Poisson Matrix Factorization for TV Recommendations

by

Ashley King

Honors Thesis

Appalachian State University

Submitted to the Department of Computer Science

in partial fulfillment of the requirements for the degree of

Bachelor of Science

May 2021

APPROVED BY:

R. Mitchell Parry, Ph.D., Thesis Project Director

Dr. James B. Fenwick Jr., Second Reader

Raghuveer Mohan, Ph.D., Departmental Honors Director

Rahman Tashakkori., Ph.D., Chair, Computer Science

ABSTRACT

Poisson Matrix Factorization for TV Recommendations.

(May 2021)

Ashley King, Appalachian State University

Appalachian State University

Thesis Chairperson: R. Mitchell Parry, Ph.D.

Recommendation systems are becoming more and more popular within e-commerce websites to help drive user engagement. It is not just limited to e-commerce though, websites such as Netflix or Spotify utilize recommendation systems to better engage users in movies and TV shows, or music. This thesis explores the mathematics and assumptions behind recommendation systems, such as how data is distributed and different algorithms used. The thesis then performs a case study on Reddit TV show data to build a recommendation system. To improve the results of the recommendation system, this thesis makes changes to a Python Recommendation System Library to enable Poisson Factorization. The changes proposed can be integrated into the existing Python library, helping other programmers make more meaningful and accurate recommendations.

Contents

List of Tables

Chapter 1

Introduction

Recommendation systems have revolutionized the way e-commerce and streaming services operate by increasing user interaction while decreasing the amount of information presented to users. Users typically have a short attention span while scrolling through websites, so it is vital to only show information that is relevant to the user. According to Netflix, users often lose interest while scrolling through their feed after 60 to 90 seconds [7]. Netflix estimates that over 80% of total hours watched is attributed to their recommendation system. The remaining 20% screen time is attributed to searching or other lookup functions [7].

Recommendation systems can take on many forms, but have the same central goal of recommending items to users based on their existing preferences. The definition of *user*, *item*, and *preference* can vary by application. For example, Netflix uses a recommendation engine which recommends movies or TV shows (items) to their subscribers (users) based on previous shows or movies they watched (preference). Amazon recommends products (items) to shoppers (users) based on their previous purchases (preferences). Spotify also utilizes recommendation systems to recommend music (items) to their music listeners (users) based on previous music they have listened to (preferences).

This paper outlines recommendation systems, improves upon an existing library, and evaluates a SubReddit TV Show recommendation system based upon the number of comments left by a user.

Chapter 2

Recommendation Background

2.1 History of Recommendation Systems

A recommendation system aims to recommend the most relevant items to a user. Famous examples are the Netflix recommendation system, which aims to recommend TV shows and movies to users, or Spotify, which recommends songs based on the users listening habits.

One of the earliest machine recommendation systems was presented in 1979, and outlined how to train computers to treat users as "distinct personalities, goals, and so forth"[15]. Rich created a basic collaborative filtering algorithm, called Grundy, to make library recommendations. The system was very basic, taking in user input in the form of their name to determine if they had used the library before. If they had not, the system would ask them to input personally descriptive adjectives, such as "unconventional open direct honest humorous persistent adventurous." Based on this input, for example since the user input "humorous," it would recommend comedy novels. Or since the user also inputted "adventurous," it might recommend action or mystery novels.

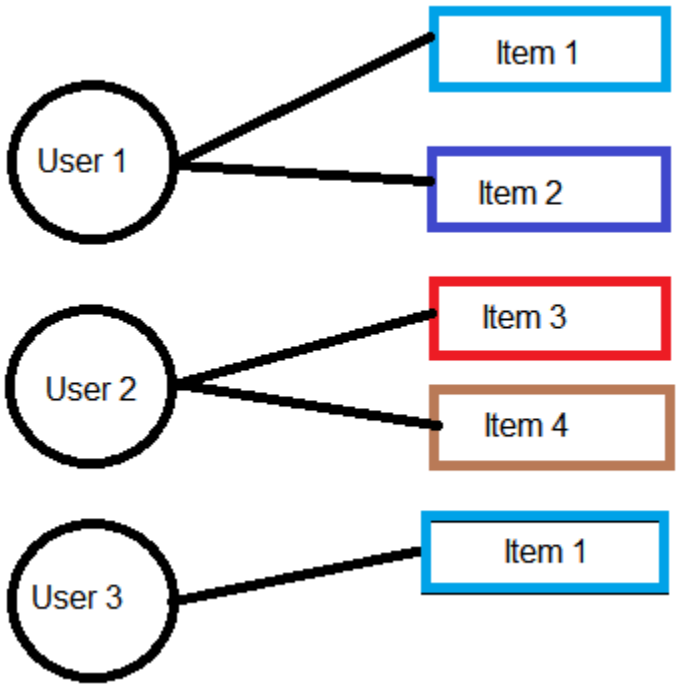
This system was very basic, and there were issues with having users manually input words to describe themselves. They might not choose the right words that encapsulate what novels they prefer, or the system might not have recommendations for a specific word. But this laid the framework for improvements and for future recommendation systems.

2.2 Types of Recommendation Systems

Recommendation systems provide meaningful recommendations to users based on previous data. What varies is how these recommendations are generated. Some differences in types of recommendation systems are if the relationships between items or users is utilized to make recommendations, or if "informative content descriptors" are utilized to make recommendations [4]. For example, collaborative filtering explores the relationships between users to provide recommendations, while content-based filtering explores the relationship between items using descriptive features [4].

Collaborative filtering

Collaborative filtering provides recommendations based upon the opinions of similar users [4].



In the above example, User 1 interacted or showed preference for Items 1 and 2. User 2 interacted or showed preference for Items 4 and 5. To generate a recommendation to User 3 for what item they would like, the system would explore which users are most similar to User 3. Since User 3 has interacted with Item 1 in the past, the recommendation system would calculate that there is a similarity between User 1 and User 3. This is because User 1 showed a

preference for Item 1, and User 3 also showed a preference Item 1. So the recommender system would make a recommendation of Item 2 to User 3.

This is similar to something seen on an e-commerce website that says "Other shoppers bought..."

To calculate the best recommendation for a certain user, the algorithm determines the similarity between the current user and all other users. One common way to calculate similarity between users is the Cosine Similarity. As computed by the Surprise Library, a python library for recommendation systems, it is defined as the similarity between a user u and other user v where $\boldsymbol{I}_{u,v}$ is a set of items both user u and user v rated.

$$\text{cosine\_sim}(u, v) = \frac{\sum\limits_{i \in \boldsymbol{I}_{u,v}} r_{ui} r_{vi}}{\sqrt{\sum\limits_{i \in \boldsymbol{I}_{u,v}} r_{ui}^2} \sqrt{\sum\limits_{i \in \boldsymbol{I}_{u,v}} r_{vi}^2}}$$

Where

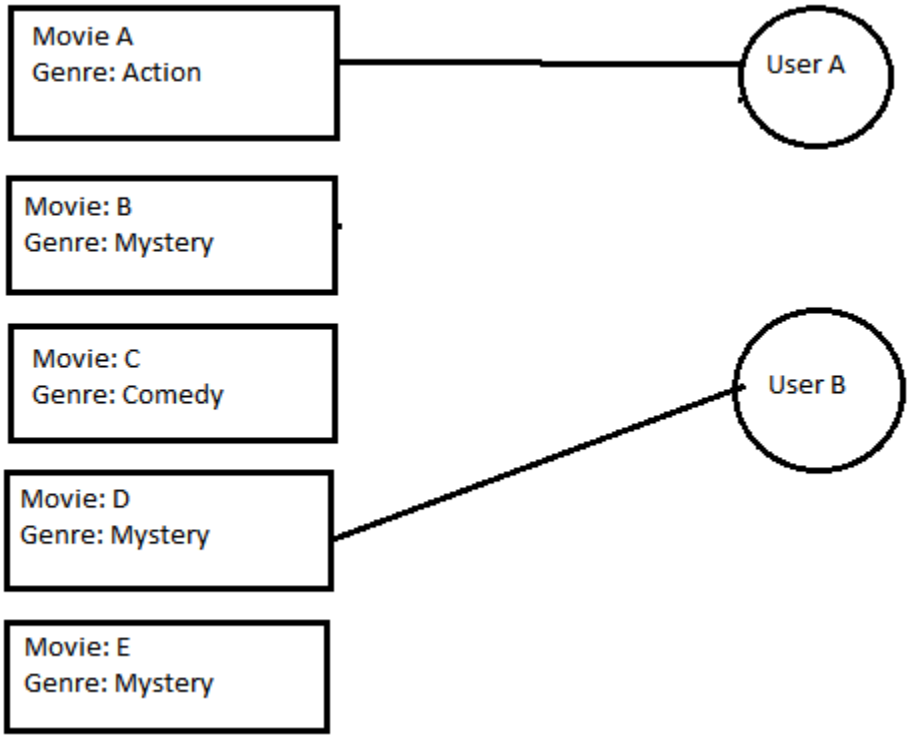
- $\text{cosine\_sim}(u, v)$ is the similarity between user u and user v,
- $\boldsymbol{I}_{u,v}$ is the set of indexes for items rated by both u and v, and
- r_{wi} is the rating of item i for user w (u or v in this example)

If the two users or items to compare are completely equal (have same user-item interactions), the cosine similarity function will return 1. Two users or items that are completely unalike and have nothing in common will return 0 for the similarity.

Based upon the most similar users, the recommendation system recommends items that the similar users also recommended. In the above example, since User 3 was most similar to User 1, the recommendation system would recommend items that User 1 liked.

Content-Based Filtering

Unlike collaborative filtering, content-based filtering makes predictions based on the similarity of each item. This is done using metadata about items, such as item descriptions. [5]



In this example, the content-based system would recommend items based on the descriptive information of each item, which is the movies genre.

User A interacted with or liked Movie A, which had the genre of Action. When providing future recommendations to User A, the system would recommend movies that have genres either of Action or similar to Action. This is because the recommender system places a high importance on relationships between items. It would not take into account anything User B liked.

For User B, the same logic would apply. Since User B interacted with a Mystery movie, the recommender system would recommend other mystery movies to the user. For example, it might recommend Movie B or Movie E, since they both have the same genre.
This is similar to what e-commerce sites would label "Similar Products...".

Hybrid Recommendation Systems

A hybrid recommendation system takes into account both user and item importance. One way to create a hybrid recommendation system is to first run a content-based recommender

system, then a collaborative recommender system, and combining or averaging the results[9]. Another approach is to combine them into one algorithm, taking into account both user and item preferences, with any and all item descriptors. The hybrid approach is not just limited to content-based or collaborative systems, it can combine any algorithm used for Recommender systems.

Netflix is a prime example of a hybrid recommendation system. Netflix not only relies on the genre of a movie (content-based), but also what movies were watched by other users (collaborative).

2.3 Approaches to Recommendation systems

Once a specific type of system is identified (either collaborative, content-based, or hybrid), an appropriate approach is then taken. The more traditional approach is a memory approach, where recommendations are made solely based on only the user or item similarities. A more modern approach is a model-based approach, where more information is provided to provide more meaningful recommendations [3].

Memory approach

"Memory based ... utilizes the entire user-item data to generate predictions"[3]. This uses only the explicit user-item data to make recommendations by calculating the similarity between each user and item. The memory approach uses statistical methods to determine the similarity between users. An example is K-Nearest-Neighbors. K-Nearest-Neighbors calculates the distance between all users in the dataset [3]. The model then chooses the most similar users, for example the top 5 or 10 most similar users. Then, the model determines what items are most commonly preferred among similar users, and makes a recommendation based upon the most commonly preferred item.

The advantages to this approach are that it is easy to implement (only the distance per each user and/or every item is calculated), and the results are easy to understand. The only hyper parameter to tune with K-Nearest-neighbors is the number of neighbors. Advanced knowledge is not needed to explain/build the system.

The disadvantages are that the memory approach's performance can be lower due to missing information about each user and each item. Also, when there are more users and more items introduced into the dataset, this can drastically increase the space complexity of the model (since distances between every user and/or every item must be stored).

Model-Based Approach

The model based approach, rather than computing similar users, first develops a model of the users ratings [19]. This is different than the Memory-Based approach because it takes more of a probabilistic approach to maximize the likelihood of the expected rating given the entire dataset. An example of an algorithm used for the Model-Based approach is a Bayesian Network Model. The Bayesian Network Model formulates a probabilistic model to maximize the likelihood of the output variable, the expected rating [19].

The model-based approach is more complicated to understand, and the results are harder to interpret. Instead of being able to explain that a user is similar to another user, the model-based approach weighs many factors, such as in the case of Netflix, their browsing habits, age, gender, information about each movie or TV show, and also how similar they are to other users.

An advantage to taking the model-based approach is that dimensionality reduction techniques can be used to reduce the space complexity of the algorithm. Techniques such as Principal Components Analysis can determine the most important components within the data, and ignore the rest. This drastically decreases the amount of data that needs to be parsed, which can decrease the amount of time needed to create a recommendation.

[3] found that on average, the model-based approach has a higher accuracy then the memory approach.

2.4 Scoring Algorithms

In order to quantify how well a model is performing, a scoring algorithm is used to evaluate the difference from the expected recommendation to the actual recommendation. Based upon

the distribution of the data, different scoring algorithms need to be used or the result may be misinterpreted.

A set of ratings, $\boldsymbol{R}$ is considered, which contains ratings, r_{ui} for user u and item i. The $\boldsymbol{R}$ might not contain a rating for every user-item pair. A recommendation model that predicts a rating for a certain user u and item i can be expressed as $\hat{r}_{ui}$. The error is the difference between the two:

$$\epsilon_{ui} = r_{ui} - \hat{r}_{ui}$$

The difference in each scoring algorithm is how high the penalty should be based upon the error term. The most common scoring algorithms for recommender systems are root mean squared error (RMSE) and mean absolute error (MAE). Another scoring algorithm, FCP, does not use the above error term but rather is calculated using fraction of concordant pairs.

RMSE: Root Mean Squared Error is calculated as the square root of the sum of errors squared. More formally, it is calculated [11] as:

$$\text{RMSE} = \sqrt{\frac{1}{|\boldsymbol{R}|} \sum_{r_{ui} \in \boldsymbol{R}} (r_{ui} - \hat{r}_{ui})^2}.$$

where $|\boldsymbol{R}|$ is the total number of ratings.

MAE: Mean Absolute Error is calculated as the arithmetic average of absolute errors. More formally, it is calculated as:

$$\text{MAE} = \frac{1}{|\boldsymbol{R}|} \sum_{r_{ui} \in \boldsymbol{R}} |r_{ui} - \hat{r}_{ui}|$$

where, again, $|\boldsymbol{R}|$ is the total number of ratings.

FCP: Fraction of Concordant Pairs: A common disadvantage of MAE and RMSE is that the scoring measures are not specifically designed for ordinal values. By computing averages, these measures assume that the spaces between ratings are equal. For example, the difference between a 5 and 4 star rating is not the same as the difference between a 2 and 1 star rating. Also, it does not take into account that different users can have different

rating scales. For example, user A rates on the full range from 1-5, while user B only rate items on a 3-4 range. [11]

FCP, which stands for Fraction of Concordant Pairs, is one proposed solution that determines what fraction of pairs are in the correct order, relative to each other, ignoring the specific values (ratings) provided.

Given a set of predicted ratings, $\hat{\boldsymbol{R}}$ and the original set of ratings, $\boldsymbol{R}$, [11] defined the number of concordant pairs for an individual user k as

$$n_c^u = |\{(i,j)|\hat{r}_{ui} > \hat{r}_{uj} \text{ and } r_{ui} > r_{uj}\}|$$
$$n_c = \sum_u n_c^u$$

where i, j are pairs of items that user u rated. The number of discordant pairs is calculated similarly,

$$n_d^u = |\{(i,j)|\hat{r}_{ui} \geq \hat{r}_{uj} \text{ and } r_{ui} < r_{uj}\}|$$
$$n_d = \sum_u n_d^u$$

This is calculated for for every user in the matrix, and FCP can then be calculated as:

$$FCP = \frac{n_c}{n_c + n_d}$$

FCP is better suited for recommendation algorithms, as it takes into account that data may not be nominal, that rating scales differ per user, and does not amplify outliers.

2.5 Open Source Software

Building a recommendation system can be easily accomplished using open-source software and libraries. This thesis utilized the Python programming language and libraries to aid with the usage of data, web scraping, and machine learning.

Python

Python is an interpreted, high-level and general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant indentation [14]. Python is often used for scientific purposes, as it is easy to read, and can be used alongside other languages, such as C, to optimize performance.The Python Package Index, PyPi, hosts thousands of modules to extend the usage of Python. They are not limited to scientific and numeric computing, but also encompass web and internet development, and database access.

Scikit-learn

Scikit-learn is a free machine learning library for the Python programming language. Scikit-learn is commonly used because it is simple and efficient, easily reusable, and open-source.

surprise

Surprise (Simple Python Recommendation System) is an open-source Scikit recommendation system library for python. Surprise mainly deals with explicit rating data, which is data that ratings are known before the model is built. The other type of data, implicit data, is gathered from users habits, such as time on the site, age, and other demographics.

This thesis uses surprise as the recommendation system library because it is commonly used, well documented, and gives control over the library. With Surprise, different recommendation algorithms can be used, their hyperparameters tuned, and evaluated using different scoring algorithms. Surprise also includes built-in datasets and custom-built datasets which are used to build and evaluate recommendation models. Surprise also encompasses most recommendation algorithms, which includes SVD (Singular Value Decomposition), KNN (K-Nearest-Neighbors), and many others. Also included in surprise are ways to evaluate model performance, which encompasses the scoring algorithms mentioned above. Using surprise, recommendation data can be loaded, a model built, and evaluated, all using one library.

Chapter 3

Mathematical Prior Work

In general, recommendation systems are machine learning algorithms that have been fine tuned to discover meaningful relationships between users and items. To do so using preferences, data must be provided to define the relationship between users and items. Machine Learning algorithms can then use this data to discover relationships and provide meaningful recommendations.

The recommendation data can be thought of as a set of ratings:

$$\boldsymbol{R} = \{r_{kl} | \text{ where user } k \text{ has rated item } l\}$$

Or, a hypothetical ratings matrix:

$$\boldsymbol{R} = \begin{bmatrix} r_{11} & r_{12} & \dots & r_{1q} \\ r_{21} & r_{22} & \dots & r_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ r_{p1} & r_{p2} & \dots & r_{pq} \end{bmatrix}$$

where p is the number of users, q is the number of items, and many of the entries are missing because each user only rates a small fraction of the item. We define $|\boldsymbol{R}|$ as the number of ratings (not missing) in the matrix. More specifically, for each user item preference, the ratings matrix may be similar to the following matrix, which describes how much users liked a movie on a scale of 1-5

	Avengers	Interstellar	Jurassic Park
Matt	1	5	3
Sarah	2	3	3
John	4	1	4

Table 3.1: Example ratings matrix

This shows that Matt (user) rated the Avengers (item) 5 stars (preference). Mathematically, relationships between users and items are explored to form meaningful recommendations. To form a recommendation for a user (in this example, person), the recommendation computes predicted preferences (in this example, movie rating) for each item (in this example, movie). A recommendation is then made based upon the highest preferences.

These same data could be represented as a data matrix and target vector:

$$\boldsymbol{X} = \begin{bmatrix} k^{(1)} & l^{(1)} \\ \vdots & \vdots \\ k^{(n)} & l^{(n)} \end{bmatrix} = \begin{bmatrix} — & \boldsymbol{x}^{(1)\top} & — \\ & \vdots & \\ — & \boldsymbol{x}^{(n)\top} & — \end{bmatrix} \quad \boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

where the first column contains the user indexes, the second column contains the item indexes and the target vector contains the ratings. That is, $y^{(i)} = r_{k^{(i)}l^{(i)}}$. The previous example might look like this:

$$\boldsymbol{X} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 2 & 1 \\ 2 & 2 \\ 2 & 3 \\ 3 & 1 \\ 3 & 2 \\ 3 & 3 \end{bmatrix} \quad \boldsymbol{y} = \begin{bmatrix} 1 \\ 5 \\ 3 \\ 2 \\ 3 \\ 3 \\ 4 \\ 1 \\ 4 \end{bmatrix}$$

One way to create a recommendation system is to model y as a function of the user and item in $\boldsymbol{x}$, $\hat{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x})$. Models differ in the way they calculate predicted ratings (preferences) and in their parameters, $\boldsymbol{\theta}$. One way to estimate the parameters is to derive cost functions that

can distinguish between better and worse choices for the parameters.

3.1 Maximum Likelihood Estimation

Maximum likelihood estimation selects the parameters for a model that maximize the probability of observing the output given the parameters. This is accomplished by including parameters, $\boldsymbol{\theta}$, which defines the relationship between the input and output variables. The Likelihood function takes on the following form:

$$\mathcal{L}(\boldsymbol{\theta}) = P(\boldsymbol{y}|\boldsymbol{\theta})$$

The parameters are estimated as the following:

$$\hat{\boldsymbol{\theta}}_{\text{MLE}} = \underset{\boldsymbol{\theta}}{\text{argmax}}\, P(\boldsymbol{y}|\boldsymbol{\theta})$$

This is read as "Select the $\boldsymbol{\theta}$ variables that maximize the likelihood of the output variable, $\boldsymbol{y}$, given the parameters $\boldsymbol{\theta}$". Since samples are independent and identically distributed, the distribution of the data set is the product of the distribution for each sample. This is computed for every individual y variable, so this equation is rewritten as

$$\hat{\boldsymbol{\theta}}_{\text{MLE}} = \underset{\boldsymbol{\theta}}{\text{argmax}} \prod_{i=1}^{n} P(y^{(i)}|\boldsymbol{\theta})$$

$P(y^{(i)}|\boldsymbol{\theta})$ differs based upon the distribution of the the correct values of $y^{(i)}$ given the model, and this thesis covers the normal distribution and Poisson distribution. To further reduce this equation, the log of the probability is taken. The cost function is defined as the negative log likelihood:

$$J_{\text{MLE}} = -\sum_{i=1}^{n} \log P(y^{(i)}|\boldsymbol{\theta})$$

3.2 Maximum a Posteriori (MAP) Estimation

As opposed to MLE which treats the parameters as fixed but unknown, maximum a posterior (MAP) estimation treats the parameters as random variables with a prior distribution. The probability of the $\boldsymbol{\theta}$ parameters given the ouputs $\boldsymbol{y}$ is the following:

$$P(\boldsymbol{\theta}|\boldsymbol{y}) = \frac{P(\boldsymbol{y}|\boldsymbol{\theta})P(\boldsymbol{\theta})}{P(\boldsymbol{y})}$$

This equation can be described as [10]:

$$\text{Posterior} = \frac{\text{likelihood} * \text{prior}}{\text{evidence}}$$

Since $P(\boldsymbol{y})$ is not dependent upon $\boldsymbol{\theta}$, it can be removed from the maximization:

$$\begin{aligned}
\hat{\boldsymbol{\theta}}_{MAP} &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, P(\boldsymbol{y}|\boldsymbol{\theta})P(\boldsymbol{\theta}) \\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \left(\prod_{i=1}^{n} P(y^{(i)}|\boldsymbol{\theta}) \right) P(\boldsymbol{\theta}) \\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \log \left(\prod_{i=1}^{n} P(y^{(i)}|\boldsymbol{\theta}) \right) P(\boldsymbol{\theta}) \\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{i=1}^{n} \log P(y^{(i)}|\boldsymbol{\theta}) + \log P(\boldsymbol{\theta})
\end{aligned}$$

The cost function for the MAP estimate can be seen as a regularized version of the MLE cost. It includes the prior distribution of $\boldsymbol{\theta}$ to penalize the selection of parameter values that are not likely:

$$J_{MAP} = -\sum_{i=1}^{n} \log P(y^{(i)}|\boldsymbol{\theta}) - \log P(\boldsymbol{\theta}) \tag{3.1}$$

The types of distributions considered in this thesis include Gaussian and Poisson for the likelihood, $P_y(\boldsymbol{y}|\boldsymbol{\theta})$, and the Gaussian distribution for $P(\boldsymbol{\theta})$. The recommendation system library utilized in later chapters only utilizes a normal distribution, but others such as the Laplace

distribution could be considered in future work.

3.3 Likelihood Functions

Gaussian Distributed Likelihood

When the likelihood function, $P(\boldsymbol{y}|\boldsymbol{\theta})$, follows a Gaussian distribution, the result is least squares regression. Least squares attempts to minimize the sum of squared errors between the model's predictions and expected output. For example, linear regression uses $\hat{y} = b + \boldsymbol{w}^\top \boldsymbol{x}$. A single sample, $\hat{y}^{(i)}$ from the $\hat{\boldsymbol{y}}$ vector can then be modeled as the following:

$$y^{(i)} = \hat{y}^{(i)} + \epsilon^{(i)}$$

Where $\epsilon^{(i)}$ is random noise that follows a Gaussian distribution, denoted by $\epsilon \sim \mathcal{N}(0, \sigma^2)$. Since ϵ follows a normal distribution, it also means that y follows a normal distribution of $y \sim \mathcal{N}(\hat{y}, \sigma^2)$. The likelihood function is then modeled as a normal distribution based on the output variables, or

$$\begin{aligned}
J &= -\sum_{i=1}^{n} \log P(y^{(i)}|\boldsymbol{\theta}) \\
&= -\sum_{i=1}^{n} \log \mathcal{N}(y^{(i)}; \hat{y}^{(i)}, \sigma^2) \\
&= -\sum_{i=1}^{n} \log \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{\left(y^{(i)} - \hat{y}^{(i)}\right)^2}{2\sigma^2}} \\
&= -\sum_{i=1}^{n} \left(\log \frac{1}{\sigma\sqrt{2\pi}} - \frac{\left(y^{(i)} - \hat{y}^{(i)}\right)^2}{2\sigma^2} \right)
\end{aligned}$$

Ignoring additive terms that do not depend on $\hat{y}$, we have a scaled version of the sum of squared errors:

$$J_{\text{SSE}} = \frac{1}{2\sigma^2} \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)} \right)^2$$

Note that σ is not dependent upon $\hat{y}$ but it will be useful to define the regularization coefficient in a later section.

Poisson Distributed Likelihood

The Poisson distribution has a single parameter λ, where λ is both the mean and variance. The Poisson distribution is often used to describe observed counts of events given a time period. For example, the number of births per hour, or the number of visitors of a website in a given hour. For example, Poisson regression models $\hat{y} = e^{b+\boldsymbol{x}^\top \boldsymbol{x}}$. More generally, when the likelihood function follows a Poisson distribution:

$$\hat{y} = \lambda,$$

and the likelihood is constructed:

$$\begin{aligned} P(\boldsymbol{\theta}|\boldsymbol{y}) &= \prod_{i=1}^{n} P(y^{(i)}|\boldsymbol{\theta}) \\ &= \prod_{i=1}^{n} \text{POIS}(\hat{y}^{(i)}) \\ &= \prod_{i=1}^{n} \frac{\lambda^{(i)y^{(i)}} e^{-\lambda^{(i)}}}{y^{(i)}!} \end{aligned}$$

The cost function can be derived as the negative log likelihood ignoring additive terms that do not depend on $\hat{y}$:

$$\begin{aligned} J_{\text{POIS}} &= -\log \prod_{i=1}^{n} \frac{\left(\lambda^{(i)}\right)^{y^{(i)}} e^{-\lambda^{(i)}}}{y^{(i)}!} \\ &= \sum_{i=1}^{n} \lambda^{(i)} - y^{(i)} \log(\lambda^{(i)}) \end{aligned}$$

Again, consult Appendix A for the mathematical proof.

3.4 Priors

Gaussian Prior

We model the parameters as independent and identically distributed Gaussian random variables:

$$\theta_j \sim \mathcal{N}(0, \tau^2)$$

The prior distribution for all parameters, then, is the following:

$$\begin{aligned} P(\boldsymbol{\theta}) &= \prod_{j=1}^{m} \mathcal{N}(\theta_j; 0, \tau^2) \\ &= \prod_{j=1}^{m} \frac{1}{\tau\sqrt{2\pi}} e^{-\frac{{\theta_j}^2}{2\tau^2}} \end{aligned}$$

The cost function, again, is the negative log of this probability ignoring additive terms that do not depend on θ:

$$\begin{aligned} J_{L_2} &= -\log P(\boldsymbol{\theta}) \\ &= \frac{1}{2\tau^2} \sum_{j=1}^{m} {\theta_j}^2 \end{aligned}$$

Summary

To build a cost function for the MAP estimate, a cost based on the likelihood and a penalty based on the prior are combined into one equation. For example, if the likelihood function follows a Gaussian distribution and the prior is Gaussian distributed, we get the cost function for Ridge Regression: $J_{RIDGE} = J_{SSE} + J_{L_2}$:

$$J_{RIDGE} = \frac{1}{2\sigma^2} \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)} \right)^2 + \frac{1}{2\tau^2} \sum_{j=1}^{m} {\theta_j}^2$$

If the likelihood function instead was a Poisson Distributed likelihood and the prior was a Normal prior $J_{PL2} = J_{\text{POIS}} + J_{L_2}$

$$J_{PL2} = \sum_{i=1}^{n} \left(\lambda^{(i)} - y^{(i)} \log(\lambda^{(i)}) \right) + \frac{1}{2\tau^2} \sum_{j=1}^{m} {\theta_j}^2$$

3.5 Regularization

Models are susceptible to over-fitting when the model is too complex or there is too little data. In these cases, the model is well trained for the training data, but not well trained in general, and for future use. This can lead to a high performance on the training data, but bad performance on the testing and production level data sets. Using a prior on the parameters imposes a preference for simpler models, known as regularization.

Recalling from above, an equivalent form of the Ridge Regression cost function looks like this after scaling by $2\sigma^2$:

$$J_{RIDGE} = \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)} \right)^2 + \frac{\sigma^2}{\tau^2} \sum_{j=1}^{m} {\theta_j}^2$$

This equation can be further simplified in terms of regularization. Instead of stating $\frac{\sigma^2}{\tau^2}$, which is the ratio between the output variance and the parameter variance, this is defined as λ. Note that this is a different variable and concept than the λ with Poisson regression, giving us a final form of

$$J_{RIDGE} = \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)} \right)^2 + \lambda \sum_{j=1}^{m} \theta_j^2 \tag{3.2}$$

Here λ indicates the "regularization strength," that is, how much we prefer simpler models.

3.6 Gradient Descent

Gradient descent is a general algorithm for adapting parameters to reduce the cost. The gradient of a function provides the direction of steepest ascent, where the goal is to take small steps "down the gradient," to find a local minimum. The gradient can be computed by taking the

partial derivative of the cost function with respect to each parameter, or more generally:

$$\frac{\partial J}{\partial \boldsymbol{\theta}} = \begin{bmatrix} \frac{\partial J}{\partial \theta_1} \\ \vdots \\ \frac{\partial J}{\partial \theta_m} \end{bmatrix}$$

Gradient descent can then be performed by taking small steps of size η away from the gradient, or

$$\theta_j \leftarrow \theta_j - \eta \frac{\partial J}{\partial \theta_j}$$

This can be read as the next value for θ can be calculated as the previous value minus some learning rate, η, times the derivative of the cost function. As the learning rate, η, increases, larger steps are taken. This can lead to overshooting the solution, which can lead away from the solution. As η decreases, smaller steps are taken, which can take more time to reach the local minimum.

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is a type of Gradient Descent that instead of computing the gradient with the entire data set, updates the parameters with one sample at a time. The cost function can be written as the sum of the cost for each observation, or

$$J = \sum_{i=1}^{n} J^{(i)}$$

The gradient of J can be written as the sum of the gradient for each sample:

$$\frac{\partial J}{\partial \theta_j} = \sum_{i=1}^{n} \frac{\partial J^{(i)}}{\partial \theta_j}$$

Instead of computing the gradient of J, SGD computes it on one sample i, updating the parameters each time:

$$\theta_j \leftarrow \theta_j - \tau \frac{\partial J^{(i)}}{\partial \theta_j}$$

Ridge Regression with Stochastic Gradient Descent

To apply Stochastic Gradient Descent to linear regression, we first much write the cost function as a sum of single-sample cost functions. Recall that for linear regression $\hat{y} = \boldsymbol{w}^\top \boldsymbol{x}$:

$$J = \frac{1}{2\sigma^2} \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)} \right)^2 + \frac{1}{2\tau^2} \sum_{j=1}^{m} {w_j}^2$$

Or, equivalently by scaling by σ^2:

$$\begin{aligned} J &= \frac{1}{2} \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)} \right)^2 + \frac{\sigma^2}{2\tau^2} \sum_{j=1}^{m} {w_j}^2 \\ &= \sum_{i=1}^{n} \left(\frac{1}{2} \left(y^{(i)} - \hat{y}^{(i)} \right)^2 + \frac{\sigma^2}{2n\tau^2} \sum_{j=1}^{m} {w_j}^2 \right) \\ J^{(i)} &= \frac{1}{2} \left(y^{(i)} - \hat{y}^{(i)} \right)^2 + \frac{\sigma^2}{2n\tau^2} \sum_{j=1}^{m} {w_j}^2 \end{aligned}$$

The partial derivatives of the single-sample cost are the following:

$$\begin{aligned} \frac{\partial J^{(i)}}{\partial b} &= - \left(y^{(i)} - \hat{y}^{(i)} \right) \\ \frac{\partial J^{(i)}}{\partial w_j} &= - \left(y^{(i)} - \hat{y}^{(i)} \right) x_j^{(i)} + \frac{\sigma^2}{n\tau^2} w_j \end{aligned}$$

Using $\lambda = \sigma^2/(n\tau^2)$ for the regularization coefficient, the update rules are the following:

$$\begin{aligned} b &\leftarrow b + \eta \left(y^{(i)} - \hat{y}^{(i)} \right) \\ w_j &\leftarrow w_j + \eta \left((y^{(i)} - \hat{y}^{(i)}) x_j^{(i)} - \lambda w_j \right) \end{aligned}$$

Poisson Regression with Regularization using Stochastic Gradient Descent

To apply gradient descent to Poisson regression, the derivative of the cost function is taken. The same general rule as above is used but instead, $\hat{y}$ is calculated differently. For clarity, the regularization parameter λ is renamed to λ_{REG}. Recall that Poisson regression models

$\hat{y} = e^{b+\boldsymbol{w}^\top \boldsymbol{x}^{(i)}}$.

$$J = \sum_{i=1}^{n} \left(\lambda^{(i)} - y^{(i)} \log(\lambda^{(i)}) \right) + \frac{1}{2\tau^2} \sum_{j=1}^{m} {w_j}^2 \tag{3.3}$$

$$= \sum_{i=1}^{n} \left(\lambda^{(i)} - y^{(i)} \log(\lambda^{(i)}) + \frac{1}{2n\tau^2} \sum_{j=1}^{m} {w_j}^2 \right) \tag{3.4}$$

$$J^{(i)} = \lambda^{(i)} - y^{(i)} \log(\lambda^{(i)}) + \frac{1}{2n\tau^2} \sum_{j=1}^{m} {w_j}^2 \tag{3.5}$$

The partial derivatives are the following:

$$\frac{\partial J^{(i)}}{\partial b} = \lambda^{(i)} - y^{(i)} \tag{3.6}$$

$$\frac{\partial J^{(i)}}{\partial w_j} = (\lambda^{(i)} - y^{(i)}) x_j^{(i)} + \frac{1}{n\tau^2} w_j \tag{3.7}$$

Using $\lambda_{\text{REG}} = 1/(n\tau^2)$, the updates are:

$$b \leftarrow b + \eta \left(y^{(i)} - \lambda^{(i)} \right) \tag{3.8}$$

$$w_j \leftarrow w_j + \eta \left((y^{(i)} - \lambda^{(i)}) x_j^{(i)} - \lambda_{REG} w_j \right) \tag{3.9}$$

Notice that these updates are equivalent to the updates for ridge regression except with $\hat{y}^{(i)}$ replaced by $\lambda^{(i)}$.

3.7 Recommendation Models

A recommendation model can be defined by an equation for the ratings, $\hat{r}_{kl}$, the probability distribution for r, and the prior on the parameters. First we define a baseline model:

$$\hat{r}_{kl} = \mu + u_k + v_l,$$

where μ is the mean item rating, u_k is the bias for user k, and v_l is the bias for item l. μ can be calculated as the mean rating over the entire data set. The the vectors containing the user

($\boldsymbol{u}$) and item ($\boldsymbol{v}$) biases are calculated using gradient descent.

To determine how each variable is updated, the distribution of the data is defined.

$$\begin{aligned} P(\boldsymbol{r}|\boldsymbol{u},\boldsymbol{v}) &= \prod_{r_{kl}\in\boldsymbol{R}} \mathcal{N}(r_{kl};\hat{r}_{kl},\sigma^2) \\ &= \prod_{r_{kl}\in\boldsymbol{R}} \mathcal{N}(r_{kl};\hat{r}_{kl},\sigma^2) \end{aligned}$$

This means that the output variable, given a user k and item l, follows a normal distribution with mean $\mu + u_k + v_l$. The cost function, J can be generalized as the sum of squared errors:

$$\begin{aligned} J &= \sum_{r_{kl}\in\boldsymbol{R}} \left(\frac{1}{2}\left(r_{kl}-\hat{r}_{kl}\right)^2 + \frac{\lambda}{2}\sum_i u_i^2 + \frac{\lambda}{2}\sum_j v_j^2 \right) \\ J^{(kl)} &= \frac{1}{2}\left(r_{kl}-\hat{r}_{kl}\right)^2 + \frac{\lambda}{2}\sum_i u_i^2 + \frac{\lambda}{2}\sum_j v_j^2 \end{aligned}$$

with partial derivatives:

$$\begin{aligned} \frac{\partial J^{(kl)}}{\partial \mu} &= -(r_{kl}-\hat{r}_{kl}) \\ \frac{\partial J^{(kl)}}{\partial u_k} &= -(r_{kl}-\hat{r}_{kl}) + \lambda u_k \\ \frac{\partial J^{(kl)}}{\partial v_l} &= -(r_{kl}-\hat{r}_{kl}) + \lambda v_l \end{aligned}$$

and update rules:

$$\begin{aligned} \mu &\leftarrow \mu + \eta\left(r_{kl}-\hat{r}_{kl}\right) \\ u_k &\leftarrow u_k + \eta\left(r_{kl}-\hat{r}_{kl}-\lambda u_k\right) \\ v_l &\leftarrow v_l + \eta\left(r_{kl}-\hat{r}_{kl}-\lambda v_l\right) \end{aligned}$$

Having a bias per user and bias per item vector does not properly express the relationships per each user-item. To better explain the relationships in the data, matrix factorization techniques, such as Singular Value Decomposition, are utilized.

Gaussian Matrix Factorization

Singular Value Decomposition is a Linear Algebra technique to reduce the explained deviance in data to a smaller dimension. The SVD algorithm rose to popularity for recommendation when Simon Funk utilized it for a Netflix prize challenge to see what collaborative filtering algorithm is best to predict ratings for future films.

Recommendation Models use similar matrix factorization technique to SVD by representing each user and item as a vector. The dot-product between a user and an item indicates the preference of a user for an item:

$$\hat{r}_{kl} = \mu + u_k + v_l + \boldsymbol{p}_k^\top \boldsymbol{q}_l,$$

where

- $\boldsymbol{p}_k$: The vector describing user k,
- $\boldsymbol{q}_l$: The vector describing item l,
- u_k: The bias for user k, and
- v_l The bias for item l.

The length of the user/item vectors is a hyper-parameter (number of factors) that are tuned to capture the most amount of knowledge to approximate the original matrix. For example, in a user-movie data set, the number of factors is used to effectively capture what "type" of a movie a given user might like. If there was only 1 factor, the factor gauges one range of genre a user might like or dislike. For example, the one factor could be used to capture if a user preferred horror or comedy. Then, as another factor is added, another axis of preference is added. For example, if 2 factors were used, the first factor might encapsulate horror versus comedy while the next vector encapsulates fantasy versus history.

When there are a low number of factors, this results in a compressed version of the matrix, with losses. As the number of factors increase, the original matrix can be better represented, but it requires more storage and computation.

The recommendation algorithm attempts to find latent dimensions that minimizes the sum-squared distance to the target matrix, $\boldsymbol{R}$ [18]. The $\boldsymbol{q}$ vector is the latent representation for an item, the $\boldsymbol{p}$ vector is the latent representation for a user.

The extent to which a user interacts with (whether positive or negative) is based upon the $\boldsymbol{q}$ and $\boldsymbol{p}$ vectors. If the result of the $\boldsymbol{q}^T\boldsymbol{p}$ is near 0, there is little to no interaction between that user and item. In that case, the prediction is solely based upon the baseline measures, which are the μ, u, and v which are per each user and item. But if the result of $\boldsymbol{q}^\top\boldsymbol{p}$ is a positive number, there is a positive interaction between that user and item, resulting in a higher rating. If the result of $\boldsymbol{q}^\top\boldsymbol{p}$ points is a negative number, there is a negative interaction between that user and item, resulting in a lower rating. The recommendation algorithm aims to find the most optimal vectors to maximum the likelihood of the output variable [12].

Cost Function and updates for Gaussian Matrix Factorization

A $\boldsymbol{y}$ value in SVD including user biases, item biases, and a mean rating can be calculated as

$$\hat{\boldsymbol{R}} = \boldsymbol{P}\boldsymbol{Q}^\top + \boldsymbol{u} + \boldsymbol{v}^T + \mu$$

$$\hat{r}_{kl} = {\boldsymbol{p}_k}^T\boldsymbol{q}_l + u_k + v_l + \mu$$

The cost can then be modeled as the Sum of Squared Errors.

$$J = \frac{1}{2}\sum_{r_{kl}\in\boldsymbol{R}}\left((r_{kl}-\hat{r}_{kl})^2 + \lambda u_k^2 + \lambda v_l^2 + \lambda\|\boldsymbol{p}_k\|^2 + \lambda\|\boldsymbol{q}_l\|^2\right)$$

$$J^{(kl)} = \frac{1}{2}\left((r_{kl}-\hat{r}_{kl})^2 + \lambda u_k^2 + \lambda v_l^2 + \lambda\|\boldsymbol{p}_k\|^2 + \lambda\|\boldsymbol{q}_l\|^2\right),$$

with partial derivatives:

$$\begin{aligned}
\frac{\partial J^{(kl)}}{\partial \mu} &= -(r_{kl} - \hat{r}_{kl}) \\
\frac{\partial J^{(kl)}}{\partial u_k} &= -(r_{kl} - \hat{r}_{kl}) + \lambda u_k \\
\frac{\partial J^{(kl)}}{\partial v_l} &= -\left(r_{kl} - \hat{r}_{kl}\right) + \lambda v_l \\
\frac{\partial J^{(kl)}}{\partial \boldsymbol{p}_k} &= -\left(r_{kl} - \hat{r}_{kl}\right) \boldsymbol{q}_l + \lambda \boldsymbol{p}_k \\
\frac{\partial J^{(kl)}}{\partial \boldsymbol{q}_l} &= -\left(r_{kl} - \hat{r}_{kl}\right) \boldsymbol{p}_k + \lambda \boldsymbol{q}_l
\end{aligned}$$

with update rules:

$$\begin{aligned}
\mu &\leftarrow \mu + \eta \left(r_{kl} - \hat{r}_{kl}\right) \\
u_k &\leftarrow u_k + \eta \left(r_{kl} - \hat{r}_{kl} - \lambda u_k\right) \\
v_l &\leftarrow v_l + \eta \left(r_{kl} - \hat{r}_{kl} - \lambda v_l\right) \\
\boldsymbol{p}_k &\leftarrow \boldsymbol{p}_k + \eta \left((r_{kl} - \hat{r}_{kl}) \boldsymbol{q}_l - \lambda \boldsymbol{p}_k\right) \\
\boldsymbol{q}_l &\leftarrow \boldsymbol{q}_l + \eta \left((r_{kl} - \hat{r}_{kl}) \boldsymbol{p}_k - \lambda \boldsymbol{q}_l\right)
\end{aligned}$$

Poisson Matrix Factorization

If the expected ratings were instead estimated using a Poisson Distribution,

$$\begin{aligned}
\hat{\boldsymbol{R}} &= e^{\boldsymbol{P}\boldsymbol{Q}^\top + \boldsymbol{u} + \boldsymbol{v}^\top + \mu} \\
\hat{r}_{kl} &= e^{\boldsymbol{p}_k^\top \boldsymbol{q}_l + u_k + v_l + \mu}
\end{aligned}$$

The cost function is then defined as

$$\begin{aligned}
J &= \sum_{r_{kl} \in R} \hat{r}_{kl} - r_{kl} \log(\hat{r}_{kl}) + \frac{\lambda}{2} \left({u_k}^2 + {v_l}^2 + ||\boldsymbol{p}_k||^2 + ||\boldsymbol{q}_l||^2\right) \\
J^{(kl)} &= \hat{r}_{kl} - r_{kl} \log(\hat{r}_{kl}) + \frac{\lambda}{2} \left({u_k}^2 + {v_l}^2 + ||\boldsymbol{p}_k||^2 + ||\boldsymbol{q}_l||^2\right),
\end{aligned}$$

with partial derivatives:

$$\begin{aligned}
\frac{\partial J^{(kl)}}{\partial \mu} &= \hat{r}_{kl} - r_{kl} \\
\frac{\partial J^{(kl)}}{\partial u_k} &= \hat{r}_{kl} - r_{kl} + \lambda u_k \\
\frac{\partial J^{(kl)}}{\partial v_l} &= \hat{r}_{kl} - r_{kl} + \lambda v_u \\
\frac{\partial J^{(kl)}}{\partial \boldsymbol{q}_l} &= (\hat{r}_{kl} - r_{kl})\, \boldsymbol{p}_k + \lambda \boldsymbol{q}_l \\
\frac{\partial J^{(kl)}}{\partial \boldsymbol{p}_k} &= (\hat{r}_{kl} - r_{kl})\, \boldsymbol{q}_l + \lambda \boldsymbol{p}_k
\end{aligned}$$

with updates:

$$\begin{aligned}
\mu &\leftarrow \mu + \eta\,(r_{kl} - \hat{r}_{kl}) \\
u_k &\leftarrow u_k + \eta\,(r_{kl} - \hat{r}_{kl} - \lambda u_k) \\
v_l &\leftarrow v_l + \eta\,(r_{kl} - \hat{r}_{kl} - \lambda v_l) \\
\boldsymbol{p}_k &\leftarrow \boldsymbol{p}_k + \eta\,((r_{kl} - \hat{r}_{kl})\boldsymbol{q}_l - \lambda \boldsymbol{p}_k) \\
\boldsymbol{q}_l &\leftarrow \boldsymbol{q}_l + \eta\,((r_{kl} - \hat{r}_{kl})\boldsymbol{p}_k - \lambda \boldsymbol{q}_l)
\end{aligned}$$

Note that these are the same updates as Poisson Regression and Gaussian Matrix Factorization. The only thing that changes is how $\hat{r}$ is calculated.

Chapter 4

Data

This thesis attempts to build a recommendation system for TV shows. Data was collected from Reddit, an online community that averages 430 million monthly active users [6]. Data from TV show SubReddits was collected to form a user-item matrix. In the context of recommendation systems, the users were Redditors, and the items were their interaction with TV show SubReddits. The interaction, or expected rating was measured by how many comments a given user had left on the SubReddit.

4.1 Web Scraping

Web scraping is the process of extracting data from websites. Web scraping can be accomplished manually by navigating to websites and copy/pasting the data. This process can also be automated, by navigating websites using a "robot," which is a computing tool to automatically "crawl" a website and download information. Web scraping is often times used to get the most up to date stock prices, or to get recent posts from a website.

This thesis scraped data using the Reddit API, or Application Programmer Interface, which provides an easy way to retrieve data from a website. A programmer can place queries to the website and receive a structured response. The programmer interacts directly with the website API, which is faster than manually parsing the website.

4.2 Scraping Reddit Data

Reddit

Reddit is an online community where users network in communities called SubReddits, make posts, and interact with other users. Users, known as Redditors, can create posts, upvote (like) posts, downvote (dislike) posts, and make comments.

PRAW

PRAW (Python Reddit API Wrapper) is an open-source Python Package that allows simple access to Reddit's API. Once a API key is generated through Reddit's API website, PRAW can be used to query information from Reddit, such as retrieve comments from a SubReddit. To download 25 comments from the "Community" SubReddit, the following code would be used:

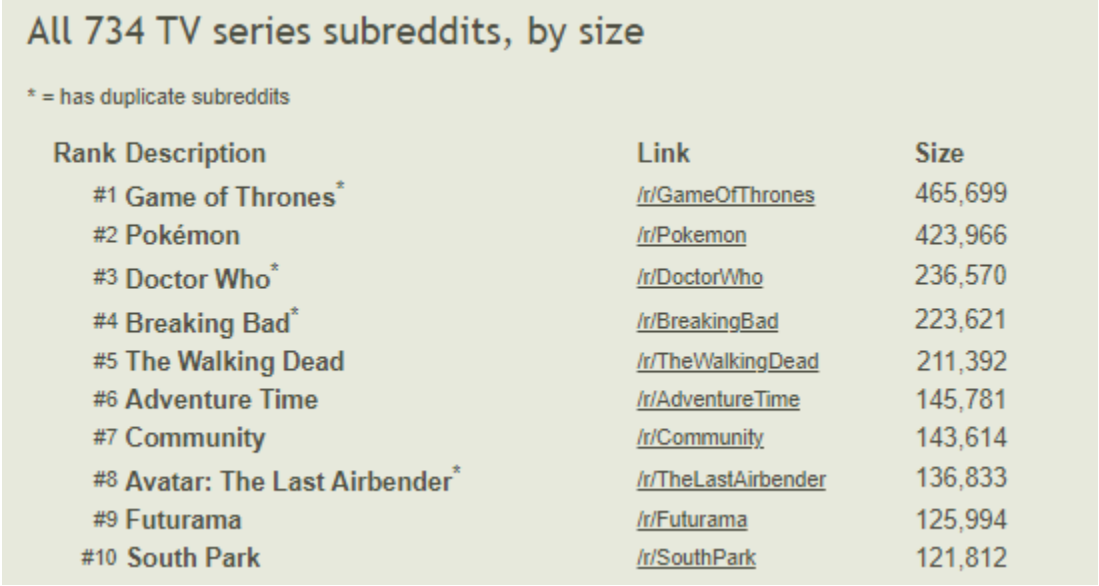
```
1   import praw
2
3   reddit = praw.Reddit(
4       user_agent="Comment Extraction (by u/USERNAME)",
5       client_id="CLIENT_ID",
6       client_secret="CLIENT_SECRET",
7       username="USERNAME",
8       password="PASSWORD",
9   )
10
11  for comment in reddit.subreddit("Community").comments(limit=25):
12      print(comment.author)
```

4.3 Determining suitable SubReddits to scrape

SubReddits are similar to a blog, or a group on Facebook. A SubReddit has a central topic, such as "News," or "U.S. Politics." SubReddits can take on a wide array of topics, but this thesis focuses specifically on SubReddits based upon TV shows. Some examples are "Brooklyn99" or "Community." A list of TV Show SubReddits was queried from wikidot, which included 743 total TV SubReddits [1]:

id	author	Subreddit	created	score
gu7m8qw	Stuped1811	adventuretime	1618193380.0	1
gu7m1uv	Stuped1811	adventuretime	1618193276.0	1
gu7lhi4	LOL3334444	adventuretime	1618192978.0	2
gu7lcs5	Stuped1811	adventuretime	1618192906.0	1
gu7kpxv	hunnyb33˙	adventuretime	1618192557.0	1
gu7klu2	Pap8r-Mango	adventuretime	1618192494.0	1

Figure 4.1: Example: /r/adventuretime



All 734 TV series subreddits, by size

* = has duplicate subreddits

Rank	Description	Link	Size
#1	Game of Thrones*	/r/GameOfThrones	465,699
#2	Pokémon	/r/Pokemon	423,966
#3	Doctor Who*	/r/DoctorWho	236,570
#4	Breaking Bad*	/r/BreakingBad	223,621
#5	The Walking Dead	/r/TheWalkingDead	211,392
#6	Adventure Time	/r/AdventureTime	145,781
#7	Community	/r/Community	143,614
#8	Avatar: The Last Airbender*	/r/TheLastAirbender	136,833
#9	Futurama	/r/Futurama	125,994
#10	South Park	/r/SouthPark	121,812

A simple script was run to check that all of the SubReddits existed and were up to date. Some issues found with SubReddits included: they were either deleted or banned, they were not based on TV shows, or a more up-to-date SubReddit existed. Once all entries were evaluated and corrected, there were 687 total suitable SubReddits to scrape data from.

For each SubReddit, a query was placed to retrieve a maximum of 1000 comments, sorted by date created in descending order (Figure 4.1) with the following definitions:

- **id**: ID of the submission.
- **author**: The Redditor's username.
- **SubReddit**: Fullname of the SubReddit.
- **created**: Time the submission was created, represented in Unix Time.
- **score**: The number of upvotes for the comment.

The data was then saved to a CSV file. After each request, the program waited 2 seconds to obey Reddit's robot.txt file (a textual form of how long a robot must wait to parse a given website).

Merging results into triple format

For every SubReddit CSV file, the data was aggregated for each user-item pair. After the data was aggregated, two separate data files were created:

Master The master file contained the following rows:

id	author	created	Subreddit	NumComments
gtaa7yd	platinumgoddess˙12	1617485422.0	12ozmouse	27
gt1bflu	Benji1819	1617288587.0	12ozmouse	4
gsw9pmu	StingrayOC	1617179929.0	12ozmouse	20
gst8gx2	Ocramtan	1617119312.0	12ozmouse	1
gsnj9j6	MBTHVSK	1616991179.0	12ozmouse	2
gsnj6ct	MBTHVSK	1616991117.0	12ozmouse	2
gsc4afm	shooterboss	1616791694.0	12ozmouse	7
gsbnto0	Viewbob¨true	1616785413.0	12ozmouse	3
t8kka9	lucius42	1617451436.0	Babylon5	3

Triple The triple file contains the following rows:

author	Subreddit	NumComments
c1daley	1600penn	1
Marb˙Reds	1600penn	3
jsh1138	1600penn	3
Clayburn	1600penn	1
zatch17	1600penn	2

The triple file, which will be used for the recommendation system, included:

- 183,133 total rows
- 155,324 total unique users
- 677 total SubReddits
- Comments created from 2010-2021
- The number of comments range from 1 to 270, with a mean of 2.51

Filtered dataset

The above dataset shows that there was a large number of users commenting on only one SubReddit which can lead to a badly performing model, as there is possibly low interaction

between users and SubReddits. To form a dataset with more user-SubReddit interaction, two subsets of the dataset were created. One subset of the dataset was formed to include only rows that had a minimum interaction of 2 SubReddits and SubReddits with a minimum of 2 Redditors commenting. The same was repeated for subsets of 4 and 8.

The subset of data with minimum 2 interaction included the following:

- 45624 total rows
- 17836 total unique users
- 630 total SubReddits
- The number of comments range from 1 to 270, with a mean of 3.23

The subset of data with minimum 4 interaction included:

- 10537 total rows
- 1887 total unique users
- 519 total SubReddits
- The number of comments range from 1 to 270, with a mean of 3.9966

The subset of data with minimum 8 interaction included:

- 1100 total rows
- 199 total unique users
- 155 total SubReddits
- The number of comments range from 1 to 207, with a mean of 3.70

Only including Redditors with high interaction and SubReddits with a high number of comments should produce more accurate results.

Chapter 5

Surprise Library Recommendation Algorithms

5.1 Recommendation Algorithm

The Surprise library provides the following recommendation algorithms:

Basic Algorithms

The Basic Algorithms package contains two simple models to be used as baseline measures.

Constant The Constant algorithm predicts the mean rating based upon the distribution of the training set (assumed to be a normal distribution).

The constant algorithm predicts ratings as:.

$$\hat{r}_{kl} = \mu$$

Where μ is the mean rating of the entire dataset. The data is expected to follow a normal distribution, $\mathcal{N}(\mu, \sigma)$.

Baseline The baseline algorithm is considered the most basic algorithm that learns from each item and user.

The baseline algorithm is considered the most basic algorithm that learns from each item and user. The baseline algorithm predicts ratings as:

$$\hat{r}_{kl} = \mu + u_k + v_l$$

If the user is unknown, then u_k is set to 0. If the item is unknown, v_l is also set to 0.

5.2 Gaussian Matrix Factorization

In the Surprise library, the SVD algorithm makes a prediction based upon a normal distribution of the ratings. The biased SVD algorithm predicts ratings as:

$$\hat{r}_{kl} = \mu + u_k + v_l + \boldsymbol{q}_l^T \boldsymbol{p}_k$$

An unbiased SVD algorithm does not take into account user and item bias, and predicts ratings as:

$$\hat{r}_{kl} = \boldsymbol{q}_l^T \boldsymbol{p}_k$$

An unbiased SVD algorithm is known as Probabilistic Matrix Factorization. When used in conjunction with other models such as Restricted Boltzmann Machines models, Probabilistic Matrix Factorization performed on average 7% better than Netflix's own recommendation engine[18].

Chapter 6

Proposed Changes

To build a more accurate recommendation model, this thesis proposes changing the Surprise Library to include a Poisson Distribution.

6.1 Making changes to the Recommendation Library

The existing Surprise SVD algorithm assumes that the data is normally distributed. (Figure 6.1) shows the distribution of the Reddit data, which follows a Poisson distribution.

A Poisson Factorization approach for recommendation systems has been proposed but is not part of the Surprise library [8]. Similar techniques were used to introduce Poisson Factorization to the Surprise Library.

The Surprise code repository was cloned from GitHub, and changes were made to the matrix_factorization.pyx file. A pyx file is a cython file, which integrates features of C with Python.

To introduce the choice of having a regular cost versus Poisson cost, the init function was changed to include a cost parameter.

Yo make predictions, Surprise calculates $\hat{r}$, or the expected rating, as

```
1         r_hat = z = (global_mean + bu[u] + bi[i] + dot)
```

As discussed in earlier chapters, calculating a rating from a Poisson Distributed SVD can be calculated as

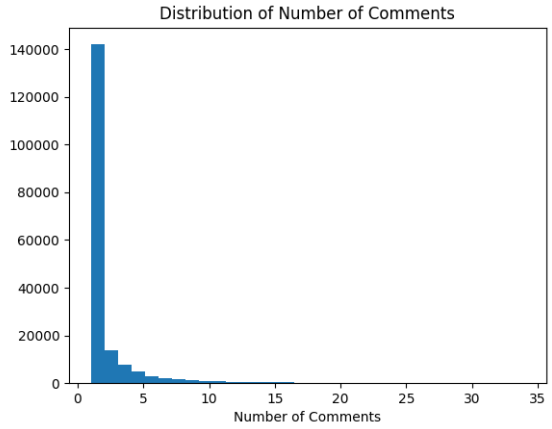


Figure 6.1: Distribution of Reddit data

$$\hat{r}_{kl} = e^{\mu + u_k + v_l + \boldsymbol{q}_l^T \boldsymbol{p}_k}$$

To introduce these changes to the Surprise Python library, $\hat{r}$ was calculated as

```
1        r_hat = np.exp(z)
```

. The error term is then calculated as

```
1        err = r - r_hat
```

As noted in Chapter 3, the update rules are the same.

Adding Cost per Epoch

Most machine learning libraries include the functionality to print, or display, the cost. Including this enables a programmer to see if the cost is decreasing over time, and at what rate it is decreasing. This thesis proposes adding this functionality to Surprise.

For normally distributed data, the cost is calculated as the sum of squared errors. For a Poisson distribution, the Poisson cost is used, which was calculated in earlier chapters. The Poisson cost does not converge to 0, it converges to a minimum of `r*(np.log(r)-1)`.

Every epoch, the overall cost is calculated as

$$cost = \frac{\text{regcost} + \text{objcost}}{\text{number of ratings}}$$

```
1 if self.cost == 'Poisson':
2     obj_cost += r*(np.log(r)-1) - (r*z - r_hat)
3 else:
4     obj_cost += err ** 2
```

Figure 6.2: Computing the cost

This takes into account the cost from the model and the cost from the weights. This was divided by the number of ratings because as the number of samples increase, the regularization on the weights should decrease.

Adding this to the Surprise library allows programmers to monitor how their model is performing over time.

Chapter 7

Testing

In order to show the viability of the framework presented in this thesis, a number of tests were performed on the framework. This section details the methodology used in those tests and their results.

7.1 SVD

To ensure that the surprise SVD algorithm was performing as expected, data was generated to follow how Surprise calculates a rating:

$$\hat{r}_{kl} = \mu + u_k + v_l + \boldsymbol{q}_l^T \boldsymbol{p}_k$$

To simulate the data, the following variables were modified:

- standard deviation of weights
- noise
- number of components

To evaluate the algorithm, the expected cost was calculated before training and after fitting/predicting. The cost was calculated based upon the distribution of the data.

The cost after fitting the model was expected to be equal to the cost calculated before training.

Listing 7.1: The `Testing` python file.

```
import pandas as pd
import numpy.random as n
import numpy as np
from surprise import SVD
from surprise import Dataset
from surprise import Reader
from surprise import accuracy
from surprise.model_selection import train_test_split
from matplotlib import pyplot as plt
import random
import numpy as np

random.seed(1)
np.random.seed(1)

def svd_sim(std_noise, std_weights, numComponents, mu, cost):
    numUsers = 100
    numItems = 100
    p = np.random.normal(0, std_weights, size=(numUsers, numComponents))
    q = np.random.normal(0, std_weights, size=(numItems, numComponents))
    bu = np.random.normal(0, std_weights, size=(numUsers, 1))
    bi = np.random.normal(0, std_weights, size=(numItems, 1))

    if cost == "poisson":
        r_hat_ui = np.exp(mu + bu + bi.T + np.dot(p, q.T))
        r_ui = np.random.poisson(lam=r_hat_ui).astype("float")
        poiss_cost = np.mean(r_ui*(np.log(r_ui + 1e-6)-1) - (r_ui*np.log(r_hat_ui)
            - r_hat_ui))
        print(f"{poiss_cost}", end = ",")
    elif cost == "normal":
        r_hat_ui = mu + bu + bi.T + np.dot(p, q.T)
        r_ui = np.random.normal(loc=r_hat_ui, scale=std_noise)
        rmse = np.mean((r_hat_ui - r_ui) ** 2)**0.5
        print(f"{rmse}", end = ",")

```

```
35      # making the ratings into a matrix
36      i, j = np.meshgrid(range(numUsers),range(numItems), indexing='ij')
37      r_ui = r_ui.reshape((numUsers*numItems, 1))
38      i = i.reshape((numUsers*numItems, 1))
39      j = j.reshape((numUsers*numItems, 1))
40      matrix = np.concatenate([i, j, r_ui], axis=1)
41      df = pd.DataFrame(matrix, columns=['author', 'subreddit', 'NumComments'])
42      reader = Reader(rating_scale=(1e-6, float("inf")))
43      data = Dataset.load_from_df(df, reader)
44      trainset = data.build_full_trainset()
45      testset = trainset.build_testset()
46      algo = SVD(n_epochs=500, cost=cost, verbose=False, random_state=1, lr_all
            =0.001, n_factors=numComponents,
47              reg_all=std_noise ** 2 / (std_weights ** 2 * trainset.n_ratings))
48      algo.fit(trainset)
49      predictions = algo.test(testset)
50
51      r_ui = np.array([p[2] for p in predictions])
52      r_hat_ui = np.array([p[3] for p in predictions])
53      if cost == "poisson":
54          poiss_cost = np.mean(r_ui * (np.log(r_ui + 1e-6) - 1) - (r_ui * np.log(
                r_hat_ui + 1e-6) - r_hat_ui))
55          print(f"{poiss_cost}")
56      elif cost == "normal":
57          rmse = np.mean((r_hat_ui - r_ui) ** 2) ** 0.5
58          print(f"{rmse}")
59
60
61  print("normal,0.0,0.5,0,5,",end="")
62  svd_sim(0.0,0.5,0,5,"normal")
63  print("normal,0.0,0.5,10,5,",end="")
64  svd_sim(0.0,0.5,10,5,"normal")
65  print("normal,0.5,0.5,0,5,",end="")
66  svd_sim(0.5,0.5,0,5,"normal")
67  print("normal,0.5,0.5,10,5,",end="")
68  svd_sim(0.5,0.5,10,5,"normal")
```

```
print("poisson,1,0.5,0,0,",end="")
svd_sim(1,0.5,0,0,"poisson")
print("poisson,1,0.5,10,0,",end="")
svd_sim(1,0.5,10,0,"poisson")
```

The results were as follows:

	stdnoise	stdweights	numcomp	mu	expectedcost	actualcost
normal	0.0	0.5	10	5	0.0	0.0055
normal	0.5	0.5	0	5	0.4989	0.4939
normal	0.5	0.5	10	5	0.4995	0.4424
poisson	1	0.5	0	0	0.5314	0.5226
poisson	1	0.5	10	0	0.5137	0.4005

The expected cost and actual cost are nearly identical for all cases. As more components are used, the actual cost is expected to be less accurate as there are more variables for noise and error to be present. As more noise is added to the model, the error is expected to be higher.

Chapter 8

Results

To assess the results of the algorithms, a grid search was run (if the algorithm had hyper parameters). A grid search is used to determine the best combination of hyper parameters to achieve the highest score. For the purpose of this thesis, the highest FCP score will be used to determine the best hyper parameters.

The parameters taken into consideration for the grid search were

- Learning rate: [0.01, 0.001, 0.0001, 0.00001]
- Regularization rate: [1.0, 0.1, 0.01, 0.001, 0]
- Number of factors: [100, 75, 50, 25, 10, 5, 2, 1, 0]
- Number of epochs: [100, 300, 500]
- Biased: [True, False]

There are more hyper parameters that Surprise includes, but for the purpose of this thesis, only the above hyper parameters are evaluated. If a more exhaustive grid search was completed, the final recommendation model might perform more highly.

The grid search was completed on training data, which included 70% of the original data. 5 fold cross-validation was performed, which splits the data into 5 equal partitions. For each partition, the model is fit with the other 4 partitions, and evaluated using the remaining partition.

Once the best combination of hyper parameters were found (based on highest FCP score), the algorithm was evaluated on the test set. The test set included 30% of the data, held out before training. When evaluating the model, the following metrics were used: FCP, MAE, and RMSE. (Chi-Squared and Poisson Deviance were also calculated, but not included in final results).

Listing 8.1: The `Grid Search` and `Test Performance` python file.

```
from surprise import SVD, Dataset, Reader, NormalPredictor, BaselineOnly, accuracy
from surprise.model_selection import GridSearchCV, KFold
from sklearn.model_selection import train_test_split
import pandas as pd
import random
import numpy as np

# setting random seeds
np.random.seed(1)
random.seed(1)

df = pd.read_csv('triple_all_comments.csv')

# This limits the data to authors who have commented in at least 'n' subreddits and
# subreddits with at least 'n' commenters.
min_comments = 0
df = df.set_index('author').loc[df['author'].value_counts() >= min_comments].
    reset_index()
df = df.set_index('Subreddit').loc[df['Subreddit'].value_counts() >= min_comments].
    reset_index()
print(len(df))


trainset, testset = train_test_split(df, test_size=.30, random_state=1)
reader = Reader(rating_scale=(1e-6, float("inf")))

# this line converts the data into the required format for testing
# it is still the same dimensions and same data
test_data = Dataset.load_from_df(testset, reader).build_full_trainset().
    build_testset()
# test_data= t_data.build_full_trainset().build_testset()

# builds a trainset from our data
train_data = Dataset.load_from_df(trainset, reader).build_full_trainset()
# train_data = tr_data.build_full_trainset()


# builds train data for surprise.model_selection.GridSearchCV
data = Dataset.load_from_df(trainset, reader)

param_grid = {'n_epochs': [500, 300, 100],
              'lr_all': [0.01, 0.001, 0.0001, 0.00001],
              'reg_all': [1.0, 0.1, 0.01, 0.001, 0],
              'n_factors': [100, 75, 50, 25, 10, 5, 2, 1, 0],
              'random_state': [1],
              'biased': [False, True]}

```

```
cv = KFold(n_splits=5, random_state=1)
gs = {}
for cost in ['normal', 'poisson']:
    param_grid.update({'cost': [cost]})
    gs[cost] = GridSearchCV(SVD, param_grid, measures=['rmse', 'mae',  'fcp', 'chi2', 'pois'], cv=cv, n_jobs=-1, joblib_verbose=10)
    gs[cost].fit(data)

    # best RMSE score
    print(cost)
    print("FCP")
    print(gs[cost].best_score['fcp'])
    print(gs[cost].best_params['fcp'])
    print("MAE")
    print(gs[cost].best_score['mae'])
    print(gs[cost].best_params['mae'])
    print("RMSE")
    print(gs[cost].best_score['rmse'])
    print(gs[cost].best_params['rmse'])
    print("CHI2")
    print(gs[cost].best_score['chi2'])
    print(gs[cost].best_params['chi2'])
    print("POIS")
    print(gs[cost].best_score['pois'])
    print(gs[cost].best_params['pois'])

    df = pd.DataFrame(data=gs[cost].cv_results)
    df.to_csv(f'cv_results_{cost}_min_comments_{min_comments}.csv', index=False)

find_best(["normal"])
find_best(["poisson"])

constant = NormalPredictor()
baseline = BaselineOnly()
svd = gs['normal'].best_estimator['fcp']
poisson = gs['poisson'].best_estimator['fcp']
results = []
for model, name in zip([constant, baseline, svd, poisson], ['constant', 'baseline', 'svd', 'poisson']):
    model.fit(train_data)
    predictions = model.test(test_data)
    fcp = accuracy.fcp(predictions, verbose=True)
    rmse = accuracy.rmse(predictions, verbose=True)
    mae = accuracy.mae(predictions, verbose=True)
    # The Pearson chi-square statistic:
    # https://en.wikipedia.org/wiki/Pearson%27s_chi-squared_test#Calculating_the_test-statistic
    # pearson = sum((pred.r_ui - pred.est)**2/pred.est for pred in predictions)
    # The Poisson deviance:
    # https://en.wikipedia.org/wiki/Deviance_(statistics)#Examples
    # poisson_deviance = 2 * sum(pred.r_ui * np.log(pred.r_ui/pred.est) - (pred.r_ui - pred.est)
    #                            for pred in predictions)
    chi2 = accuracy.chi2(predictions, verbose=True)
    pois = accuracy.pois(predictions, verbose=True)
    d = {
        'model': name,
        'fcp': fcp,
        'rmse': rmse,
```

```
100         'mae': mae,
101         'chi2': chi2,
102         'pois': pois,
103     }
104     results.append(d)
105
106 df = pd.DataFrame(data=results)
107 df.to_csv(f'test_results_min_comments_{min_comments}.csv', index=False)
```

8.1 Filtered performance of 2 interactions

As discussed above, having users and SubReddits with low interactions can lead to a badly performing model. One alternative this thesis proposes is to limit the dataset to only include users who have interacted with at least 2 SubReddits and SubReddits with at least 2 users interacting with them.

Constant Algorithm

Training and evaluating the Constant Algorithm on the subset of data yielded the following results:

	Score
FCP	0.4549
RMSE	8.4684
MAE	4.9921

Table 8.1: Results of filtered test set performance for Constant Algorithm

The FCP score for the constant algorithm was 0.4549, which we will use to compare future model performance to.

Baseline Algorithm

The baseline algorithm also does not utilize hyper parameters. Evaluating the Baseline Algorithm on the testset yielded the following results:

The FCP score of 0.5489 was higher than the Constant Algorithm.

	Score
FCP	0.5489
RMSE	6.5363
MAE	2.6836

Table 8.2: Results of filtered test set performance for Baseline Predictor

Gaussian Matrix Factorization

The SVD algorithm within Surprise utilizes hyper parameters, so a grid search was completed for the normally distributed SVD algorithm. The results are as follows:

	score	epochs	lr	reg	nfactors	biased
FCP	0.5151	100	0.01	0	0	True
RMSE	6.4586	100	0.001	1.0	100	True
MAE	2.6030	500	0.001	0.1	100	True

Table 8.3: Results of grid search for SVD

To evaluate how the recommendation system was performing, the data was then tested on the test set. The test set includes 30% of the data held out before training. The results were as follows:

	Score
FCP	0.5496
RMSE	7.0264
MAE	2.9825

Table 8.4: Results of test set performance for SVD algorithm

The test set results are similar to the baseline model, but the FCP of 0.5496 was slightly higher than the baseline model.

Poisson Matrix Factorization

A grid search was completed for the Poisson Matrix Factorization. The results were as follows:

The results of how well the best FCP model performed on the test set are as follows:

The FCP score for the PMF is slightly higher than the Baseline and SVD algorithm.

	score	epochs	lr	reg	nfactors	biased
FCP	0.5196	500	0.001	0	0	True
RMSE	6.5634	100	0.001	0.1	25	True
MAE	2.4064	500	0.001	0	25	False

Table 8.5: Results of grid search for SVD with Poisson Cost

	Score
FCP	0.5591
RMSE	9.2643
MAE	2.9770

Table 8.6: Results of test set performance for PMF

Comparing the results of all algorithms,

	FCP	RMSE	MAE
Constant	0.4549	8.4684	4.9921
Baseline	0.5489	6.5363	2.6836
SVD	0.5496	7.0264	2.9825
PMF	0.5591	9.2643	2.9770

Table 8.7: Results of all models

Comparing results, the PMF algorithm, according to the FCP measure, is performing the best of the four. The same steps were repeated, but for the subset of data that included the 4 minimum instead.

8.2 Filtered performance of 4 interactions

As discussed above, having users and SubReddits with low interactions can lead to a badly performing model. One alternative this thesis proposes is to limit the dataset to only include users who have interacted with at least 4 SubReddits and SubReddits with at least 4 users interacting with them.

The same models were evaluated but instead of the full dataset, only the subset of data that only included authors who have commented on 4 SubReddits and SubReddits with at least 4 commenters was used. The same 70 % 30% train and test set splitting technique was used.

Constant Algorithm

Training and evaluating the Constant Algorithm on the subset of data yielded the following results:

	Score
FCP	0.4826
RMSE	11.9241
MAE	6.3980

Table 8.8: Results of filtered test set performance for Constant Algorithm

The FCP score for the constant algorithm was 0.482619, which we will use to compare future model performance to.

Baseline Algorithm

The baseline algorithm also does not utilize hyper parameters. Evaluating the Baseline Algorithm on the testset yielded the following results:

	Score
FCP	0.5982
RMSE	9.5770
MAE	3.4836

Table 8.9: Results of filtered test set performance for Baseline Predictor

The FCP score of 0.5982 was higher than the Constant Algorithm.

Gaussian Matrix Factorization

The SVD algorithm within Surprise utilizes hyper parameters, so a grid search was completed for the normally distributed SVD algorithm. The results are as follows:

	score	epochs	lr	reg	nfactors	biased
FCP	0.5597	500	0.00001	0.1	2	True
RMSE	7.8922	300	0.001	1.0	0	True
MAE	3.3526	500	0.001	0.1	75	True

Table 8.10: Results of grid search for SVD

To evaluate how the recommendation system was performing, the data was then tested on the test set. The test set includes 30% of the data held out before training. The results were as follows:

	Score
FCP	0.5815
RMSE	9.8606
MAE	3.6845

Table 8.11: Results of test set performance for SVD algorithm

The test set results are similar to the baseline model, but the FCP of 0.581456 was slightly lower than the baseline model.

Poisson Matrix Factorization

A grid search was completed for the Poisson Matrix Factorization. The results were as follows:

	score	epochs	lr	reg	nfactors	biased
FCP	0.5716	100	0.001	1.0	0	True
RMSE	6.3890	500	0.001	0.1	10	True
MAE	2.4637	100	0.001	0	1	False

Table 8.12: Results of grid search for SVD with Poisson Cost

The results of how well the best FCP model performed on the test set are as follows:

	Score
FCP	0.5818
RMSE	9.6486
MAE	3.7101

Table 8.13: Results of test set performance for PMF

The FCP score for the PMF is slightly lower than the Baseline algorithm but slightly higher than the SVD algorithm.

Comparing the results of all algorithms,

	FCP	RMSE	MAE
Constant	0.4826	11.9241	6.3980
Baseline	0.5982	9.5770	3.4836
SVD	0.5815	9.8606	3.6845
PMF	0.5818	9.6486	3.7101

Table 8.14: Results of all models

Comparing results, the Baseline Algorithm, according to the FCP measure, is performing the best of the four.

The same steps were repeated, but for the subset of data that included the 8 minimum instead.

8.3 Filtered performance of 8 interactions

The same models were evaluated but instead of the full dataset, only the subset of data that only included authors who have commented on 8 SubReddits and SubReddits with at least 8 commenters was used.

Constant Algorithm

Training and evaluating the Constant Algorithm on the subset of data yielded the following results:

	Score
FCP	0.4185
RMSE	14.0497
MAE	5.8809

Table 8.15: Results of filtered test set performance for Constant Algorithm

Baseline Algorithm

The baseline algorithm also does not utilize hyper parameters. Evaluating the Baseline Algorithm on the testset yielded the following results:

The FCP score was higher than the Constant model.

	Score
FCP	0.6518
RMSE	12.6021
MAE	3.6221

Table 8.16: Results of filtered test set performance for Baseline Predictor

Gaussian Matrix Factorization

The SVD algorithm within Surprise utilizes hyper parameters, so a grid search was completed for the normally distributed SVD algorithm. The results are as follows:

	score	epochs	lr	reg	nfactors	biased
FCP	0.5650	500	0.01	1.0	75	False
RMSE	6.3082	100	0.01	0	75	True
MAE	2.7632	500	0.01	0.1	75	True

Table 8.17: Results of grid search for SVD

To evaluate how the recommendation system was performing, the data was then tested on the test set. The test set includes 30% of the data held out before training. The results were as follows:

	Score
FCP	0.5379
RMSE	12.7955
MAE	3.4222

Table 8.18: Results of test set performance for SVD algorithm

Poisson Matrix Factorization

A grid search was completed for the Poisson Matrix Factorization. The results were as follows:

The results of how well the best FCP model performed on the test set are as follows:

The FCP score for the Baseline Algorithm is the best-performing of all models.

	score	epochs	lr	reg	nfactors	biased
FCP	0.5716	100	0.001	1.0	0	True
RMSE	6.5890	500	0.001	0.1	10	True
MAE	2.4637	100	0.001	0	1	False

Table 8.19: Results of grid search for SVD with Poisson Cost

	Score
FCP	0.6234
RMSE	12.8825
MAE	4.9405

Table 8.20: Results of test set performance for PMF

Comparing the results of all algorithms,

	FCP	RMSE	MAE
Constant	0.4185	14.0947	5.8809
Baseline	0.6513	12.6020	3.6221
SVD	0.5379	12.7955	3.4222
PMF	0.6234	12.8825	4.9405

Table 8.21: Results of all models

Comparing results, the Constant algorithm, according to the FCP measure, was performing the best. But the PMF algorithm, according to the FCP score, is outperforming the SVD algorithm, This shows that using SVD with a Poisson distribution, or PMF, improved the accuracy of the recommendation system for the filtered dataset.

Chapter 9

Conclusion

9.1 Conclusion

Within this thesis, 4 algorithms for Recommendation Systems were built and evaluated using 3 different datasets. For the subset of data with 2 interactions, the best performing model according to FCP was PMF, with a score of 0.5591. For the subset of data with 4 interactions, the best performing model according to FCP was the Baseline model, with a score of 0.5982. This is already an increase of score compared to the subset of data with 2 interactions. For the subset of data with 8 interactions, the best performing model according to FCP was the Baseline algorithm, with a score of 0.6513. These results suggest that narrowing data to only include the most active Redditors and SubReddits increases the accuracy of the Reddit Recommendation models.

The results suggest that the best model, according to FCP, was the Baseline algorithm for the subset of data with 8 interactions. The highest score for Poisson Matrix Factorization, was also for the subset of data with 8 interactions. These results suggest that using a subset of data with 8 interactions results in Recommendation Algorithms that perform the best on test data.

9.2 Future Work

Future work may include adding an early stopping mechanism. This means that the algorithm will stop learning if the cost is less than some tolerance to save time and combat over fitting. This is widely utilized within advanced Python libraries such as Scikit-Learn. Adding this to the Surprise Python Library, can decrease the amount of time needed to train a model.

In the Mathematical Prior Work Chapter, only Normal Priors were covered. Another popular prior, Laplace or L_1, is not included within the Surprise Library. Adding an option for the users to choose the type of regularization increases the functionality of the library, and is common within other Python libraries.

The Poisson Deviance and Chi-Squared statistic that was proposed can also be included in future Surprise changes. These were added to the Surprise library but the statistics still need to be fine-tuned for recommendation system purposes. Adding a proper, functional Poisson Deviance and Chi-Squared statistic would help programmers understand their model and how it is performing.

Other work may include using other types of distributions. This requires determining how a rating is calculated, determining the update rules, and allowing a user to specify which distribution their data follows.

9.3 Summary

This thesis proposed changes to the Surprise Python Library to take into account that data follows a Poisson distribution. Making these changes helped increase the accuracy of predictions for the Reddit TV Show dataset, and also proved that valuable recommendations can be made for Reddit TV Show SubReddits. The thesis also tested the functionality of the Surprise library normally distributed SVD and Poisson distributed SVD. This demonstrated that the existing framework was performing as expected, and that the changes to the framework were both accurate and did not impact the underlying algorithm.

Bibliography

[1] List of tv show subreddits. http://tv-SubReddits.wikidot.com/.

[2] Poisson regression. https://online.stat.psu.edu/stat501/lesson/15/15.4.

[3] P. H. Aditya, I. Budi, and Q. Munajat. A comparative analysis of memory-based and model-based collaborative filtering on the implementation of recommender system for e-commerce in Indonesia: A case study pt x. In *2016 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, pages 303–308, 2016.

[4] Justin Basilico and Thomas Hofmann. Unifying collaborative and content-based filtering. In *Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, page 9. Association for Computing Machinery, 2004.

[5] Poonam B.Thorat, R. Goudar, and Sunita Barve. Survey on collaborative filtering, content-based filtering and hybrid recommendation system. *International Journal of Computer Applications*, 110:31–36, 01 2015.

[6] Brian Dean. Reddit usage and growth statistics: How many people use reddit in 2021? https://backlinko.com/reddit-users.

[7] CARLOS A. GOMEZ-URIBE and NEIL HUNT. The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems*, 6(13):13–19, 2015.

[8] Prem Gopalan, Jake M. Hofman, and David M. Blei. Scalable recommendation with poisson factorization. *CoRR*, abs/1311.1704, 2013.

[9] Umair Javed, Kamran Shaukat, Ibrahim A. Hameed, Farhat Iqbal, Talha Mahboob Alam, and Suhuai Luo. A review of content-based and context-based recommendation systems. *International Journal of Emerging Technologies in Learning*, 16(03):25–26, 2021.

[10] Brian Keng. A probabilistic interprentation of regularization. https://bjlkeng.github.io/posts/probabilistic-interpretation-of-regularization/, 2016.

[11] Y. Koren and J. Sill. Collaborative filtering on ordinal user feedback. In *IJCAI*, 2013.

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

[13] Netflix. Netflix prize. https://www.netflixprize.com/.

[14] Python. About python. https://www.python.org/about/apps/.

[15] Elaine Rich. User modeling via stereotypes. *COGNITIVE SCIENCE*, 3:329–354, 1979.

[16] Tim Roughgarden and Gregory Valiant. Cs168: The modern algorithmic toolbox lecture# 9: The singular value decomposition (svd) and low-rank matrix approximations. *http://theory. stanford. edu/˜ tim/s15/l/l9. pdf*, 2015.

[17] Sebastian Ruder. An overview of gradient descent optimization algorithms. *Insight Centre for Data Analytics*, 2016.

[18] Ruslan Salakhutdinov and Andriy Mnih. Probabilistic matrix factorization. *NIPS*, page 1257–1264, 2007.

[19] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, page 285–295. Association for Computing Machinery, 2001.

[20] Duke University. Poisson regression and model checking. `https://www2.stat.duke.edu/courses/Fall17/sta521/knitr/Lec-9-Poisson-Checks/predictive-checking.pdf`, 2017.

[21] Wessel N. van Wieringen. Lecture notes on ridge regression. *arXiv e-prints*, sep 2015.

Appendices

Appendix A

Mathematical derivations

During the thesis there were several derivations that were summarized to reduce redundancies. The equations are fully worked out below

A.1 Derivation of Cost Function

Linear Regression

Given that

$$\hat{y}^{(i)} = \boldsymbol{w}^T \boldsymbol{x}^{(i)} + b$$

For the Maximum Likelihood Estimate,

$$\begin{aligned}
\hat{b},\, \hat{\boldsymbol{w}} &= \mathcal{L}(b,\, \boldsymbol{w}|\mathrm{y}) \\
&= \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{\left(y^{(i)} - \hat{y}^{(i)}\right)^2}{2\sigma^2}} \\
&= \log\left(\prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{\left(y^{(i)} - \hat{y}^{(i)}\right)^2}{2\sigma^2}}\right) \\
&= \sum_{i=1}^{n} \log(\frac{1}{\sigma\sqrt{2\pi}}) + \log(e^{-\frac{\left(y^{(i)} - \hat{y}^{(i)}\right)^2}{2\sigma^2}}) \\
&= \sum_{i=1}^{n} \log(e^{-\frac{\left(y^{(i)} - \hat{y}^{(i)}\right)^2}{2\sigma^2}}) \\
&= \sum_{i=1}^{n} -\frac{\left(y^{(i)} - \hat{y}^{(i)}\right)^2}{2\sigma^2}
\end{aligned}$$

Instead of maximizing $-\left(y^{(i)} - \hat{y}^{(i)}\right)^2$ we can instead minimize $\left(y^{(i)} - \hat{y}^{(i)}\right)^2$, which further simplifies to

$$= \sum_{i=1}^{n} \frac{\left(y^{(i)} - \hat{y}^{(i)}\right)^2}{2\sigma^2} \tag{A.1}$$

$$J_{SSE} = \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y^{(i)} - \hat{y}^{(i)})^2 \tag{A.2}$$

Poisson Regression

$$\begin{aligned}
z^{(i)} &= \boldsymbol{w}^T \boldsymbol{x}^{(i)} + b \\
z^{(i)} &= \log(\lambda^{(i)}) \\
x^{(i)} &= e^{z^{(i)}}
\end{aligned}$$

$$
\begin{aligned}
&= \prod_{i=1}^{n} \frac{\left(\lambda^{(i)}\right)^{y^{(i)}} e^{-\lambda^{(i)}}}{y^{(i)}!} \\
&= \log\left(\prod_{i=1}^{n} \frac{\left(\lambda^{(i)}\right)^{y^{(i)}} e^{-\lambda^{(i)}}}{y^{(i)}!}\right) \\
&= \sum_{i=1}^{n} z^{(i)} y^{(i)} - \lambda^{(i)} - \log(y^{(i)}!) \\
&= \sum_{i=1}^{n} z^{(i)} y^{(i)} - \lambda^{(i)} \\
J_{Poisson} &= \sum_{i=1}^{n} \lambda^{(i)} - y^{(i)} \log \lambda^{(i)}
\end{aligned}
$$

A.2 Derivations for Priors

Normal

$$
\begin{aligned}
J_{L_2} &= -\log P(\boldsymbol{\theta}) \\
&= -\log\left(\prod_{j=1}^{p} \frac{1}{\tau\sqrt{2\pi}} e^{-\frac{\left(\boldsymbol{\theta}_j\right)^2}{2\tau^2}}\right) \\
&= -\sum_{j=1}^{p} \log\left(\frac{1}{\tau\sqrt{2\pi}}\right) + \log\left(e^{-\frac{\left(\boldsymbol{\theta}_j\right)^2}{2\tau^2}}\right) \\
&= -\sum_{j=1}^{p} \log\left(e^{-\frac{\left(\boldsymbol{\theta}_j\right)^2}{2\tau^2}}\right)
\end{aligned}
$$

$$
J_{L_2} = \frac{1}{2\tau^2} \sum_{j=1}^{p} \boldsymbol{\theta}_j^2 \tag{A.3}
$$

A.3 Posterior

$$
\begin{aligned}
J &= J_{SSE} + J_{L_2} \\
&= \frac{1}{2\sigma^2}\left(\sum_{i=1}^{n} - \left(y^{(i)} - \hat{y}^{(i)}\right)^2 + \frac{2\sigma^2}{2\tau^2}\sum_{j=1}^{p}\boldsymbol{\theta}_j^2\right) \\
&= \frac{1}{2\sigma^2}\left(\sum_{i=1}^{n} - \left(y^{(i)} - \hat{y}^{(i)}\right)^2 + \frac{\sigma^2}{\tau^2}\sum_{j=1}^{p}\boldsymbol{\theta}_j^2\right) \\
&= \sum_{i=1}^{n} - \left(y^{(i)} - \hat{y}^{(i)}\right)^2 + \frac{\sigma^2}{\tau^2}\sum_{j=1}^{p}\boldsymbol{\theta}_j^2
\end{aligned}
$$

Therefore we can formally define

$$J = \sum_{i=1}^{n} - \left(y^{(i)} - \hat{y}^{(i)}\right)^2 + \frac{\sigma^2}{\tau^2}\sum_{j=1}^{p}\boldsymbol{\theta}_j^2 \tag{A.4}$$

$$J^{(i)} = - \left(y^{(i)} - \hat{y}^{(i)}\right)^2 + \frac{\sigma^2}{\tau^2}\sum_{j=1}^{p}\boldsymbol{\theta}_j^2 \tag{A.5}$$

$$\tag{A.6}$$

Instead of maximizing the negative sums, we can instead minimize the positive sums

$$= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)}\right)^2 + \frac{\sigma^2}{\tau^2}\sum_{j=1}^{p}\boldsymbol{\theta}_j^2 \tag{A.7}$$

We can further simplify this equation in terms of regularization. Instead of stating $\frac{\sigma^2}{\tau^2}$, which is the standard deviation of our output equation squared divided by the standard deviation of our weights squared, we can give this a name of λ. Note that this is a different variable and concept than the λ with poisson regression, giving us a final form of

$$= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)}\right)^2 + \lambda\sum_{j=1}^{p}\boldsymbol{\theta}_j^2 \tag{A.8}$$

A.4 Proofs for SVD

$$
\begin{aligned}
\hat{b},\, \hat{\boldsymbol{w}} &= \mathcal{L}(b,\, \boldsymbol{w}|\mathrm{y}) \\
&= j = 1 \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{\left(y^{(i)} - \hat{y}^{(i)}\right)^2}{2\sigma^2}} \\
&= \log\left(\prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{\left(y^{(i)} - \hat{y}^{(i)}\right)^2}{2\sigma^2}}\right) \\
&= \sum_{i=1}^{n} \log \frac{1}{\sigma\sqrt{2\pi}} + \log e^{-\frac{\left(y^{(i)} - \hat{y}^{(i)}\right)^2}{2\sigma^2}} \\
&= \sum_{i=1}^{n} \log e^{-\frac{\left(y^{(i)} - \hat{y}^{(i)}\right)^2}{2\sigma^2}} \\
&= \sum_{i=1}^{n} -\frac{\left(y^{(i)} - \hat{y}^{(i)}\right)^2}{2\sigma^2}
\end{aligned}
$$