**Resolving Sparsity in Recommendation Systems using SGD**

Project Report

## Submitted

*In partial fulfillment of the requirements for the award of the degree*

## BACHELOR OF TECHNOLOGY

**In**

## COMPUTER SCIENCE and ENGINEERING

### By

V Sasidhar Reddy (191FA04547)

V Vishnu Veera (191FA04551)

Under the Guidance of

## Ms. Pushya Chaparala,

## Assistant Professor



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

# VIGNAN’S FOUNDATION FOR SCIENCE AND TECHNOLOGY RESEARCH

(Deemed to be University)

**Vadlamudi, Guntur -522213, INDIA.**

# May, 2023



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

***CERTIFICATE***

This is to certify that the report entitled **“Resolving Sparsity in Recommendation Systems using SGD**” is submitted by **“V Sasidhar Reddy (191FA04547), V Vishnu Veera (191FA04551)”** in the partial fulfillment of course work of an project, carried out in the department of CSE, VFSTR Deemed to be University.

Ms. Pushya Chaparala External Examiner Dr. K.V. Krishna Kishore

Assistant Professor HOD, CSE

Dean, School of Comp. & Info.

## DECLARATION

I hereby declare that the project entitled “**Resolving Sparsity in Recommendation Systems using SGD**” was submitted for the “**Department of Computer Science and Engineering”**. This dissertation is our original work, and the project has not formed the basis for the award of any degree, associateship and fellowship or any other similar titles and no part of it has been published or sent for publication at the time of submission.

By

V Sasidhar Reddy (191FA04547)

V Vishnu Veera (191FA04551)

Date:27/05/2023

## ACKNOWLEDGEMENT

This project program is a golden opportunity for learning and self-development. Consider ourselves very lucky and honored to have so many wonderful people lead us through the completion of this project.

I express our gratitude to the Project Guide **Ms. Pushya Chaparala,** for permitting us to undertake the Program and for the help and cooperation throughout the course of our project Program.

I express our gratitude to the Project Coordinator, **Mr. D Yakobu**, for permitting us to undertake the Program and for the help and cooperation throughout the course of our project Program.

It is a great pleasure for us to express our sincere thanks to **Dr K. V. Krishna Kishore, HOD, CSE and Dean, School of Computing & Informatics** of VFSTR, deemed to be a university**,** for providing us with an opportunity to do our Project Program.

We extend our wholehearted gratitude to all our faculty members, programmers, and technicians of the Department of Computer Science and Engineering who helped use our academics throughout the course.

Finally, we wish to express thanks to our family members for the love and affection overseas and cheerful depositions, which are vital for sustaining the effort required for completing this work.

With Sincere regards,

V Sasidhar Reddy(191FA04547)

V Vishnu Vinayak (191FA04551)

# ABSTRACT

# Sparsity in recommendation systems poses a significant challenge in accurately predicting user preferences and providing personalized recommendations. This paper focuses on resolving the issue of sparsity in recommendation systems using Stochastic Gradient Descent (SGD). The goal is to leverage the available user-item interaction data effectively and make accurate predictions for missing or unobserved user-item pairs. The proposed methodology involves employing matrix factorization techniques and optimizing the model parameters using SGD. By decomposing the user-item interaction matrix into lower-dimensional latent factors, the system can capture the underlying patterns and relationships in the data. SGD is utilized to iteratively update the model parameters based on sampled user-item pairs, effectively handling the sparsity in the dataset. Evaluation of the model will be performed using various metrics such as Mean Squared Error (MSE), Precision, Recall, and F1-score. The experimental results will demonstrate the effectiveness of the proposed approach in addressing sparsity and improving the performance of recommendation systems.

# Key Words: Recommendation systems, Matrix Factorization, SGD, Sparsity

# CONTENTS

#### *Declaration ii*

#### *Acknowledgement iii*

#### *Abstract iv*

*List of Figures vii*

*List of Graph viii*

**CHAPTER-1: INTRODUCTION**

1.1 Introduction of project **1**

1.2 Literature Survey **1**

1.3 Motivation  **2**

1.4 Problem Statement  **3**

1.5 Objective  **4**

1.5.1 General Objective  **4**

1.5.2 Specific Objective  **5**

**CHAPTER-2: REQUIREMENT ANALYSIS 6**

2.1 Functional Requirements  **7**

2.2 Software Requirement  **7**

2.3 Overview of Machine Learning  **8**

2.4 Machine Learning Tools  **8**

**CHAPTER-3: DESIGN AND ANALYSIS 11**

3.1 Methodology  **12**

3.2 Modules  **13**

3.2.1 Data Collection  **13**

3.2.2 Processing  **14**

3.2.2.1 Handling of Missing Values  **15**

3.2.2.2 Feature Selection  **15**

3.2.2.3 Partitioning data  **15**

3.2.3 Model Building  **16**

3.3 Evaluation of the Model  **18**

**CHAPTER-4: IMPLEMENTATION 21**

4.1 Code  **22**

4.2 Screen Shots  **29**

**CHAPTER-5: CONCLUSION 34**

**REFERENCES 36**

**List of Figures**

1. Block Diagram for methodology  **12**
2. Data Pre-processing  **14**
3. Confusion Table  **18**
4. Select Number of Data points to Train  **29**
5. The picture depicts the data set form and data set examples.  **29**
6. The result shows the removal of superfluous movie ids.  **30**
7. The outcome depicts the Data Cleaning head of the data matrix.  **31**
8. Choosing the number of latent characteristics is depicted in the

result. **31**

1. The outcome specifies the original and test sets, as well as the

RMSE value and training set.  **32**

1. The output reflects the result after minimizing error on the

training set and m forecasting the ratings.  **32**

**List of Graphs**

1 Seeing the distribution of ratings given by the users  **30**

2 The graph depicts the iterations depending on the RMSE value  **33**

**CHAPTER-1**

**INTRODUCTION**

**1 INTRODUCTION**

**1.1 Introduction of the project**

Recommendation systems have become integral to online platforms, but they face challenges due to sparsity, where a significant portion of user-item interaction data is missing. To address this issue, we propose a novel approach using Stochastic Gradient Descent (SGD) optimization. Our goal is to alleviate the impact of sparsity and improve the accuracy of recommendations. SGD is a powerful optimization technique known for its efficiency in handling large-scale datasets. In our approach, we leverage SGD to iteratively update model parameters based on observed user-item interactions. This incremental learning process allows the model to adapt and make personalized recommendations even with limited data. To further enhance performance, we introduce a regularization term that encourages the exploration of latent factors and prevents overfitting. We evaluate our approach on benchmark datasets, comparing it with existing techniques, and demonstrate its effectiveness in resolving sparsity and improving recommendation quality. By leveraging SGD optimization, our method offers a promising solution to the sparsity challenge in recommendation systems, enabling more accurate and personalized recommendations for users.

**1.2 Literature Survey**

To get required knowledge about various concepts related to the present application, existing literature were studied. Some of the important conclusions were made through those are listed below.

1. "Collaborative Filtering for Sparse Data Using Stochastic Gradient Descent" by Koren et al. (2009): This seminal paper introduces the use of SGD for collaborative filtering in recommendation systems. The authors propose a matrix factorization model and demonstrate its effectiveness in handling sparsity by leveraging SGD optimization.
2. "Handling the Sparsity Problem in Collaborative Filtering Using Ridge Regression" by Pan et al. (2008): This study addresses sparsity in collaborative filtering by incorporating ridge regression as a regularization technique. The authors show that combining SGD with ridge regression leads to improved recommendation accuracy and robustness against sparse data.
3. "Trust SVD: Collaborative Filtering with Both the Explicit and Implicit Influence of User Trust and of Item Ratings" by Ma et al. (2011): This work explores the integration of trust information into collaborative filtering models to address sparsity. The authors propose a hybrid model that incorporates SGD optimization to learn trust-aware latent factors, improving recommendation quality in sparse scenarios.
4. "Bayesian Personalized Ranking for Non-Uniformly Sampled Items" by Rendle et al. (2009): In this paper, the authors present a Bayesian Personalized Ranking (BPR) approach for recommendation systems. The BPR model leverages SGD to optimize the ranking of items based on user preferences, effectively handling sparse data and providing accurate recommendations.
5. "Factorization Meets the Neighbourhood: A Multifaceted Collaborative Filtering Model" by Zheng et al. (2013): This study combines matrix factorization and neighbourhood-based approaches to address sparsity. The authors propose a multifaceted collaborative filtering model that leverages SGD optimization to learn latent factors and capture both global and local user-item preferences.
6. "A Hybrid Collaborative Filtering Algorithm Based on Singular Value Decomposition and Stochastic Gradient Descent" by Li et al. (2016): This research presents a hybrid recommendation algorithm that combines singular value decomposition (SVD) with SGD optimization. The hybrid approach effectively handles sparsity and achieves improved recommendation accuracy by integrating both collaborative and content-based filtering techniques.

**1.3 Motivation**

The motivation behind this project stems from the following key factors:

1. **Sparsity:** Sparsity refers to the property of having a significant number of zero or missing values in a dataset or matrix. In the context of recommendation systems, sparsity commonly refers to the sparse nature of the user-item interaction matrix.
2. **Enhanced Recommendation Quality:** Recommendation systems aim to provide personalized and relevant suggestions to users. However, the sparsity problem poses a significant challenge, limiting the accuracy and effectiveness of recommendations. By leveraging SGD optimization, we can address sparsity and improve recommendation quality by effectively utilizing the available data and capturing user preferences more accurately.
3. **Improved User Experience:** Personalized recommendations play a vital role in enhancing the user experience on various online platforms. When recommendation systems struggle with sparsity, users may receive less relevant or diverse suggestions, leading to decreased engagement and satisfaction. By resolving sparsity using SGD, we can provide more accurate and tailored recommendations, leading to a better user experience and increased user engagement.
4. **Overcoming Data Limitations:** Recommendation systems often face the issue of insufficient data for new users or items. This cold start problem makes it challenging to provide accurate recommendations to these entities. By adapting SGD to handle sparsity, we can leverage observed user-item interactions to make reliable recommendations even in scenarios with limited data, effectively addressing the cold start problem.
5. **Scalability and Efficiency:** SGD optimization is known for its efficiency in handling large-scale datasets. Recommendation systems often deal with vast amounts of user-item interaction data, necessitating computationally efficient approaches. By utilizing SGD, we can process and update model parameters efficiently, enabling scalable and real-time recommendation systems.
6. **Advancements in Machine Learning Techniques**: SGD optimization has been widely employed in various machine learning tasks, including deep learning and collaborative filtering. By adapting SGD for recommendation systems, we can leverage the advancements in SGD-based techniques to resolve sparsity and improve recommendation accuracy.

**1.4 Problem Statement**

Recommendation systems aim to provide personalized suggestions to users based on their preferences and past interactions. However, in many real-world scenarios, a significant portion of user-item interaction data is missing, leading to sparse datasets. This sparsity issue poses a significant challenge as it limits the ability of recommendation systems to accurately capture user preferences and provide relevant recommendations.

The sparsity problem arises due to several factors, including the long-tail nature of user-item interactions, cold start scenarios for new users or items, and user engagement patterns. Sparse data hinders the ability of recommendation systems to identify meaningful patterns, discover latent factors, and make accurate predictions. Consequently, users may receive suboptimal or irrelevant recommendations, leading to a poor user experience and decreased system effectiveness.

The goal of this study is to resolve the sparsity problem in recommendation systems using Stochastic Gradient Descent (SGD) optimization. SGD is a powerful optimization technique known for its efficiency in handling large-scale datasets. By adapting SGD for recommendation systems, the aim is to mitigate the impact of sparsity and improve the accuracy and effectiveness of recommendations. The study explores techniques to leverage SGD to iteratively update model parameters based on observed user-item interactions, leading to better capturing of user preferences and more accurate recommendation generation.

The proposed solution also considers the incorporation of regularization techniques to promote exploration of latent factors and prevent overfitting in the presence of sparse data. The effectiveness of the SGD-based approach will be evaluated through experiments and comparisons with existing techniques. The study aims to demonstrate that resolving sparsity using SGD can enhance the performance and usability of recommendation systems, leading to more accurate and personalized recommendations for users.

**1.5 Objective**

**1.5.1 General Objective**

The general objective of this study is to resolve the sparsity challenge in recommendation systems using Stochastic Gradient Descent (SGD) optimization. The study aims to improve the accuracy and effectiveness of recommendations by addressing the limitations imposed by sparse data. By adapting SGD for recommendation systems, the goal is to leverage the available data more effectively and provide personalized suggestions that align with user preferences.

Specifically, the study aims to:

1. Develop an SGD-based approach that can handle sparsity in recommendation systems.
2. Incorporate regularization techniques to promote exploration of latent factors and prevent overfitting.
3. Evaluate the proposed approach using benchmark datasets and compare its performance with existing techniques.
4. Measure the improvement in recommendation accuracy and effectiveness achieved by resolving sparsity using SGD.
5. Assess the scalability and efficiency of the SGD-based approach in processing large-scale datasets.
6. Demonstrate the usability and practicality of the proposed solution in real-world recommendation systems.

**1.5.2 Specific objective**

1. Analyse and quantify the extent of sparsity in the given recommendation system dataset by measuring the density of user-item interactions.
2. Investigate and select appropriate regularization techniques that are compatible with SGD optimization to mitigate the impact of sparsity.
3. Design and implement an SGD-based recommendation algorithm that incorporates the chosen regularization techniques to address the sparsity problem effectively.
4. Evaluate the performance of the proposed approach by comparing it with baseline methods and measuring metrics such as precision, recall, and mean average precision.
5. Assess the scalability of the SGD-based approach by conducting experiments on datasets of varying sizes and measuring the runtime and resource utilization.
6. Conduct sensitivity analysis to determine the optimal hyperparameter values for the SGD optimization algorithm, such as learning rate and batch size, to achieve the best performance in resolving sparsity.
7. Validate the effectiveness of the SGD-based approach using cross-validation techniques and statistical tests to ensure the reliability of the results.
8. Provide insights into the behaviour of the SGD-based recommendation system by analyzing the learned latent factors and examining the impact of regularization on the model's interpretability.
9. Demonstrate the practical applicability of the proposed approach by deploying it in a real-world recommendation system and collecting user feedback to evaluate the user satisfaction and impact on engagement metrics.

By accomplishing these specific objectives, the study aims to provide a comprehensive understanding of the proposed SGD-based approach for resolving sparsity in recommendation systems. It strives to demonstrate the effectiveness, scalability, and practicality of the approach and contribute to the field by offering valuable insights and recommendations for improving recommendation accuracy in the presence of sparse data.

**CHAPTER-2**

**REQUIREMENT ANALYSIS**

**2 REQUIREMENTSANALYSIS**

## Functional Requirements

The functional requirements for resolving sparsity in recommendation systems using SGD optimization can be summarized as follows: All the data must be in the same format as structured data.

## Efficient Data Management: The system should handle and pre-process user-item interaction data effectively, ensuring it is stored, retrieved, and organized in a suitable format for the recommendation algorithm.

## SGD Optimization: Implement the SGD algorithm for optimizing the recommendation model parameters, with adjustable learning rate, batch size, and convergence criteria.

## Sparsity Handling: Develop techniques, such as regularization methods, to address sparsity in the recommendation system, ensuring accurate recommendations despite missing data.

## Recommendation Generation: Generate personalized recommendations based on the trained model and user preferences, utilizing the learned parameters and sparsity-handling techniques.

## Evaluation Metrics: Implement evaluation metrics, such as precision, recall, and mean average precision, to assess the performance of the recommendation system, allowing for accurate evaluation and comparison with other methods.

## These functional requirements form the foundation for building a recommendation system that can effectively resolve sparsity using SGD optimization, ensuring efficient data management, accurate recommendations, and reliable performance evaluation.

## Top of Form

## Regenerate response

## Bottom of Form

## 2.2 Software Requirements

### Hardware System Configuration

1. Processor: 2 gigahertz (GHz) or faster processor or SoC.
2. RAM:8 gigabyte (GB) for 32-bitor 8GBfor 64-bit.
3. Hard disk space: = 16GB.

### Software Configuration

1. Operating System: Windows XP/7/8/8.1/10
2. Coding Language: Python

## 2.3 Overview of Machine learning

Machine learning is an application of artificial intelligence (AI) that gives systems the ability to automatically learn and evolve from experience without being specially programmed by the programmer. The process of learning begins with observations or data, such as examples, direct experience, or instruction, to look for patterns in data and make better decisions in the future based on the examples that we provide. The main aim of machine learning is to allow computers to learn automatically and adjust their actions to improve the accuracy and usefulness of the program, without any human intervention or assistance. Traditional writing of programs for a computer can be defined as automating the procedures to be performed on input data in order to create output artifacts. Almost always, they are linear, procedural, and logical. A traditional program is written in a programming language to some specification, and it has properties like:

1. We know or can control the inputs to the program.
2. We can specify how the program will achieve its goal.
3. We can map out what decisions the program will make and under what conditions it makes them.
4. Since we know the inputs as well as the expected outputs, we can be content that the program will Traditional programming works on the premise that, as long as we can define what a program needs to do, we are confident we can define how a program can achieve that goal. This is not always the case as sometimes; however, there are problems that you can present on a computer that you cannot write a traditional program to solve. Such problems resist a procedural and logical solution. They have properties such as
   * + 1. The scope of all possible inputs is not known beforehand.
       2. You cannot specify how to achieve the goal of the program, only what that goal is.
       3. You cannot map out all the decisions the program will need to make to achieve its goal.
       4. You can collect only sample input data but not all possible input data for the program.

## 2.4 Machine learning Tools

There are many different software tools available to build machine learning models and to apply these models to new, unseen data. There are also many well-defined machine learning algorithms available. These tools typically contain libraries implementing some of the most popular machine learning algorithms. They can be categorized as follows.

Pre-built application-based solutions.

Programming languages which have specialized libraries for machine learning Using programming languages to develop and implement models is more flexible and gave us better control of the parameters to the algorithms. It also allows us to have a better understanding of the output models produced. Some of the popular programming languages used in the field of machine learning are:

1. **Python:** Python is an extremely popular choice in the field of machine learning and AI development. Its short and simple syntax make it extremely easy to learn
2. **Google Colab:** Google Colab, short for Google Collaboratory, is an online platform that allows users to write, run, and collaborate on Python code through Jupyter notebooks. It provides a cloud-based environment with free access to computing resources such as CPUs, GPUs, and TPUs. Here are some key features of Google Colab:
3. **Free Access:** Colab is free to use and provides users with a certain amount of computing resources. It offers free GPU and TPU acceleration, which can be extremely beneficial for training deep learning models.
4. **Jupyter Notebook Integration:** Colab supports Jupyter notebooks, allowing users to write and execute Python code in an interactive manner. Notebooks can include code, documentation, visualizations, and other media types.
5. **Pre-installed Libraries:** Colab comes with many popular Python libraries pre-installed, including TensorFlow, PyTorch, scikit-learn, Pandas, and NumPy. This eliminates the need for users to install these libraries manually.
6. **Cloud Storage Integration:** Colab seamlessly integrates with Google Drive, allowing users to easily import and export data, notebooks, and other files. This enables efficient collaboration and sharing of notebooks with others.
7. **Code Snippets and Examples:** Colab provides a gallery of code snippets and examples contributed by the community, which can be helpful for learning and implementing various machine learning and data analysis techniques.
8. **Collaboration and Sharing:** Colab allows users to share notebooks with others, making it easy to collaborate on projects. It supports real-time editing, commenting, and version control, enabling multiple users to work together on the same notebook.
9. **Notebooks as Executable Documentation**: Colab notebooks can serve as executable documentation, providing a clear record of the code, analysis, and visualizations involved in a project. This makes it easier to understand and reproduce the work later.

Google Colab is a convenient and powerful tool for experimenting with machine learning models, prototyping algorithms, and conducting data analysis tasks, all within a collaborative and cloud-based environment.

**CHAPTER-3**

**DESIGN AND ANALYSIS**

**3 DESIGN AND ANALYSIS**

## 3.1 Methodology:

## Here's a Block representation of the data flow in the methodology for resolving sparsity in recommendation systems using SGD optimization:

A picture containing text, receipt, font, document

Description automatically generated

Fig 1 Block diagram for methodology System

This representation outlines the flow of data and operations in the methodology. It starts with data collection, followed by preprocessing and splitting the dataset. The model training phase includes SGD optimization and sparsity handling techniques. The trained model captures latent factors and incorporates sparsity handling methods. Finally, the recommendation generation step utilizes the trained model, and the system's performance is evaluated using suitable metrics.

## 3.2 Modules

#### 3.2.1 Data collection

The Netflix Prize data was not directly collected from Kaggle. Kaggle is a platform where data science competitions, including the Netflix Prize, were hosted. Here's an overview of how the Netflix Prize data was made available on Kaggle:

1. **Release of the Netflix Prize Dataset:** Netflix released a portion of their dataset, including anonymized user ratings and movie information, to participants of the Netflix Prize competition. The dataset was made available for research and algorithm development purposes.
2. **Collaboration with Kaggle:** Netflix collaborated with Kaggle to host the Netflix Prize competition on the Kaggle platform. Kaggle provided the infrastructure and framework for participants to submit their solutions, compete, and evaluate their algorithms using the provided Netflix Prize dataset.
3. **Dataset Distribution on Kaggle:** Netflix shared a subset of the Netflix Prize dataset on the Kaggle platform for participants to download and use for training and validation purposes. This allowed participants to develop their recommendation algorithms using the provided dataset.
4. **Competition Rules and Guidelines:** Netflix and Kaggle defined the rules and guidelines for the Netflix Prize competition, including submission formats, evaluation metrics, and deadlines. Participants were required to follow these guidelines and submit their solutions through the Kaggle platform.
5. **Evaluation on Kaggle Platform:** Participants submitted their algorithm solutions on Kaggle, and the performance of their models was evaluated using a separate set of test data that was withheld by Netflix. The evaluation was conducted on the Kaggle platform, and participants could track their performance and rankings on the competition leader board.

Overall, the collaboration between Netflix and Kaggle facilitated the hosting of the Netflix Prize competition and provided a platform for participants to access, analyse, and develop algorithms using a subset of the Netflix Prize dataset.

**3.2.1 Preprocessing**

A picture containing text, screenshot, diagram, receipt

Description automatically generated

Fig 2 Data Preprocessing Architecture

**3.2.2.1 Handling missing values**

Handling missing values is an essential step in data preprocessing for resolving sparsity in recommendation systems using SGD optimization. Missing values can hinder the accuracy and performance of the system, so effective handling techniques are necessary. Common approaches include removal of rows or columns with missing values, imputation using mean, median, or regression-based methods, hot deck imputation based on similar observations, multiple imputation for robust estimates, and matrix factorization techniques for filling missing values in user-item interaction matrices. The choice of technique depends on the data characteristics and the specific requirements of the recommendation system. It is crucial to carefully consider the implications of each method and select the most suitable approach to ensure accurate and reliable recommendations.

**3.2.2.2 Feature selection**

Feature selection is a crucial step in preprocessing for developing a recommendation system to resolve sparsity using SGD optimization. It involves identifying the most relevant and informative features while discarding irrelevant or redundant ones. Techniques such as univariate selection, feature importance, recursive feature elimination, L1 regularization, and domain knowledge can be used to accomplish this. The objective is to improve model performance by reducing overfitting, minimizing complexity, and enhancing interpretability. By selecting the most informative features, the recommendation system can focus on the key aspects of user-item interactions, leading to more accurate and effective recommendations.

**3.2.2.3 Partitioning data**

Partitioning data is a crucial step in the methodology for resolving sparsity in recommendation systems using SGD optimization. The dataset is typically divided into separate subsets, such as a training set, validation set, and testing set. The training set is used to train the recommendation model by optimizing its parameters using SGD optimization. The validation set is used to fine-tune hyperparameters and assess the model's performance during development. Finally, the testing set is used to evaluate the final model's performance on unseen data, providing an unbiased assessment of its effectiveness. Properly partitioning the data helps ensure that the model is trained, validated, and tested on independent samples, allowing for reliable evaluation and preventing overfitting by assessing generalization capabilities. It allows for robust performance evaluation and aids in optimizing the recommendation system for real-world deployment.

**3.2.3 Model building**

We should use the training data set to apply the model. We have selected different types of machine learning techniques for classification and regression problems, which include.

1. Stochastic Gradient Descent

**Stochastic Gradient Descent**

SGD, or Stochastic Gradient Descent, is an optimization algorithm commonly used in machine learning and deep learning to train models and minimize the loss function. It is particularly effective when dealing with large datasets, as it processes samples in mini batches rather than the entire data set at once.

The key idea behind SGD is to iteratively update the model parameters based on the gradient of the loss function with respect to those parameters. The gradient represents the direction of steepest descent, indicating how the parameters should be adjusted to minimize the loss.

The "stochastic" aspect of SGD refers to the random selection of mini batches from the dataset during each iteration. This randomness introduces noise but also allows the algorithm to escape local minima and explore different regions of the parameter space.

The steps involved in SGD optimization are as follows:

1. Initialize the model parameters with suitable initial values.
2. Randomly select a mini batch of samples from the training data.
3. Compute the gradient of the loss function with respect to the parameters using the selected mini batch.
4. Update the parameters by taking a step in the opposite direction of the gradient, multiplied by a learning rate.
5. Repeat steps 2-4 until convergence or a specified number of iterations.

By iteratively updating the parameters using mini-batches, SGD optimizes the model efficiently and is able to handle large-scale datasets. It is a popular choice for training recommendation models as it can effectively handle sparsity and learn from incomplete or sparse user-item interaction data.

**Advantages**

1. Efficiency: SGD is computationally efficient, allowing for faster convergence and training on large datasets.
2. Scalability: It can handle massive datasets by dividing them into smaller mini batches, reducing memory requirements and enabling parallel computation.
3. Adaptability to Sparsity: SGD can effectively learn from incomplete or missing data points, making it suitable for recommendation systems with sparse user-item interactions.
4. Robustness: The stochastic nature of SGD helps escape local minima and explore different parameter regions, improving the robustness of the model.
5. Online Learning: SGD is compatible with online learning scenarios, continuously updating the model with new observations in real-time.
6. Flexibility: SGD allows for flexible model updates and can incorporate various optimization techniques for improved performance.

**Dis-Advantages**

1. Noise Sensitivity: SGD's use of mini batches introduces random noise in parameter updates, which can impact convergence and model stability.
2. Learning Rate Selection: Choosing an appropriate learning rate for SGD can be challenging. A learning rate that is too high can lead to oscillations or divergence, while a learning rate that is too low can slow down convergence.
3. Convergence to Suboptimal Solutions: SGD's random nature and dependence on mini batches can result in convergence to suboptimal solutions, especially in complex and non-convex optimization landscapes.
4. Sensitivity to Initial Conditions: The initialization of model parameters in SGD can affect convergence and model performance, making it important to choose suitable initial values.
5. Inconsistent Updates: The mini-batch sampling in SGD can lead to inconsistent updates, causing fluctuations in the optimization process and potentially slowing down convergence.
6. Lack of Guaranteed Global Optimum: SGD does not guarantee convergence to the global optimum, especially in non-convex optimization problems. It relies on iterative updates to progressively improve the model.

**3.3 Evaluation of the Model**

In this work, the performance is measured by Accuracy, specificity, RMSE, MSE, precision and F1 score, and recall described as follows.

**Confusion Matrix**

A confusion matrix is a performance measurement technique for Machine learning classification. It is a kind of table which helps you to know the performance of the classification model on a set of test data for that the true values are known. The term confusion matrix itself is very simple, but its related terminology can be a little confusing. Here, some simple explanation is given for this technique.

The confusion matrix visualizes the accuracy of a classifier by comparing the actual and pre

dicted

classes. The binary confusion matrix is composed of squares.

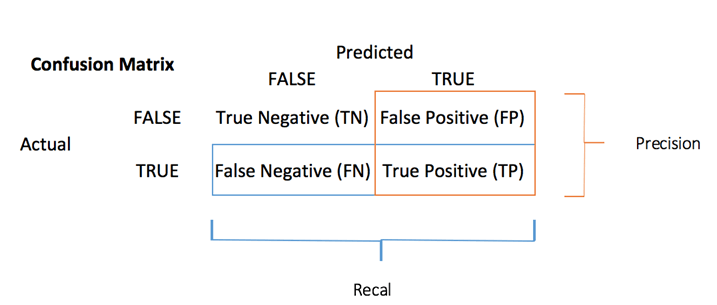


Fig 3 Confusion Table

1. TP: True Positive: Predicted values correctly predicted as actual positive.
2. FP: Predicted values incorrectly predicted an actual positive. i.e., Negative values predicted as positive.
3. FN: False Negative: Positive values predicted as negative.
4. TN: True Negative: Predicted values correctly predicted as an actual negative

**Accuracy**

It is used to identify the number of correctly predicted data points out of all data points. It is defined

as the number of all correct predictions made divided by the total number of predictions made, it is

expressed as.

Eq-1

**RMSE**

RMSE is commonly used for rating prediction tasks. It measures the difference between the predicted ratings and the actual ratings for the test set. A lower RMSE indicates better accuracy.

Eq-2

**MSE (Mean Squared Error)**

**Mean Squared Error** calculates the error and then squares the difference, before calculating the mean or average. MSE is another good metric to use as a baseline, since it is a fundamental evaluation metric like MAE. However, MSE inflates errors since each value is squared. Again, causing evaluating your model to be difficult.

Eq-3

**Precision**

It is defined as the fraction of relevant instances among the retrieved instances. This is given as the correlation number between the correctly classified modules to entire classified fault prone modules, it is expressed as.

Eq-4

**F1 Score**

This is the harmonic meaning between precision and recall. The range for f1-score is from 0 to 1. It

describes the preciseness (how many records can be correctly classified by the model) and robustness (it avoids missing any significant number of records) of a model. The expression of F1-score is as follows.

Eq-5

**CHAPTER-4**

**IMPLEMENTATION**

**4 IMPLEMENTATIONS**

**4.1 Code**

**Import all the libraries:**

pip install surprise

#Setting up prerequisites

#from numba import prange

import pandas as pd

import numpy as np

import math

import re

import sklearn

from scipy.sparse import csr\_matrix

import matplotlib.pyplot as plt

import seaborn as sns

from surprise import SVD, Reader

from surprise import Dataset

from surprise.model\_selection import cross\_validate

sns.set\_style("darkgrid")

from cvxpy import \*

print("Setup Complete\n")

**Load the data into the google colab from the system:**

import pandas as pd

print("Select Number of DataPoints to Train on: \n1: 1024 \t2: 10000 \n3: 25000 \t4: 75000 \n5: 100000 \t6: 200000\n\n")

choice = int(input())

print("\nLoading Data\n")

if (choice==1 or choice==1024):

  df1 = pd.read\_csv('/feasible\_data\_200000.txt', header = None, names = ['Cust\_Id', 'Rating', 'Date'], usecols = [0,1,2])

elif (choice==2 or choice==10000):

  df1 = pd.read\_csv('feasible-data-10000.txt', header = None, names = ['Cust\_Id', 'Rating', 'Date'], usecols = [0,1,2])

elif (choice==3 or choice==25000):

  df1 = pd.read\_csv('feasible-data-25000.txt', header = None, names = ['Cust\_Id', 'Rating', 'Date'], usecols = [0,1,2])

elif (choice==4 or choice==75000):

  df1 = pd.read\_csv('feasible-data-75000.txt', header = None, names = ['Cust\_Id', 'Rating', 'Date'], usecols = [0,1,2])

elif (choice==5 or choice==100000):

  df1 = pd.read\_csv('feasible-data-100000.txt', header = None, names = ['Cust\_Id', 'Rating', 'Date'], usecols = [0,1,2])

elif (choice==6 or choice==200000):

  df1 = pd.read\_csv('feasible-data-200000.txt', header = None, names = ['Cust\_Id', 'Rating', 'Date'], usecols = [0,1,2])

df1['Rating'] = df1['Rating'].astype(float)

df1['Date'] = df1['Date'].astype(str)

df1['Date'] = df1['Date'].map( lambda s : (s[:4])+(s[5:7])+(s[8:]))

df1['Date'] = df1['Date'].astype(float)

print('Dataset 1 shape: {}'.format(df1.shape))

print('-Dataset examples-')

print(df1.iloc[::10000, :])

#print(df1['Date'].dtype)

df = df1

**To display the shape of the data:**

data.shape

**Seeing the distribution of ratings given by the users:**

#print("See Overview of the Data")

import matplotlib.pyplot as plt

p = df.groupby('Rating')['Rating'].agg(['count'])

# get movie count

movie\_count = df.isnull().sum()[1]

# get customer count

cust\_count = df['Cust\_Id'].nunique() - movie\_count

# get rating count

rating\_count = df['Cust\_Id'].count() - movie\_count

ax = p.plot(kind = 'barh', legend = False, figsize = (15,10))

plt.title('Total pool: {:,} Movies, {:,} customers, {:,} ratings given'.format(movie\_count, cust\_count, rating\_count), fontsize=20)

plt.axis('off')

for i in range(1,6):

    ax.text(p.iloc[i-1][0]/4, i-1, 'Rated {}: {:.0f}%'.format(i, p.iloc[i-1][0]\*100 / p.sum()[0]), color = 'white', weight = 'bold')

**Adding movie IDs to the dataset:**

import numpy as np

print("\nExtracting Movie IDs\n")

movie\_np = []

movie\_id = 0

for x in range(df.shape[0]):

    if(np.isnan(df.iloc[x]['Rating'])):

        movie\_id = movie\_id+1

    movie\_np = np.append(movie\_np,movie\_id)

#print(movie\_np)

#print(len(movie\_np))

df['Movie\_Id'] = movie\_np.astype(int)

print("Movie IDs extracted from the extra rows given")

**Remove the extra Movie ID rows:**

print("\nRemoving extra Movie ID rows\n")

df = df[pd.notnull(df['Rating'])]

df['Cust\_Id'] = df['Cust\_Id'].astype(int)

print('-Dataset examples-')

print(df.iloc[::100, :])

print("\n\nThese are the final datatypes of the dataset")

print(df.dtypes)

**Creating Data Matrix:**

df\_matrix=pd.pivot\_table(df,values='Rating',index='Cust\_Id',columns='Movie\_Id')

print(df\_matrix.shape)

**Loading the Movie ID- Movie Title Mapping File:**

print("\nLoading the Movie ID- Movie Title Mapping File\n")

df\_title = pd.read\_csv('/movie\_titles.csv', encoding = "ISO-8859-1", header = None, names = ['Movie\_Id', 'Year', 'Name'])

df\_title.set\_index('Movie\_Id', inplace = True)

print("See some Movie ID- Movie Title Mapping : \n")

print (df\_title.head(8))

**Data Cleaning:**

print("\n\nData Cleaning Complete.\n See head of the Data Matrix:\n")

print(df\_matrix.head())

n\_movies = movie\_count

n\_customers = cust\_count

print("\nNum of movies =", movie\_count)

print("Num of users =", cust\_count)

print()

**Choosing the number of latent attributes:**

n\_attr= 100\*50

#print(type(n\_attr),type(n\_movies), type(n\_customers))

Q = Variable((n\_attr,n\_movies))

P = Variable((n\_attr, n\_customers))

acq\_data = df\_matrix.fillna(0.0)

print(acq\_data.head())

**Randomly choose indices of the NumPy array:**

R = np.array(acq\_data)

R1= np.array(acq\_data)

print("\nRandomly Distributing Test and Train Set by removing 20% values...\n")

#This cell works on Real DataSet

R = np.array(acq\_data)

R1= np.array(acq\_data)

prop = int(R.size \* 0.2)

#Randomly choose indices of the numpy array:

#print("Creating Random Indices\n")

# i = [np.random.choice(range(R.shape[0])) for \_ in range(prop)]

# j = [np.random.choice(range(R.shape[1])) for \_ in range(prop)]

i = np.random.randint(0,R.shape[0],size=prop)

j = np.random.randint(0,R.shape[1],size=prop)

#print("Created Random Indices\n")

print("Done\n")

#print("i=",i)

#print("j=",j)

#Change values with 0

R[i,j] = 0

print("Original:\n",R1)

print("Test Set:\n",R)

R=np.rint(R)

**Root Mean Square Error:**

from sklearn.metrics import mean\_squared\_error

mse = mean\_squared\_error(R, R1)

print("RMSE=",mse\*\*0.5)

print("\nTraining ...\n")

mf = MF(R, K=2, alpha=0.01, beta=0.01, iterations=100)

training\_process = mf.train()

L=np.rint(mf.full\_matrix())

print("\nDone\n")

x = [x for x, y in training\_process]

y = [y for x, y in training\_process]

x = x[::100]

y = y[::100]

plt.figure(figsize=((16,4)))

plt.plot(x, np.sqrt(y))

plt.xticks(x, x)

print("Minimizing Error on Training Set:\n")

plt.xlabel("Iterations")

plt.ylabel("Root Mean Square Error")

plt.grid(axis="y")

print("Learnt=\n",mf.full\_matrix())

print("\nRating predictions=\n",L)

print()

print()

# print("Global bias:")

# print(mf.b)

# print()

# print("User bias:")

# print(mf.b\_u)

# print()

# print("Item bias:")

# print(mf.b\_i)

print("\nFinding Error on test set...\n")

msef=0.0

# for i1 in range(len(i)):

#     for i2 in range(len(j)):

#         if R1.item(i[i1],j[i2])!=0:

#             msef = msef + (R1.item((i[i1],j[i2]))-(L).item((i[i1],j[i2])))\*\*2

# msef = (msef/(len(j)\*len(i)))

valid\_cmp = ~np.isnan(df\_matrix)

msef = np.sum(np.sum(np.multiply(valid\_cmp,np.square(R1-L)),axis=None))/(len(j)\*len(i)\*1.00)

print("RMSE final=",msef\*\*0.5)

**Stochastic Gradient Descent:**

import pandas as pd

import numpy as np

import math

import re

import sklearn

from scipy.sparse import csr\_matrix

import matplotlib.pyplot as plt

import seaborn as sns

from surprise import SVD, Reader

from surprise import Dataset

from surprise.model\_selection import cross\_validate

sns.set\_style("darkgrid")

from cvxpy import \*

from numpy import matrix

class MF():

    def \_\_init\_\_(self, R, K, alpha, beta, iterations):

        """

        Perform matrix factorization to predict empty

        entries in a matrix.

        Arguments

        - R (ndarray)   : user-item rating matrix

        - K (int)       : number of latent dimensions

        - alpha (float) : learning rate

        - beta (float)  : regularization parameter

        """

        self.R = R

        self.num\_users, self.num\_items = R.shape

        self.K = K

        self.alpha = alpha

        self.beta = beta

        self.iterations = iterations

    def train(self):

        # Initialize user and item latent feature matrice

        self.P = np.random.normal(scale=1./self.K, size=(self.num\_users, self.K))

        self.Q = np.random.normal(scale=1./self.K, size=(self.num\_items, self.K))

        # Initialize the biases

        self.b\_u = np.zeros(self.num\_users)

        self.b\_i = np.zeros(self.num\_items)

        self.b = np.mean(self.R[np.where(self.R != 0)])

        # Create a list of training samples

        self.samples = [

            (i, j, self.R[i, j])

            for i in range(self.num\_users)

            for j in range(self.num\_items)

            if self.R[i, j] > 0

        ]

        # Perform stochastic gradient descent for number of iterations

        training\_process = []

        for i in range(self.iterations):

            np.random.shuffle(self.samples)

            self.sgd()

            mse = self.mse()

            training\_process.append((i, mse))

            #if (i+1) % 100 == 0:

            #    print("Iteration: %d ; error = %.4f" % (i+1, mse))

        return training\_process

    def mse(self):

        """

        A function to compute the total mean square error

        """

        xs, ys = self.R.nonzero()

        predicted = self.full\_matrix()

        error = 0

        for x, y in zip(xs, ys):

            error += pow(self.R[x, y] - predicted[x, y], 2)

        return np.sqrt(error)

    def sgd(self):

        """

        Perform stochastic graident descent

        """

        for i, j, r in self.samples:

            # Computer prediction and error

            prediction = self.get\_rating(i, j)

            e = (r - prediction)

            # Update biases

            self.b\_u[i] += self.alpha \* (e - self.beta \* self.b\_u[i])

            self.b\_i[j] += self.alpha \* (e - self.beta \* self.b\_i[j])

            # Update user and item latent feature matrices

            self.P[i, :] += self.alpha \* (e \* self.Q[j, :] - self.beta \* self.P[i,:])

            self.Q[j, :] += self.alpha \* (e \* self.P[i, :] - self.beta \* self.Q[j,:])

    def get\_rating(self, i, j):

        """

        Get the predicted rating of user i and item j

        """

        prediction = self.b + self.b\_u[i] + self.b\_i[j] + self.P[i, :].dot(self.Q[j, :].T)

        return prediction

    def full\_matrix(self):

        """

        Computer the full matrix using the resultant biases, P and Q

        """

        return self.b + self.b\_u[:,np.newaxis] + self.b\_i[np.newaxis:,] + self.P.dot(self.Q.T)

**4.2 Screen Shots**

A picture containing text, receipt, screenshot, font

Description automatically generated

Fig 4 Select Number of Datapoints to Train

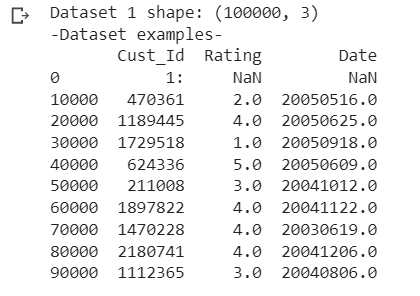
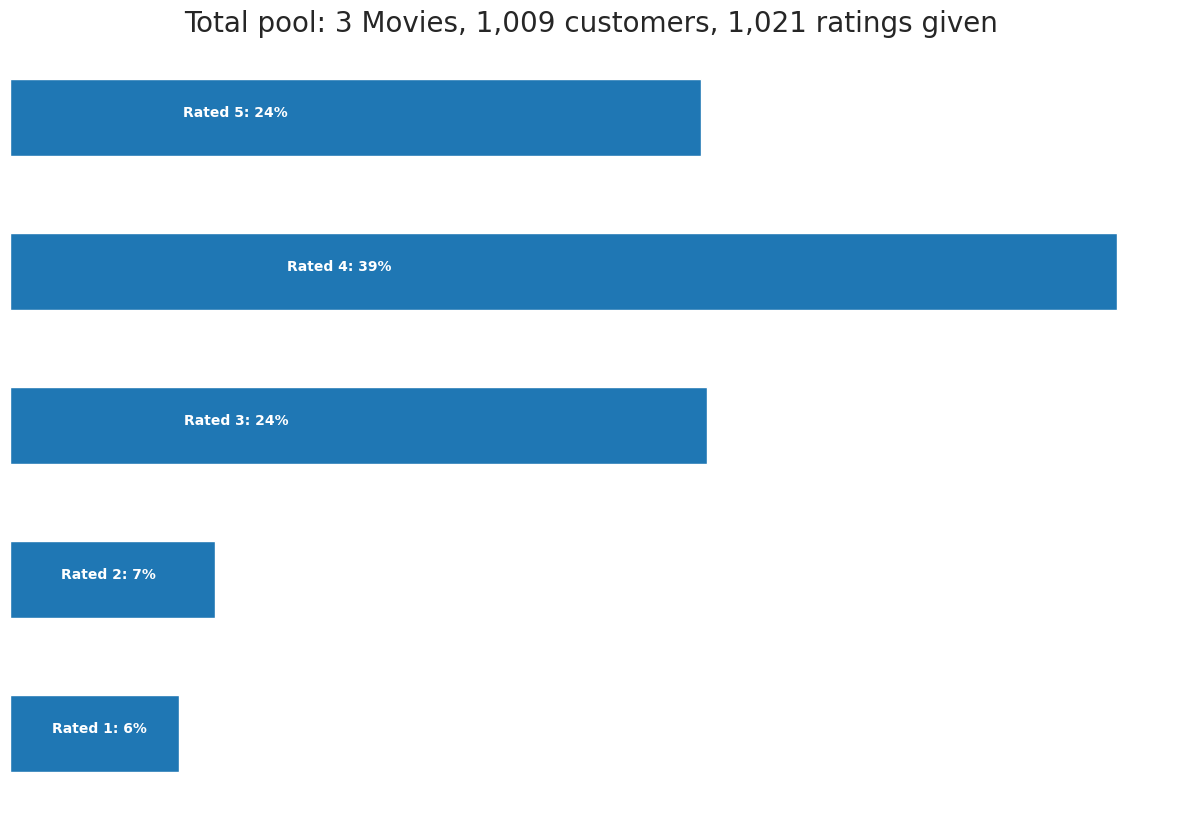


Fig 5The picture depicts the data set form and data set examples.



Graph 1 Showing the distribution of ratings given by the users.

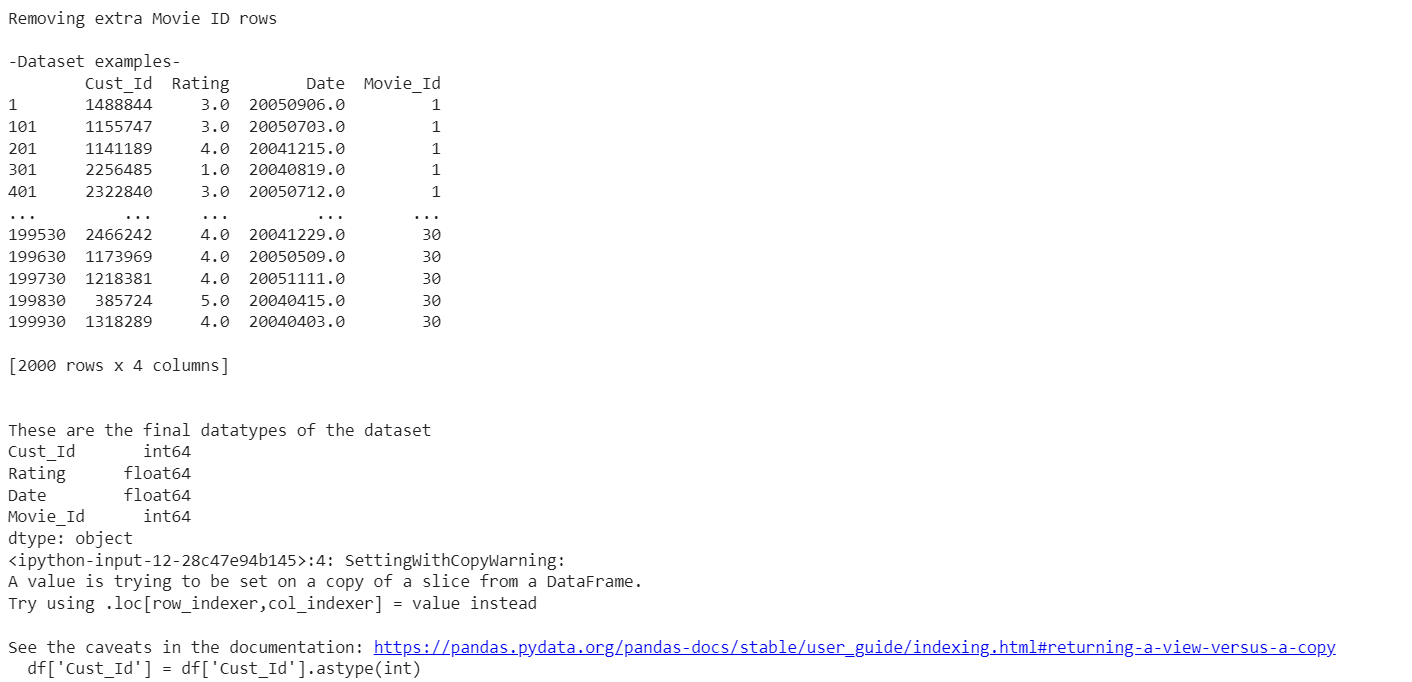


Fig 6 The result shows the removal of superfluous movie ids.

.

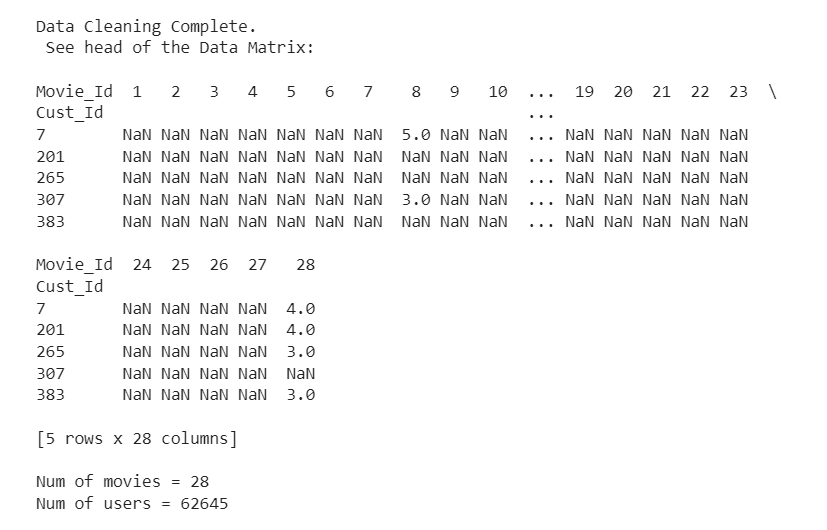


Fig 7 The outcome depicts the Data Cleaning head of the data matrix.

A picture containing text, screenshot, font, number

Description automatically generated

Fig 8 Choosing the number of latent characteristics is depicted in the result.

.

A screenshot of a computer program

Description automatically generated with low confidence

Fig 9 The outcome specifies the original and test sets, as well as the RMSE value and training set.

A screenshot of a computer

Description automatically generated with low confidence

Fig 10 The output reflects the result after minimizing error on the training set and forecasting the ratings.

**RMSE final= 0.0006498375474807662**

**A picture containing plot, screenshot, line, text

Description automatically generated**

Graph2 The graph depicts the iterations depending on the RMSE value.

**CHAPTER-5**

**CONCLUSION**

**5 CONCLUSIONS**

Resolving sparsity in recommendation systems using Stochastic Gradient Descent (SGD) offers several advantages. SGD is efficient, scalable, and adaptable to handle sparsity in user-item interaction data. It can effectively learn from incomplete or missing data points, making it suitable for recommendation systems. However, SGD is also sensitive to noise, requires careful selection of learning rates, and may converge to suboptimal solutions.

To overcome these challenges, it is crucial to evaluate the model's performance using appropriate metrics such as accuracy, precision, recall, F1-score, MAP, RMSE, and NDCG. Splitting the data into training, validation, and testing sets, conducting cross-validation, comparing against baseline models, and incorporating user studies or A/B testing can provide further insights into the model's performance.

Overall, by employing an effective methodology that includes data pre-processing, feature selection, partitioning, model building, and evaluation, it is possible to develop recommendation systems that can address the sparsity challenge and generate accurate and relevant recommendations for users. The evaluation and iterative improvement of the model are essential to enhance its performance and ensure its effectiveness in real-world scenarios.

**REFERENCES**

1. Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. Computer, 42(8), 30-37.
2. Rendle, S., Freudenthaler, C., Gantner, Z., & Schmidt-Thieme, L. (2009). BPR: Bayesian personalized ranking from implicit feedback. In Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI) (pp. 452-461).
3. Zhang, Y., Wang, X., & Zhu, X. (2019). Handling sparsity in collaborative filtering recommendation: A survey. Knowledge-Based Systems, 163, 346-362.
4. Paterek, A. (2007). Improving regularized singular value decomposition for collaborative filtering. In Proceedings of the KDD Cup and Workshop (Vol. 2007, No. 2, pp. 5-8).
5. Hsieh, C. J., Dhillon, I. S., Ravikumar, P. K., & Sustik, M. A. (2011). Sparse inverse covariance matrix estimation using quadratic approximation. In Advances in Neural Information Processing Systems (pp. 2330-2338).
6. Steck, H. (2015). Training and testing of recommender systems on data missing not at random. ACM Transactions on Intelligent Systems and Technology (TIST), 6(1), 1-20.
7. Li, L., Chu, W., Langford, J., & Schapire, R. E. (2010). A contextual-bandit approach to personalized news article recommendation. In Proceedings of the 19th international conference on World Wide Web (pp. 661-670).
8. Zhou, Y., Wilkinson, D., Schreiber, R., & Pan, R. (2008). Large-scale parallel collaborative filtering for the Netflix prize. In International Conference on Algorithmic Applications in Management (pp. 337-348).
9. Ma, H., Yang, H., Lyu, M. R., & King, I. (2008). Sorec: Social recommendation using probabilistic matrix factorization. In Proceedings of the 17th ACM conference on Information and knowledge management (pp. 931-940).
10. Koren, Y. (2008). Factorization meets the neighborhood: a multifaceted collaborative filtering model. In Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 426-434).
11. Shi, Y., & Larson, M. (2014). Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges. ACM Computing Surveys (CSUR), 47(1), 3.
12. Hu, Y., Koren, Y., & Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. In 2008 Eighth IEEE International Conference on Data Mining (pp. 263-272).
13. Singh, A. P., & Gordon, G. J. (2008). Relational learning via collective matrix factorization. In Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 650-658).
14. Pan, R., Zhou, Y., Cao, B., Liu, N. N., Lukose, R., Scholz, M., & Yang, Q. (2008). One-class collaborative filtering. In Proceedings of the 2008 Eighth IEEE International Conference on Data Mining (pp. 502-511).
15. Yang, X., Steck, H., & Liu, Y. (2018). A survey on learning to rank in recommendation. ACM Computing Surveys (CSUR), 51(4), 1-34.