. Kumar, *Performance analysis of topic modeling algorithms for news articles*, vol. 2017. 2017.

[31] G. Brookes and T. McEnery, "The utility of topic modelling for discourse studies: A critical evaluation," *Discourse Stud.*, p. 146144561881403, 2018.

[32] A. Törnberg, "Combining CDA and topic modeling : Analyzing discursive connections between Islamophobia and anti-feminism on an online forum Combining CDA and topic modeling : Analyzing discursive connections between Islamophobia and anti-feminism on an online forum," no. November, 2017.

[33] T. Mcenery, M. McGlashan, and R. Love, "Press and social media reaction to ideologically inspired murder : The case of Lee Rigby," 2015.

[34] W. Zhao *et al.*, "A heuristic approach to determine an appropriate number of topics in topic modeling," *BMC Bioinformatics*, vol. 16, no. Suppl 13, p. S8, 2015.

[35] D. Sisiaridis and O. Markowitch, "Feature Extraction And Feature Selection : Reducing Data Complexity With Apache," vol. 9, no. 6, pp. 39–51, 2017.

[36] N. Ranjbar Kermany and S. H. Alizadeh, "A hybrid multi-criteria recommender system using ontology and neuro-fuzzy techniques," *Electron. Commer. Res. Appl.*, 2017.

[37] B. Zhang and B. Yuan, "Improved collaborative filtering recommendation algorithm of the similarity measure," *AIP Conf. Proc.*, vol. 1839, 2017.

[38] Köhler Victor, "ALS Implicit Collaborative Filtering – Rn Engineering – Medium." [Online]. Available: https://medium.com/radon-dev/als-implicit-collaborative-filtering-5ed653ba39fe. [Accessed: 20-Nov-2018].

[39] A. Pal, P. Parhi, and M. Aggarwal, "An improved content-based collaborative filtering algorithm for movie recommendations," in *2017 10th International Conference on Contemporary Computing, IC3 2017*, 2018, vol. 2018-Janua, no. August, pp. 1–3.

[40] M. Nkwo, R. Orji, J. Nwokeji, and C. Ndulue, "E-commerce personalization in Africa: A

comparative analysis of Jumia and Konga," *CEUR Workshop Proc.*, vol. 2089, pp. 68–76, 2018.

[41] "How Netflix's Recommendations System Works." [Online]. Available: https://help.netflix.com/en/node/100639. [Accessed: 18-Jul-2019].

[42] F.Maxwell and J. A. Konstan, "The MovieLens Datasets: History and Context," *ACM Trans. Intell. Syst. Technol.*, 2015.

[43] "MovieLens | GroupLens." [Online]. Available: https://grouplens.org/datasets/movielens/. [Accessed: 19-Feb-2019].

[44] F. M. Harper and J. A. Konstan, "The MovieLens Datasets," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, pp. 1–19, Dec. 2015.

[45] G. Zhang and X. Zhou, "Similarity-Based Matrix Factorization for Recommender Systems," in *Proceedings - 2017 10th International Symposium on Computational Intelligence and Design, ISCID 2017*, 2018, vol. 2018-Janua, pp. 7–11.

[46] R. Zadeh, "Distributed Algorithms and Optimization - Matrix Completion via Alternating Least Square (ALS)." 2015.

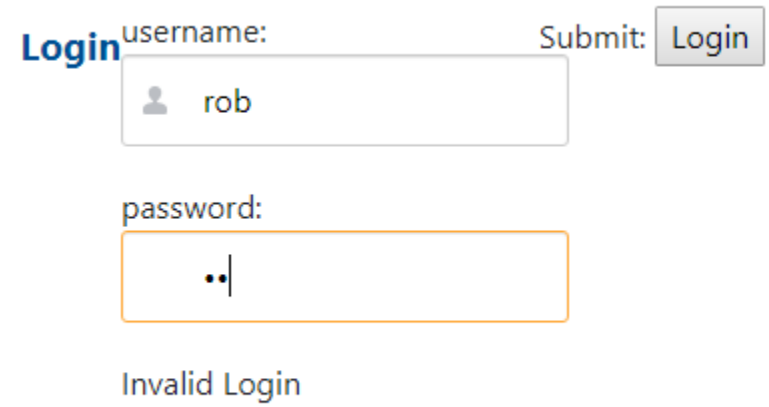
[47] R. Ghayad, M. Cragg, and F. Pinter, *Elections and the Economy: What to do about Recessions?* vol. 13. 2016.

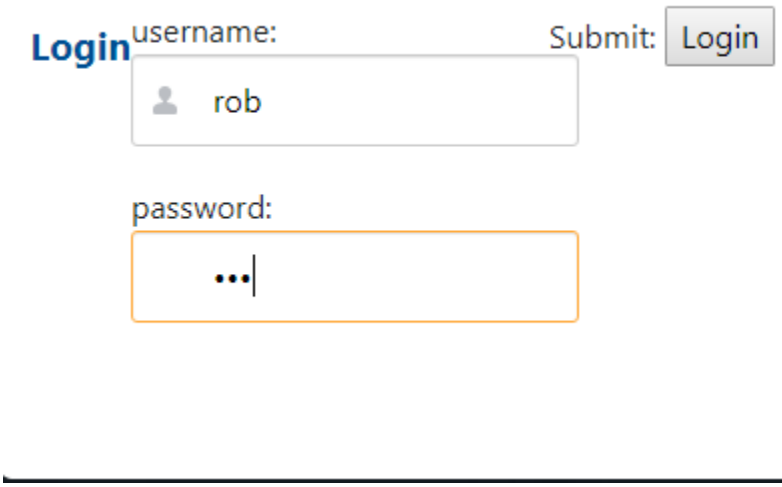
[48] S. Panigrahi, R. K. Lenka, and A. Stitipragyan, "A Hybrid Distributed Collaborative Filtering Recommender Engine Using Apache Spark," *Procedia Comput. Sci.*, vol. 83, no. BigD2M, pp. 1000–1006, 2016.

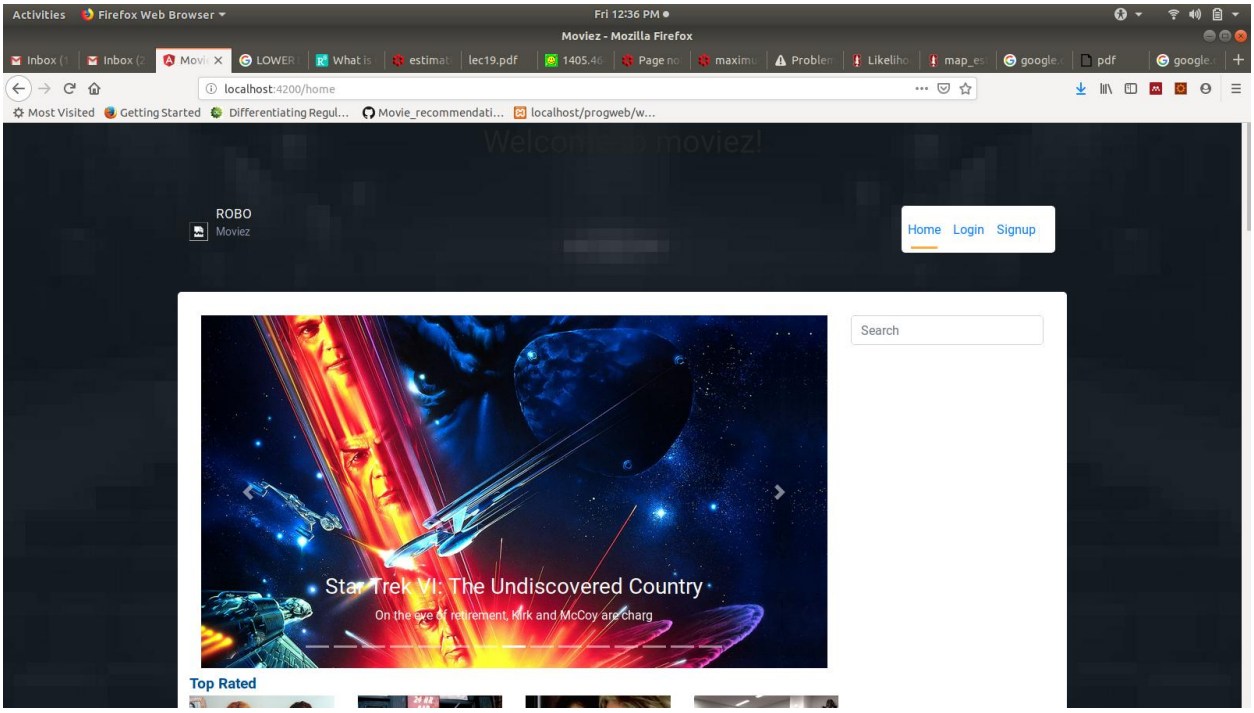
[49] "Datasets | Kaggle." [Online]. Available: https://www.kaggle.com/datasets. [Accessed: 10-Jul-2019].

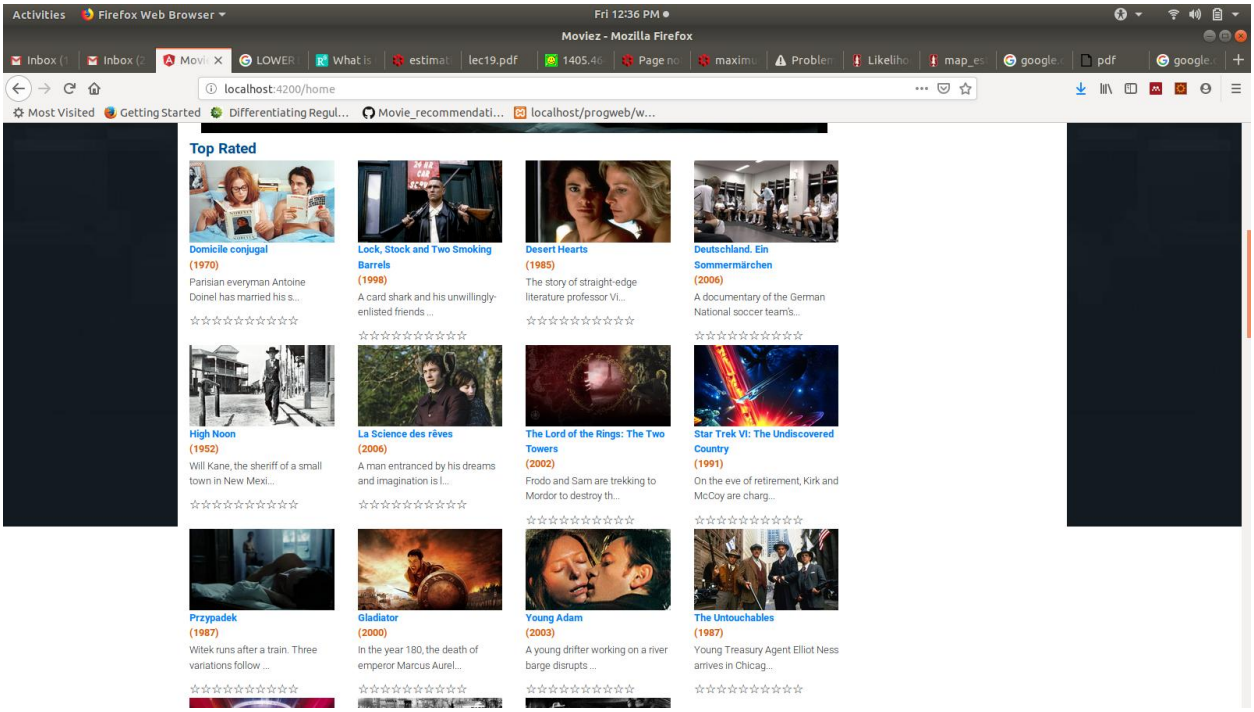
APPENDIX A

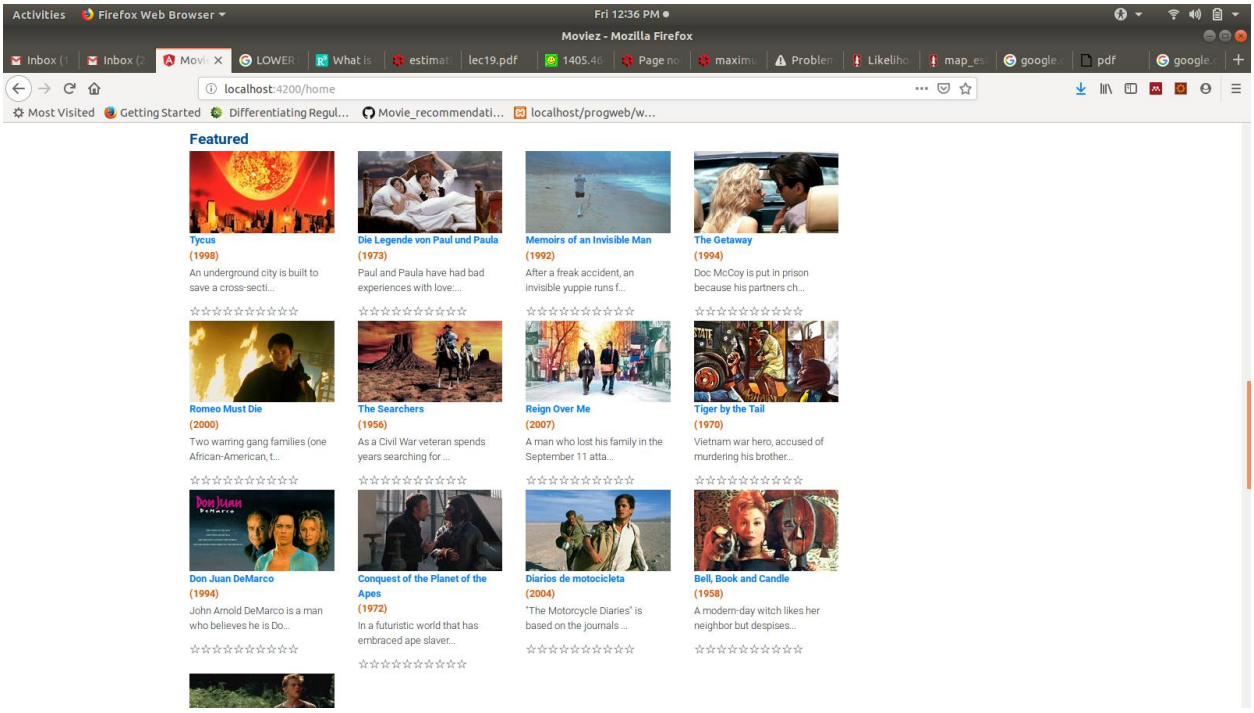
Web Portal Interfaces

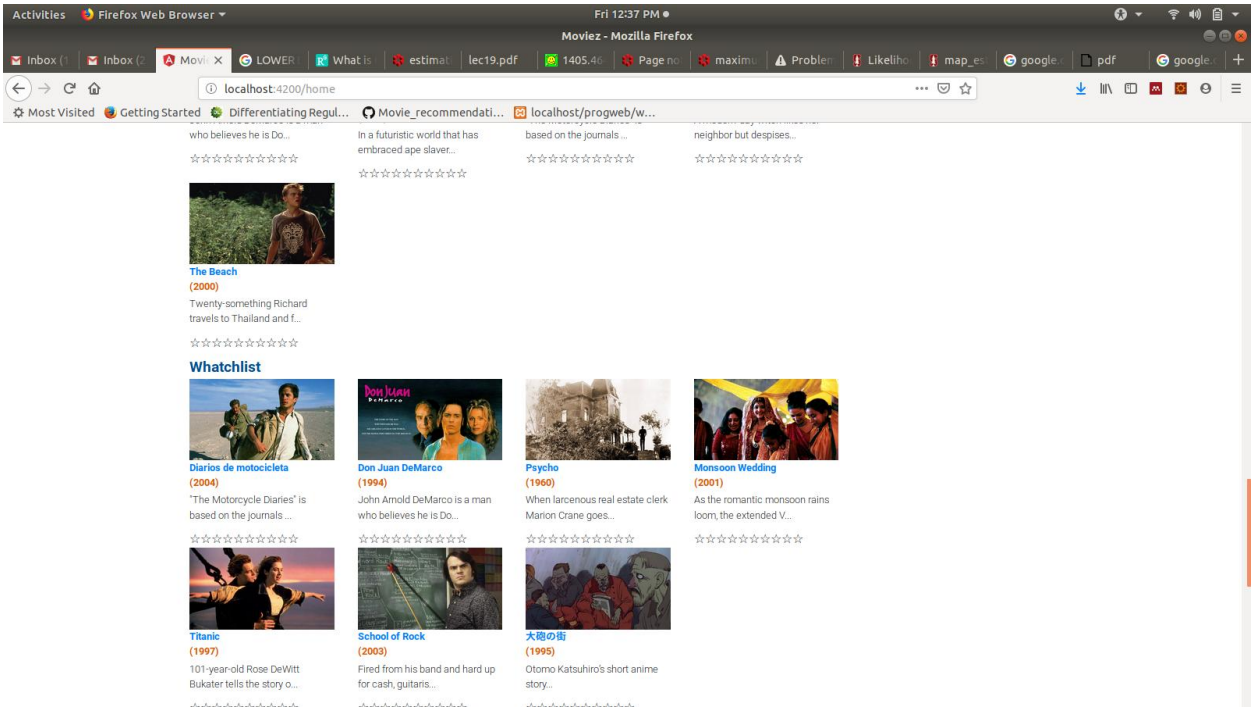


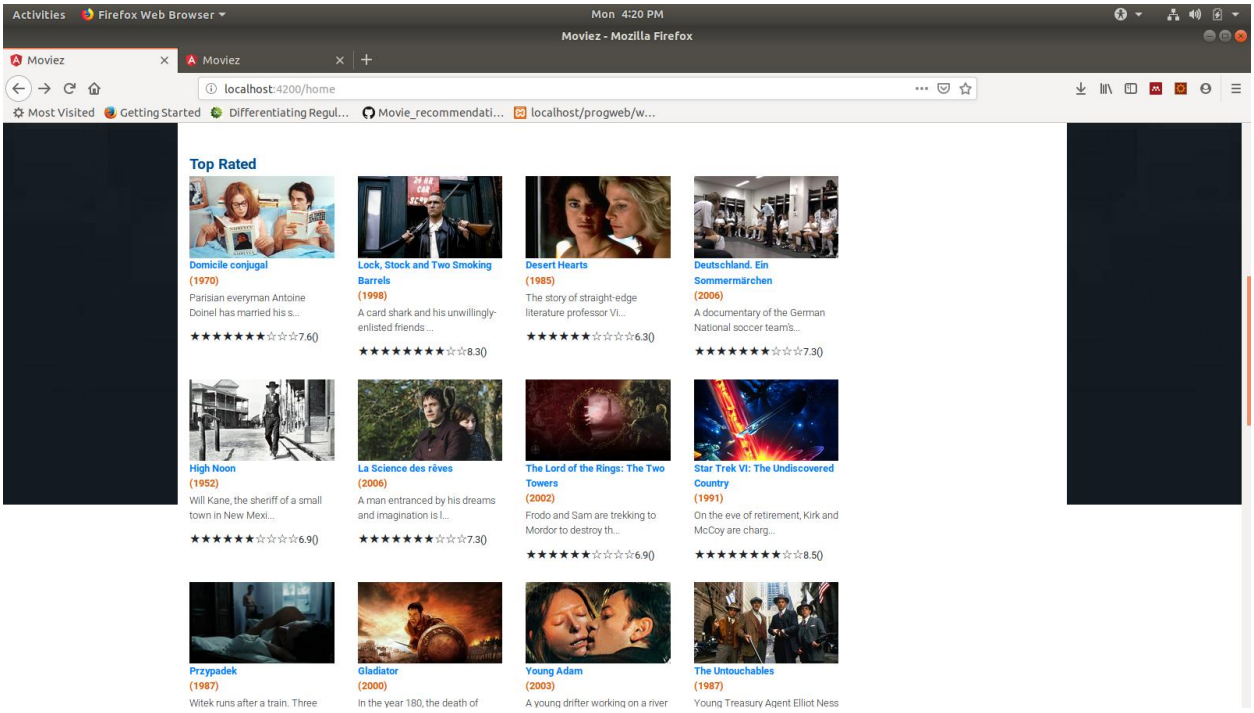


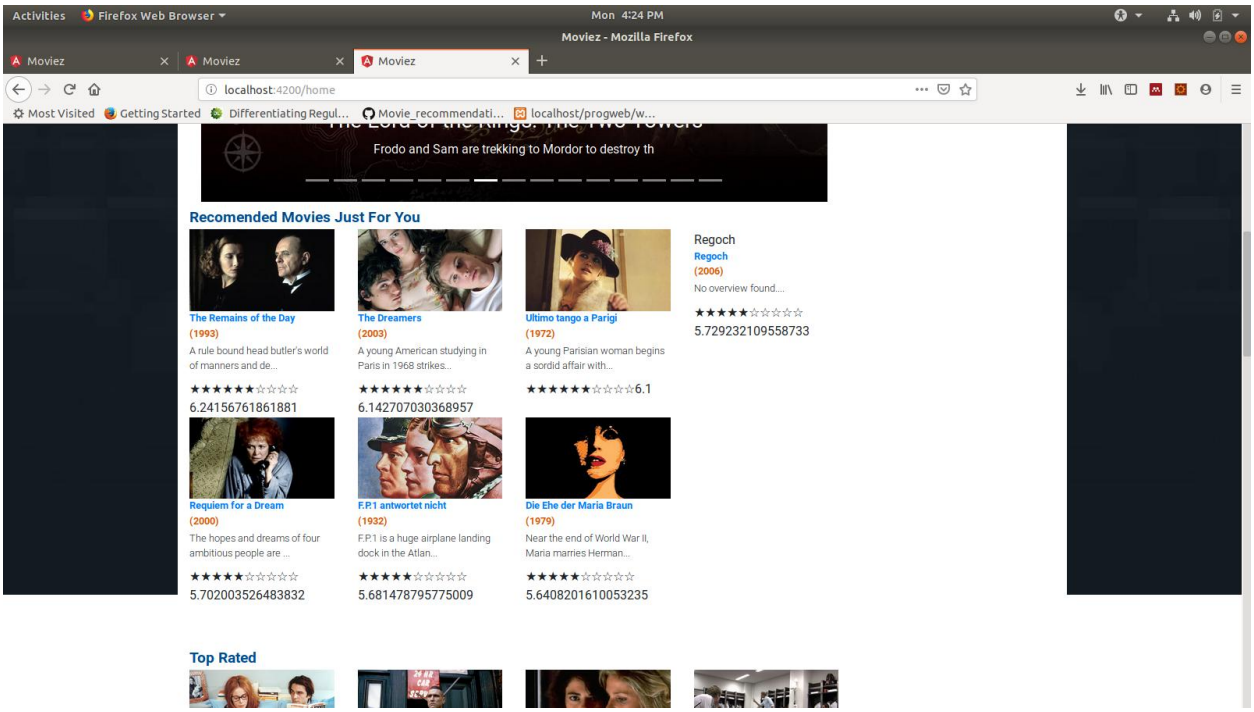


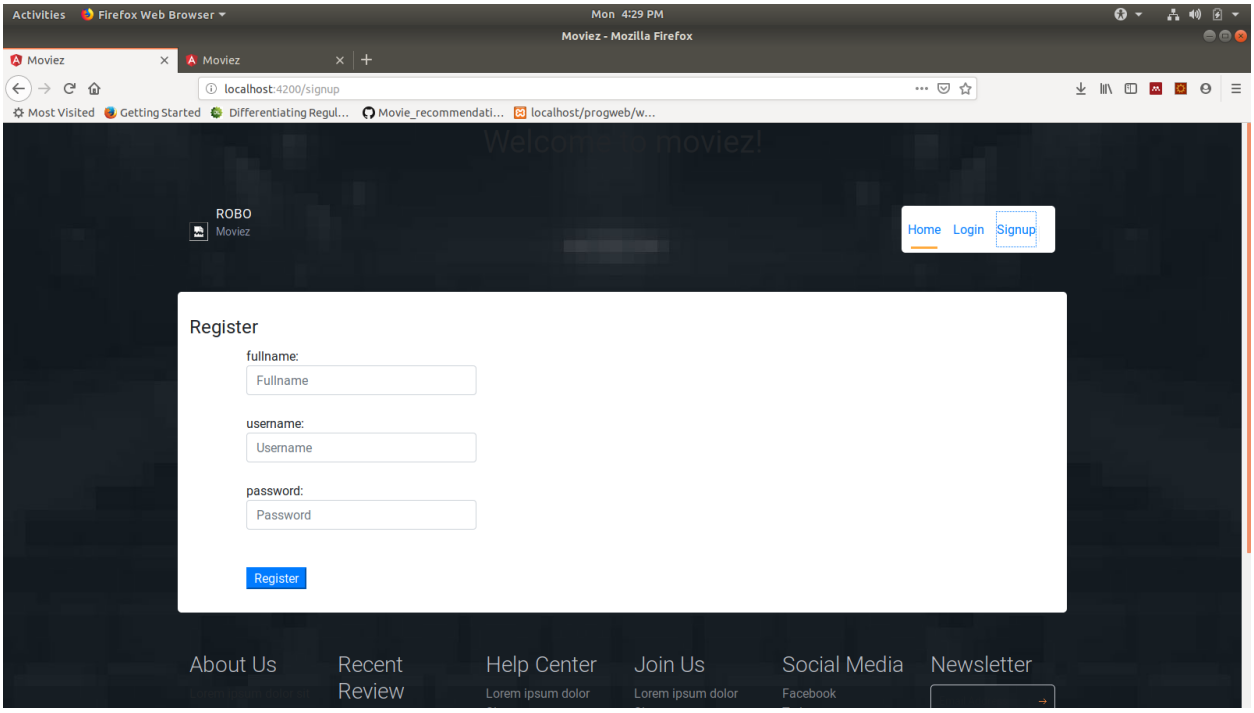












APPENDIX B

Offline Algorithm

ALS

```
package thesis
import java.util.Properties
import Main.{prop, scaledDatam, url}
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.clustering.KMeans
import org.apache.spark.ml.evaluation.ClusteringEvaluator
import org.apache.spark.ml.feature.{Normalizer, StandardScaler,
VectorAssembler}
import org.apache.spark.ml.recommendation.ALS
import org.apache.spark.sql.SparkSession
import org.apache.spark.ml.evaluation.RegressionEvaluator
object scalable2{
//------------------------------DIMENSIONALITY REDUCTION (ALS)-
  //Input: Rating File (rating.csv) //[UserId, MovieId, Rating]  //Output:
UserFeature <UserId, FeatureVector> //ProductFeature<MovieId, FeatureVector>
//Begin://On each worker node do in //parallel:
    //1. Load the data from rating.csv file for all movieLens Dataset  //data
m load(rating.csv) // $example on$
  case class Rating(userId: Int, movieId: Int, rating: Float, timestamp: Long)
  def parseRating(str: String): Rating = {
    val fields = str.split("::")
    assert(fields.size == 4)
    Rating(fields(0).toInt, fields(1).toInt, fields(2).toFloat,
fields(3).toLong)
  }  // $example off
  def main(args: Array[String]) {
    val spark = SparkSession
      .builder()
      .master("local[*]")
      .appName("ALS Algo1")
      .getOrCreate()
    import spark.implicits._
    // ml-1m dataset
    val df1 = spark.read.textFile("/home/arrandale/Desktop/thesis/ml-
1m/ratings.dat")
      .map(parseRating)
      .toDF()
```

```
  println("Results of ratings.dat file with descriptions:")
  df1.describe().show()
 df1.printSchema()
  df1.show()
// movielens-20m dataset
val df20= spark.read
  .option("inferSchema","true")
  .option("header", "true")
  .format("csv")
  .load("/home/arrandale/Desktop/thesis/ml-20m/ratings.csv")
println("Results of csv load file with descriptions:")
df20.describe().show()
df20.printSchema()
df20.show()
println("movies.csv")
val mov = spark.read
  .option("inferSchema","true")
  .option("header", "true")
  .format("csv")
  .load("/home/arrandale/Desktop/thesis/ml-20m/movies.csv")
//2. Training and testing the data
val Array(training, test) = df1.randomSplit(Array(0.8, 0.2))
   println("Training data results")
training.show()
training.describe().show()
training.printSchema()
println("Test data results")
test.show()
test.describe().show()
test.printSchema()
// 3. Build the recommendation model using ALS on the training data
val als = new ALS()
  .setMaxIter(5)
  .setRegParam(0.01)
  .setImplicitPrefs(true)
  .setUserCol("userId")
  .setItemCol("movieId")
  .setRatingCol("rating")
val model = als.fit(training)
// Evaluate the model by computing the RMSE on the test data
// Note we set cold start strategy to 'drop' to ensure we don't get NaN
evaluation metrics
model.setColdStartStrategy("drop")
val predictions = model.transform(test)
```

```
    val evaluator = new RegressionEvaluator()
      .setMetricName("rmse")
      .setLabelCol("rating")
      .setPredictionCol("prediction")
    val rmse = evaluator.evaluate(predictions)
    println(s"Root-mean-square error = $rmse")
    // Saving the model
    model.write.overwrite().save("/home/arrandale/models/sparkml.model")
    // Generate top 10 movie recommendations for each user
    val userRecs = model.recommendForAllUsers(10)
    // Generate top 10 user recommendations for each movie
    val movieRecs = model.recommendForAllItems(10)
    //val topRecsForUser = model.recommendForUserSubset(2, 6);
//.............................END OF ALS ALGORITHM.............
```

KMeans

```
//4.for i = 1 to n, n = No.of iterations// 5.for j = 1 to k,k = No. of
clusters // 6. cluster = kmeans.train(UserFeatureVector) end for end for
//for each UserFeature:
    val kData=scaledData.select("userId", "features")
    // Trains a k-means model.
    val kmeans = new KMeans().setK(5).setSeed(1L)
    val modelk = kmeans.fit(kData)
    // Make predictions 8. clusterIdmmodel.predict(UserFeature)
    val predictionsk = modelk.transform(kData)
    // Evaluate clustering by computing Silhouette score
    val evaluatork = new ClusteringEvaluator()
    // Shows the result. 9. clusterCentermmodel.clusterCenters(clusterId)
   predictionsk.collect().foreach(println)
    println("Cluster IDs for users for each type: ")
    predictionsk.printSchema()
    predictionsk.show(121)
    println("Cluster IDs for users show type: ")
    println("Cluster Centers: ")
    modelk.clusterCenters.foreach(println)
    modelk.clusterCenters.foreach(println)
```

LDA

```
//LDA
import org.apache.spark.{SparkConf, SparkContext}
object LDA {
  def main(args: Array[String]): Unit = {
        //.............NEW USER MODEL.....................
//1.Load & parse dataset with all available text/word //Creates a SparkSession
```

```
      val startTimeMillis = System.currentTimeMillis()
    val spark = SparkSession
      .builder.master("local[4]")
      .appName(s"${this.getClass.getSimpleName}")
      .getOrCreate()
    println("movies.csv")
    val ldaDataset = spark.read
      .option("inferSchema","true")
      .option("header", "true")
      .format("csv")
      .load("D:\\ml\\als\\data\\ml-20m\\movies.csv")
    println("genome-tags.csv")
    val ldaDataset2 = spark.read
      .option("inferSchema","true")
      //.option("delimiter",";")
      .option("header", "true")
      .format("csv")
      .load("D:\\ml\\als\\data\\ml-20m\\genome-tags.csv")
  println("tags.csv")
    val ldaDataset3 = spark.read
      .option("inferSchema","true")
      .option("header", "true")
      .format("csv")
      .load("D:\\ml\\als\\data\\ml-20m\\tags.csv")
      .withColumnRenamed("movieId","tagmovieId")
      .withColumnRenamed("tag","tag2")
     //2. MERGE ALL FILES WITH TEXT INFORMATION
    println("joinPowers222")
    val arr2= ldaDataset.join(ldaDataset3, ldaDataset.col("movieId") ===
ldaDataset3.col("tagmovieId
    println("joinPowers")
    val arr= ldaDataset.join(ldaDataset3, ldaDataset.col("movieId") ===
ldaDataset3.col("tagmovieId"))
      .drop("tagmovieId","timestamp")
    println("finalJoinPowers")
    val arr1= arr.join(ldaDataset2, arr.col("tag2") ===
ldaDataset2.col("tag"))
//.....................................CONVERT TO TEXT FIELDS TO ARRAY
    var exData=
arr1.select(arr1.col("userId"),arr1.col("movieId"),arr1.col("title"),arr1.col(
"genres"),arr1.col("tag"))
        import org.apache.spark.sql.functions._
    val inp_array = exData.withColumn("arrayColumn", array("title", "genres",
"tag"))
```

```
    // USE COUNT VECTORIZER TO INDEX AND CONVERT TO INDICES FOR LDA
   import org.apache.spark.ml.feature.{CountVectorizer, CountVectorizerModel}
   // fit a CountVectorizerModel from the corpus
   import java.util._
   val prop = new Properties()
   prop.setProperty("driver", "com.mysql.cj.jdbc.Driver")
   prop.setProperty("user", "rob")
  prop.setProperty("password", "rob")
   val url = "jdbc:mysql://localhost:3306/rob"
   val rank=Array(5)
   for(i<-rank ) {
     val cvModel: CountVectorizerModel = new CountVectorizer()
       .setInputCol("arrayColumn")
       .setOutputCol("features")
       .setVocabSize(i)
       .setMinDF(2)
       .fit(inp_array)
     val cvgen = cvModel.transform(inp_array) //.show(false)
     println("CVGEN COUNT VECTORIZER RESULT")
     println("COUNT VECTORIZERRESULT  ")
     println("CVGEN COUNT VECTORIZOR")
     println("COUNT VECTORIZOR(CVGEN) ")
   //.................LDA CLUSTERING MODEL.........................
     import org.apache.spark.ml.clustering.LDA
     val lda = new
LDA().setK(i).setMaxIter(10).setFeaturesCol("features").setSeed(1L)
     val model = lda.fit(cvgen)
     println("learned topics")
     val ll = model.logLikelihood(cvgen)
     val lp = model.logPerplexity(cvgen)
     println(s"The lower bound on the log likelihood of the entire corpus:
$ll")
     // Shows the result.
     val transformed = model.transform(cvgen)
     println("LDA RESULT")
     println("LDA FINAL RESULT  ")
     println(s"The upper bound on perplexity: $lp")
     // Describe topics.
println("==========vocubolary stuff=======")
     val vocab = cvModel.vocabulary
     val wordsWithWeights = udf((x: mutable.WrappedArray[Int],
                                 y: mutable.WrappedArray[Double]) => {
       x.map(i => vocab(i)).zip(y)      }
     )  val topics2 = model.describeTopics(i)
```

```
        .withColumn("topicWords",
          wordsWithWeights(col("termIndices"), col("termWeights")) )
      val topics2exploded = topics2
        .select("topic", "topicWords")
        .withColumn("topicWords", explode(col("topicWords")))
       val finalTopic = topics2exploded
        .select(       col("topic"),
          col("topicWords").getField("_1").as("word"),
          col("topicWords").getField("_2").as("weight")
       )
      finalTopic.show()
      println("================voabulary stuff==========================")
      //.................neededresults.....................
  val seeJoin = finalTopic.crossJoin(transformed)//
println(seeJoin.explain())
     val crossJ = seeJoin.drop("arrayColumn", "features", "topicDistribution",
"weight").distinct()
      println(s"joining and joining and joining   $i")
      crossJ.dropDuplicates("movieId").show(100)
      println(s"joining and joining and joining end $i")
     val endTimeMillis = System.currentTimeMillis()
      val durationSeconds = (endTimeMillis - startTimeMillis) / 1000
      println("COMPUTATIONAL TIME LDA"+durationSeconds)
      println("Saving to database")
      import spark.implicits._
      val topics = model.describeTopics(5)
      for (x <- 0 to topics.count().toInt) {
        val cluster= crossJ.filter($"topic"===""+x)
        cluster.write.mode("append").jdbc(url, "ldaClusters", prop) }   }
```

Adaptive Algorithms

KMeans Streaming

```
import java.util
import akka.actor.ActorSystem
import akka.http.scaladsl.Http
import akka.stream.ActorMaterializer
import akka.Done
import akka.http.scaladsl.server.Route
import akka.http.scaladsl.server.Directives._
import akka.http.scaladsl.model.StatusCodes
import org.apache.spark.mllib.clustering.KMeansModel
import org.apache.spark.mllib.recommendation.{MatrixFactorizationModel,
Rating}
```

```
import akka.http.scaladsl.marshallers.sprayjson.SprayJsonSupport._
import spray.json.DefaultJsonProtocol._
import scala.io.StdIn
import scala.concurrent.Future
class HttpServer  {
  import HttpServer._
  def setModelK(m:KMeansModel): Unit ={
    modelK=m}
  def getModelK():KMeansModel={
    modelK}
  def setCount(i:Int): Unit =
  {    count=i  }
  def getCount():Int={
    count  }}
object HttpServer {
  // needed to run the route  // needed to run the route
  implicit val system = ActorSystem()
  implicit val materializer = ActorMaterializer()
  //needed 4 future map/flatmap in the end &future in fetchItem and saveOrder
  implicit val executionContext = system.dispatcher
  var orders: List[Item] = Nil
  var temp_ratings: List[Rating] = Nil // domain model
  final case class Item(name: String, id: Long)
  final case class Order(items: List[Item])
  final case class UserChoice(userId: Int, movieId: Int)
  final case class Movie(movieId: Int, rating: Int)
  final case class MovieRating(ratings:List[Rating])
  implicit val itemFormat = jsonFormat2(Item)
  implicit val orderFormat = jsonFormat1(Order)
  implicit val movieFormat = jsonFormat2(Movie)
  implicit val ratingFormat = jsonFormat3(Rating)
  implicit val movieRatingFormat = jsonFormat1(MovieRating)
  var model:MatrixFactorizationModel = MllibTrain.getALSModel
  var modelK:KMeansModel=null
  var count:Int=0
  def fetchItem(itemId: Long): Future[Option[Item]] = Future {
    orders.find(o => o.id == itemId)}
  def saveOrder(order: Order): Future[Done] = {
    orders = order match {
      case Order(items) => items ::: orders
      case _            => orders  }
    Future { Done }  }
  def updateClusters(id:Int): Future[Done] = {
    println("Centers^^")
```

```
    val httpServer = new HttpServer
    println(httpServer.getModelK().clusterCenters.length)
    MllibTrain.saveMovieClustersToDb(httpServer.getModelK())
    Future { Done }  }
  def rateMovie(rating: Rating): Future[Done] = {
    temp_ratings = temp_ratings :+ rating
    val msg =
rating.user.toString+","+rating.product.toString+","+rating.rating.toString+"\
n"
    TCPLint.sendMessage(msg)
    println("Rating for ")
    print(rating)
    Future { evaluateAndRecommend() }
    Future { Done }  }
  def evaluateAndRecommend(): Unit ={
    if(temp_ratings.size>=5){
      println("retraining Model...")
      MllibTrain.trainAndSaveModel()
      model = MllibTrain.getALSModel
      println("Completed retraining Model")
      temp_ratings = Nil;    }  }
  var movies: List[Rating] = Seq(Rating(1,201,5)).toList
  def recommendMovieToUser(userId: Int): Future[Option[List[Rating]]] = Future
{
    val recommendation:Array[Rating] = model.recommendProducts(userId,10)
    movies = recommendation.toList
    movies.find(o => o.user==userId)
    Option(movies)}
  def recommendMovieToAnyUser(userId: Int,movieId: Int):
Future[Option[List[Rating]]] = Future {
    val recommendation:Array[Rating] = model.recommendProducts(userId,10)
    movies = recommendation.toList
    movies.find(o => o.user==userId)
    Option(movies) }
  def main(args: Array[String]) {
    TCPLint.setUpConnection()
    MllibTrain.trainAndSaveModel()
    model = MllibTrain.getALSModel
    val route: Route =
      get {
        pathPrefix("item" / LongNumber) { id =>
          // there might be no item for a given id
          val maybeItem: Future[Option[Item]] = fetchItem(id)
          onSuccess(maybeItem) {
```

```
            case Some(item) => complete(item)
            case None       => complete(StatusCodes.NotFound)       }    }  } ~
        get {
          pathPrefix("recommend" / IntNumber) { (id) =>
            // there might be no item for a given id
            val maybeItem: Future[Option[List[Rating]]] =
recommendMovieToUser(id)
            onSuccess(maybeItem) {
              case Some(a) => complete(a)
              case None       => complete(StatusCodes.NotFound)         }
}     } ~
    get {
      pathPrefix("update_clusters" / IntNumber) { (id) =>
        // there might be no item for a given id
        val maybeItem: Future[Done] = updateClusters(id)
        onComplete(maybeItem) { done =>
          complete("update7d")        }      }    } ~
        get {
          pathPrefix("recommend_for_any" / IntNumber/IntNumber) { (id,mId) =>
            // there might be no item for a given id
            val maybeItem: Future[Option[List[Rating]]] =
recommendMovieToAnyUser(id,mId)
            onSuccess(maybeItem) {
              case Some(a) => complete(a)
              case None       => complete(StatusCodes.NotFound)  }          }
} ~
        post {
          path("rate") {
            entity(as[Rating]) { rating =>
              val saved: Future[Done] = rateMovie(rating)
              onComplete(saved) { done =>
                complete("rating submitted")          }  }   }        }~
        post {
          path("create-order") {
            entity(as[Order]) { order =>
              val saved: Future[Done] = saveOrder(order)
              onComplete(saved) { done =>
                complete("order created")  }           }          }        }
    Future{Stream2.startStream()}
    val bindingFuture = Http().bindAndHandle(route, "localhost", 8080)
    println(s"Server online at http://localhost:8080/\nPress RETURN to
stop...")
    StdIn.readLine() // let it run until user presses return
    bindingFuture
```

```
.flatMap(_.unbind()) // trigger unbinding from the port
.onComplete(_ ⇒ system.terminate()) // and shutdown when done  }}
```

APPENDIX C

AHCF memory footprint results

ID	RDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory
8	ratingBlocks	Memory Deserialized 1x Replicated	1	100%	1174.1 KB
11	userInBlocks	Memory Deserialized 1x Replicated	4	100%	789.3 KB
12	userOutBlocks	Memory Deserialized 1x Replicated	4	100%	15.4 KB
16	itemInBlocks	Memory Deserialized 1x Replicated	4	100%	795.1 KB
17	itemOutBlocks	Memory Deserialized 1x Replicated	4	100%	23.3 KB
210	users	Memory Deserialized 1x Replicated	4	100%	129.0 KB
211	products	Memory Deserialized 1x Replicated	4	100%	219.1 KB