Harvey Thompson

Registration number 100332888

2024

A Survey, Analysis And Evaluation Of Machine Learning Models For Recommendation Systems

Supervised by Jason Lines

University of East Anglia
Faculty of Science
School of Computing Sciences

Abstract

This report provides a comprehensive examination of machine learning techniques and algorithms used in recommendation systems with a focus on movie recommendations. The main objective of this study is to evaluate and compare different machine learning models for personalized movie recommendations using the MovieLens 100K and 1M datasets and the Sci-kit Learn Surprise Library. The methods employed include data analysis, model evaluation using cross validation and grid search, and development of weighted hybrid ensemble models. The results of the study reveal insight into the performance and suitability of various algorithms, including K-Nearest Nerighbors and Matrix Factorization. The evaluation metric includes root mean squared error (RMSE)and computation time. The study concludes that matrix factorization models, specifically SVD outperform K-Nearest Neighbors in terms of accuracy and prediction times. The study also determines that the computation cost of combining algorithms make them less real-world practical without future study and refinement. The findings contribute to a deeper understanding of recommend algorithm and provide insight for the design of hybrid systems.

Acknowledgements

I would like to thank my supervisor Jason Lines for his continued support throughout this project especially when I felt I was at a dead end. I would also like personally thank my good friend Elliot for going on a year abroad, with you here I would had never been able to work this hard. Finally, I would like to thank my family for not questioning my progress on this project at any time.

Contents

1. Introduction

1.1. Overview of Recommendation Systems

A recommendation system is a type of machine learning algorithm that is widely used in all a range of fields such as entertainment, e-commerce and much more to suggest items or actions that a user may be interested in. The purpose is to provide mutual benefits to both organisations and users by stereotyping preferences and interests of past behaviours. These systems use advanced machine learning techniques and vast data pre-proccessing to provide every individual user with a personalised experience.

Recommendation systems have gained significant attention in recent years due to the widespread use across modern applications such as e-commerce, social networks and online streaming platforms. In today's digital age, applications are flooded with information and different product options which makes it increasingly more difficult for users to find what they are looking for. For example, the sheer size of Amazon.com's catalogue would make it practically impossible to navigate without the help of these underlying services that rank and recommend items. Using artificial intelligence is essential in alleviating this problem and providing a satisfactory service to users.

While technology giants such as Disney, Amazon and Ebay have definitely contributed to the advancements of these algorithms, the first recommendation system *Grundy*, as documented in Rich (1979) dates back to the 1970s. The Grundy computer librarian would first interview the user about their preferences and then using this collected data make recommendations on what books to take out. It used a rather primitive method, ending up with the same users being recommended the same books.

1.2. Aims and Objectives

The primary aim of this study is to conduct a comprehensive examination of existing and popular machine learning techniques and algorithms that are used as building blocks for modern day recommendation systems. This study will focus on movies as a subject for recommendations using a combination of the MovieLens 100K and 1M datasets to test and develop each model. The ultimate goal is to implement a range of personalised recommendation systems using Python and the sci-kit learn Surprise library. In order to achieve this aim, the study will tackle each of the following objectives:

- Analysis of data- in order to build personalised algorithms, it is imperative to understand trends inside the datasets used. The movie-lens datasets provide not only explicit rating data but also implicit data about each user that can be leveraged.
- Evaluate a range of machine learning models, including but not limited to K-Nearest Neighbours and Matrix Factorisation. Throughout the evaluation different parameters will be tweaked to change the way each model works. The goal here is to find a optimal setup that yields the best accuracy.
- Use cross validation to estimate the accuracy of each model and compare the performance using metrics which include root mean squared error (RMSE), mean absolute error (MAE) and computational time.
- Develop a portfolio of different recommendation algorithms that perform the best and re-evaluate their performance on a larger more realistic dataset to compare results.
- Build hybrid ensemble models and experiment using different weights with the intent of improving accuracy performance.

By the end of the study, there should be a deeper understanding of how different types of recommendation algorithms work, the merits and demerits of different approaches and several weighted hybrid ensembles that can compete with the existing implementations.

2. Background

2.1. Filtering Techniques

All recommendation systems can be placed into categories; 'personalised' or 'non-personalised'. In this study, the focus will be predominantly on the prior because these are the systems that provide *real* recommendations for unique users where as the latter is a universal ranking for the entire user base. For personalised recommendations many techniques however most will stem from either Content-based filtering or Collaborative filtering. Das et al. (2017) outline the merits and demerits of each approach and the authors work has provided a basis of knowledge for the work done in this study.

2.2. Content Based

Content based filtering algorithms are built off the features of items in the systems catalogue mixed with the features of items each user has previously liked. The algorithm matches items to users based of historical activity as can be seen in Figure 1. A merit of using a content based algorithm is that no data from other users is required or data such as explicit ratings. On the other hand, one demerit is that contextual analysis of each item is required in order to extract features.

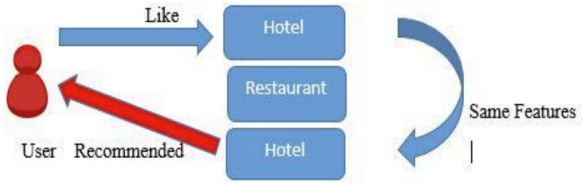

Figure 1: Content Based Filtering

Van Meteren and Van Someren (2000) suggest that the inception of content based filtering is the content page of a website of book, the authors point out that this is a binary classification problem (users either like or dislike content). Their proposed system was the PRES (Personalised Recommender System) which was an advancement of the original WebMate program Chen and Sycara (1998) which was one of the earliest of its kind. The singular frameworks did not perform very well on their own with WebMate achieving a final average accuracy of 30.4 percent however advancements in feature extraction such as metadata and machine learning models allow for content based systems to solve small scale problems or be combined with multiple algorithms.

Many different machine learning models can be used to build content based systems.Hijikata et al. (2006) use a decision tree approach in their work to create a music recommendation model. The music data set can be used to build a tree(s) based of each users preferences from a survey. The model then builds a tree similar to what is seen in Figure 2. This method could be improved by using an ensemble method such as random forest for better accuracy and prediction, however it is unlikely a content based system will work in this study due to the data set only containing rating and no extra information on each movie.

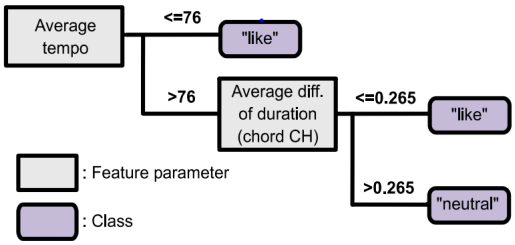

Figure 2: Content Based Filtering Using A Decision Tree

2.3. Collaborative

Collaborative filtering is the other main type of technique for recommendation systems. It works by leveraging users behavioural data to generate personalise recommendation and identifying patterns in user-item interactions. Two types of collaborative filtering exists, user-based and item-based; a user based model works by using a similarity measure on each user to find subsequent users that have rated the same items in a similar matter. Koohi and Kiani (2016) demonstrate in further detail how different user based algorithm works and discuss additional layers such as clustering. The latter option is to use item-based, although user- based has been very successful in the past, widespread use revealed potential challenges including sparsity, especially where a catalogue has a large number of item with little interactions from users. Item based provide an advantage by pairing items together rather than users but then would find it more difficult to accurately predict when dealing with new or less popular items. Sarwar et al. (2001) tests both user-user and item-item collaborative filtering algorithms in which their works find item-item to score the best MAE score.

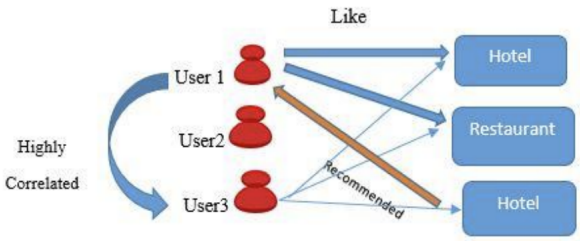

Figure 3: User based collaborative filtering example

The way a typical collaborative filtering algorithm works is by having a user-item rating matrix with a list of users U and a list of items I. Each user U_i has a list of items

$I_u i$ which the user has expressed an opinion about this item, via either explicit rating (1-5) or in some cases implicitly rating through collected data of some means. It is easy to spot how this can return a sparse a ratings matrix, if a data set only includes explicit ratings, its unlikely that many users have rated every single item they have liked/disliked.

Another problem that traditional algorithms face is the cold start problem. This refers to when a new user signs up and the system has zero data for this user and is unable to generate an accurate prediction using collaborative filtering. Melville et al. (2002) propose a 'content boosted collaborative' algorithm to tackle the problems discussed. The authors work used a pseudo-rating matrix where for each user, items that had not been rated would be implied by the content based system. The pseudo- rating matrix would then be used in the same was a normal collaborative filtering algorithm and combined with harmonic mean weighting and self weights to provide the final predictions. The merit of this is the creation of better neighbourhoods due to the dense ratings matrix as well as use of weights to prioritise active users. However the demerit of this is that storage is required to hold every single one of the implicit ratings and the authors did not comment on either the extra computational power and memory space required. As shown by Table 1, this method does perform better to the models tested against.

Table 1: Summary of MAE scores for content boosted collaborative hybrid filtering

Algorithm	**MAE**
Content Based	1.059
Collaborative	1.002
Naive Hybrid	1.011
Content-boosted	0.962

2.4. K- Nearest Neighbours

K- nearest neighbours is one of the most simple but fundamental classification models (Peterson, 2009). The way this model operates is by calculating the similarity between instances and then forming a *Similarity Matrix*, these instances can either be the users or the items depending on works better for the specific problem. To classify an object, the test instance will be measured against each instance in the training set and finding the

'K' nearest instances before voting on which class is the most popular between neighbours. In terms of use in a recommendations system, rather than classify, predictions can be generated using mathematical equations such as taking the mean rating.

Multiple equations for computing the similarity exist. Some of the most popular include:

- Euclidean Distance- Calculates the distance between two points by taking the square root of X's features squared + Y's features squared.
- Pearson's Correlation Coefficient- A statistical measure to quantify a linear relation between two instances. Unlike euclidean distance values can be negative with values ranging from -1 to 1.
- Cosine Similarity- Uses the dot product of vectors divided by their magnitudes where the vectors are the features of each instance.

Wide machine learning literature portrays opposing views regarding which similarity measure is best, creating space for debate and discussion on the topic and each has their merits depending on the problem at hand. Both McLaughlin and Herlocker (2004) and Goldberg et al. (2001) both use euclidean distance in their collaborative algorithms stating it is the most widely used and cited. However more recent research, notably by Al Hassanieh et al. (2018) suggest that a weighted Pearson's correlation performed to the highest degree of accuracy when evaluating seven different similarity measures on the movie lens 100k dataset.

2.5. Hybrid models

A hybrid recommendation system works by combining different algorithm components such as content-based and collaborative algorithms in order to achieve a better degree of accuracy in providing recommendations. Thorat et al. (2015) suggests that using a hybrid model provides a mitigation to problems such as sparsity and the cold start problem. The author provide multiple ways to implement as followed:

- Implement collaborative and content-based methods individually and aggregate their predictions.
- Integrate some content-based characteristics into a collaborative approach.

- Comprise some collaborative characteristics into a content-based approach.
- Construct a general consolidative model that integrate both content-based and collaborative characteristics.

Furthermore, hybrid systems can be split into different classes. A taxonomy by Burke (2002) outlines the seven different classes as followed:

- **Weighted-** Different recommendation component scores are combined statistically using some sort of forumla.
- **Switching-** The system switches between recommendation components based on a criterion.
- **Mixed-** Different component are presented together which can be used where large no. of recommendations are required.
- **Feature combination-** This class uses collaborative data as additional feature data rather than combining components thus reducing the sensitivity to the numbers of rated items.
- **Cascade-** Each component is weighted and can be used as a voting system using low priority ones as tie-breakers.
- **Feature Augmentation-** Uses a sequence of components however unlike the cascade class, each component has a influence on the following.
- **Meta Level-** Similar to augmentation however the entire model generated by a component is used an input for another.

2.6. Other Approaches

Documented by Bennett et al. (2007), in 2006 before Netflix was the FAANG company it is today they hosted a contest looking to improve their current algorithm *CineMatch*. A dataset of one hundred million movie rating was released to the public offering prizes to any algorithm that could deliver an improvement in RMSE by ten percent. Many of the solutions proposed were linked with matrix factorisation. Gower (2014) outlines some of the solutions including the famous SVD and SVD++ algorithm, of which both will be analysed later in this study.

Another approach proposed is a review based filtering system. Duan et al. (2022) and Liu et al. (2013) both implement similar systems that use supervised learning to extract feature candidates that match a pre-defined dictionary taken from online reviews and grouped together to identify features of an items. Again, opinions can be collated from the reviews to build user profiles so that item-topic and user-item matrixes can be built. The algorithm PORE by Liu et al. (2013) compared to traditional collaborative algorithms in accuracy and effiency which can be seen in Table 2 which has been condensed from the authors work.

Table 2: MAE scores for collaborative filtering and the PORE algorithm

Algorithm	**MAE 1**	**MAE 2**	**MAE 3**	**MAE 4**	**MAE 5**	**Average**
PORE	1.22	1.18	1.18	1.18	1.29	1.19
Item-based CF	3.11	3.10	3.09	3.09	3.10	3.10
User based CF	2.99	2.96	2.95	2.95	2.95	2.97

Despite the proven benefits, in this study a review based model would be ineffective primarily down to the fact the data pre proccesing would require a web scraper to find reviews which for movies can be sparse. Explicit rating are far more likely to provide more useful information that can be used in each algorithm.

3. Methodology

3.1. Language Choice

For this study the language of choice is Python. One reason for this decision is due to the data exploration and visualisation libraries available to use along with the library Surprise, which is at the core of the study providing prebuilt recommendation models for testing. Despite languages such as Java offering an object oriented interface to build recommendation components and JavaScript offering machine learning libraries such as TensorFlow, ultimately the functionality of Jupiter notebooks along with a library built off the infrastructure sci-kit learn was the factor that pushed it to the language choice.

The libraries Pandas (McKinney et al., 2010) and Numpy (Harris et al., 2020) are both popular data manipulation libraries and form a starter pack for any data science project. Pandas (pd) offers many powerful tools, one in particular is the Dataframe.

Dataframes are 2D table structures used to store data such as the movie lens rating set. Offering a similar interface to a traditional SQL table, data can easily be indexed, manipulated and written to csv files, making it perfect for working over time without using a database. Numpy (np) is library for working with numbers, notable functionality includes array operations and array slicing, reducing the effort to combine large sets of numbers and making it very simple to implement a weighted hybrid recommendation model. Both libraries are designed to integrate with each other and other different data science libraries such as Matplotlib.

Many different libraries for recommendation engines exist, this study will focus on the Sci-Kit Learn Surprise Library (Hug, 2020) and will benefit from the Sci-Kit base library as well (Pedregosa et al., 2011) AKA sklearn . The surprise library is specifically designed to build, test and evaluate recommendation algorithms. It focuses on collaborative filtering methods to analyse user-item interactions to make recommendations. The library has several different types of algorithms available out the box such as KNN and matrix factorization which can be tuned and modified depending on the problem for example this study will be making recommendations on different movielens datasets. Surprise also offers several utilities extending from sklearn such as method to split data, cross validation and hyper-parameter tuning; all which will be invaluable tools in designing and building a hybrid.

3.2. The Dataset(s)

The dataset(s) that are used in this study are the MovieLens 100k and 1M (Harper, 2016). They are both popular choices among the literature for evaluating and developing recommendations systems. The prior contains approximately 100,000 movie ratings by 943 users on 1682 movies and the latter contains a lot more ratings totalling approximately 1 million ratings by 6040 on 3706 movies. Both datasets are formatted in the way they include extra key information such as user demographics, movie information and timestamps however the 1M will offer more a comprehensive and diverse collection of rating and mimic a real-life system more closely.

In the study, the ml-100k dataset will be used for configuring different setup of algorithms and evaluating the best parameter for each model. Although it could be argued that using the 1M would be the better option, due to the sheer range of algorithms that are being analysed and the vast amount of different parameter setups exist for each

algorithm, using the smaller of the dataset is far more computationally beneficial. Additionally, to use the larger of the two would require up to 10x the power/time to complete. Each algorithm will go through an evaluation process that uses the ml100k dataset in order to find the most optimal parameters and then will then be trained and tested on the 1M dataset. This process of cross-dataset testing will ensure that the algorithms can be generalised to new dataset that contain movie rating and ensures each model will not be over fitted to only work on a singular specific dataset and due to similarities between the datasets should ensure the results collected are easily transferable.

3.3. Data Exploration

Understanding the data set being used in this study is vital to the success of the models, by getting a deeper understanding of the raw data the results of each algorithm will be more intuitive. This statistical analysis should build trust that cross dataset testing is an appropriate approach for this problem.

As previously discussed, both datasets offer more than just explicit ratings. Demographic data including age, gender, occupation is also available however was an optional field for participates and therefore may not be complete. The ml-100k offers the absolute age of each user but the ml-1m only offers an age range for each user therefore for simplicity and readability, the users in the 100k dataset have been also grouped into these age groups.

Figure 4 shows the distribution of ages in the data. While the histograms for each dataset are not identical, the distributions are close enough to consider similar. It can be inferred from these graphs that the 25-34 group are the most active with the ratings with 18-24 and 35-44 cohorts slightly less active and with drops the further it skews from these groups. This data suggests that the recommendation models may provide these groups with a lot more accurate ratings than 56+ where ratings are sparse.

Figure 5 show the distribution of film rating across both datasets split by gender. As shown, the distribution is asymmetric and skews to the higher end of ratings, this demonstrates that when a user dislikes an item they will not rate the item and explicit ratings only occur when a user ‘likes’ an item. Essentially this will be used to create a model of ‘likes’ and ‘dislikes’, classifying anything above 4 as a ‘like’. This data additionally portrays that although the count of male rating and significantly higher than female, a generic model on the entire population will be sufficient as the distributions

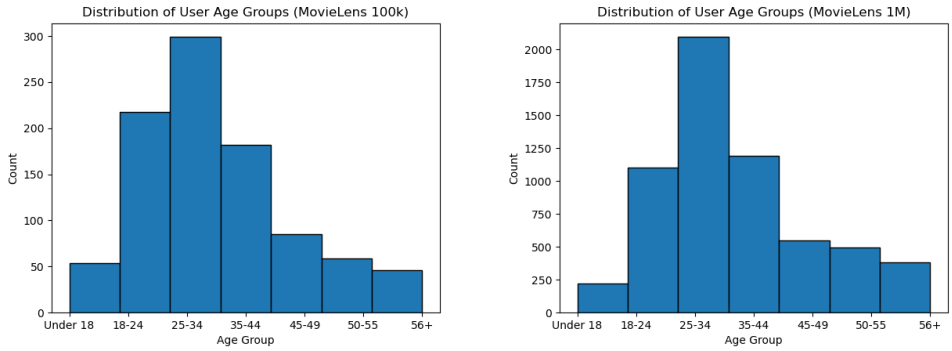

Figure 4: Age distribution for both datasets

follow the same pattern.

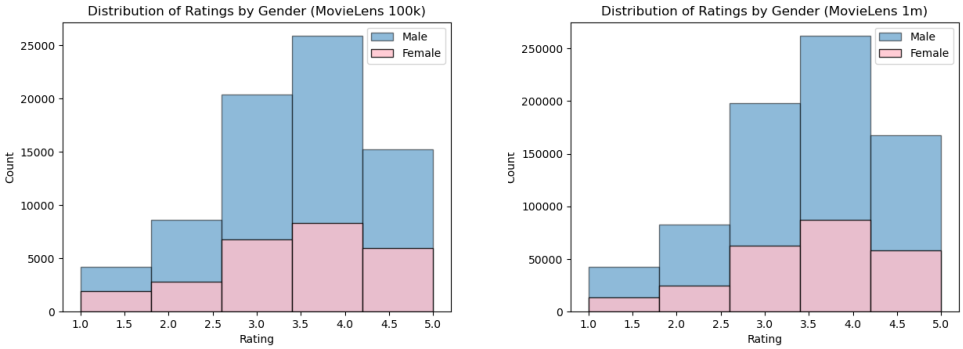

Figure 5: Rating distribution for male and female. N.B this is a not a stacked histogram, both start from 0

Using the classifier of like and dislike, Figure 6 depicts the percentage of ratings for each different genre that would be classed as 'like'. The data shows certain genres have higher ratings depending on gender. For example the category of 'romance' has a much higher percentage of female high ratings and 'action', while not as significant has a higher male like rate. It is possible that a model can incorporate these additional features as inputs in certain models, particularly a neural network where the more inputs equals better results.

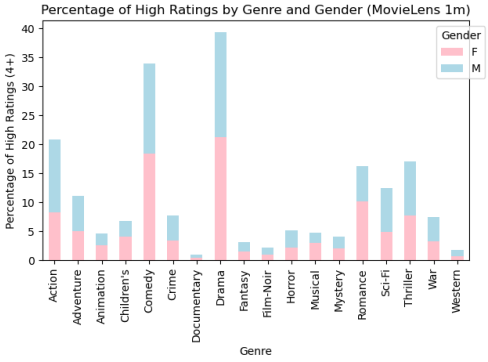

Figure 6: The percentage of high ratings (4+) per genre for each gender. N.B. This is a stacked histogram, M starts where F finishes

3.4. Notation

In this report you will find the following notation. This notation directly follows the standard outlines by Surpirse[1]. For further details please reference the official documentation.

R:the set of all ratings.

U: the set of all users. u and v denotes users.

I: the set of all items. i and j denotes items.

U_i: the set of all users that have rated item i

U_{ij}: the set of all users that have rated both items i and j

I_u: the set of all items rated by user u

I_{uv}: the set of all items rated by both users u and v

r_{ui}: the true rating of user u for item i

$\hat{r}_{ui}$: the *estimated* rating of user u for item i

b_{ui}: the baseline rating of user u for item i

[1] *https://surprise.readthedocs.io/en/stable/notation_standards.html*

3.5. K-Nearest Neighbors (KNN) Analysis

3.5.1. K Fold Cross Validation

The first section of the study aims to test and optimise the KNN algorithms. Surprise offers several versions of KNN, all with tuneable parameters such item-based or user-based similarity . To analyse the algorithms accuracy rates, k-fold cross validation will be used. K-fold cross validation is a technique to evaluate models by splitting the data in K equally sized subsets, then training the model on K-1 subsets using the last set for testing until all K combinations have been testing. This method can provide more reliable and unbiased estimations for predictions by taking the average value for each metric. Another method is leave-one-out validation which works similar how it trains on n-1 instances where n is the number of data instances and uses the final one to compute accuracy and this is repeated n times. Although leave-one-out is a popular method for evaluating recommendation systems, due to the computation cost, a 5 fold cross validation algorithm was chosen, the code for this can be seen in the Listing 1 and the full function in Appendix A.

```
1
2 # Define the K-fold cross-validation iterator with 5 splits
3 kf = KFold(n_splits=5, random_state=1)
4
5 # Perform K-fold cross-validation
6 for trainset, testset in kf.split(data):
7     # Train the algorithm on the training set and measure the time
    taken
8     start_fit = time.time()
9     algo.fit(trainset)
10    fit_times = np.append(fit_times, time.time() - start_fit)
11
12    # Predict the ratings for the test set and measure the time taken
13    start_predict = time.time()
14    predictions = algo.test(testset)
15    predict_times = np.append(predict_times, time.time() -
    start_predict)
16
17    # Compute the RMSE, MSE, and MAE scores for the predictions
18    rmse_scores = np.append(rmse_scores, rmse(predictions, verbose=
    False))
```

```
19      mse_scores = np.append(mse_scores, mse(predictions, verbose=False
        ))
20      mae_scores = np.append(mae_scores, mae(predictions, verbose=False
        ))
```

Listing 1: Code snippet for performing cross fold validation in python

The function returns the average RMSE,MSE, MAE and timing data. RMSE (root mean squared error) is commonly used to quantify the distance between predictions and the actual value and is computed as $\sqrt{\frac{1}{|\hat{R}|}\sum_{\hat{r}_{ui}\in\hat{R}}(r_{ui}-\hat{r}_{ui})^2}$. and will be the main accuracy metric used to evaluate the models in this study. The lower the value is, the better the accuracy. RMSE penalises larger errors more heavily than other accuracy measures making it sensitive to outliers in the dataset and therefore MAE (mean absolute error) computed as $\frac{1}{|\hat{R}|}\sum_{\hat{r}_{ui}\in\hat{R}}|r_{ui}-\hat{r}_{ui}|$ will also be used. The training and prediction times will also be used to understand the relationship between accuracy and time.

3.5.2. Distance Measuring

When it comes to computing the similarities between two items, many statistical methods exist. The equations that will be examined will be the Cosine similarity measure and the Pearsons Correlation where Cosine is computed as $\frac{\sum_{i\in I_{uv}} r_{ui}\cdot r_{vi}}{\sqrt{\sum_{i\in I_{uv}} r_{ui}^2}\cdot\sqrt{\sum_{i\in I_{uv}} r_{vi}^2}}$ and Pearson as $\frac{\sum_{i\in I_{uv}} (r_{ui}-\mu_u)\cdot(r_{vi}-\mu_v)}{\sqrt{\sum_{i\in I_{uv}} (r_{ui}-\mu_u)^2}\cdot\sqrt{\sum_{i\in I_{uv}} (r_{vi}-\mu_v)^2}}$. (Note these equations are for users, for items replace u, v with i, j) Surprise offers parameter tuning for similarity which makes it easy to run an experiment to find the configuration that returns the lowest RMSE. The pseudo code in Algorithm 1 demonstrates the process to evaluate each similarity measure.

3.5.3. Designing a Hybrid

The results of the similarity experiment will allow for creation of the first hybrid design of this study. This KNN hybrid model will use a weighted 50-50 hybrid approach, using a user-based component and a item-based component. Each will be trained independently on the train set and will generate two prediction sets *predictA* and *predictB*. The results will be combined using array arithmetic and the final prediction set will compute as $(predictA * 0.5) + (predictB * 0.5)$.

Algorithm 1 Testing k values and similarity measures

```
Initialize empty dataframe user_data with appropriate columns
Initialize empty dataframe item_data with appropriate columns
for k in range(20, 101, 5) do
    for type in ['User', 'Item'] do
        for measure in ['Person', 'Cosine'] do
            Create algorithm using k, type, and measure
            results ← testingAlgorithm(algorithm)
            if type = 'User' then
                Add results to user_data
            else
                Add results to item_data
```

The idea behind using both an user and item based component is that the model will generalise well, if the dataset is heavy in items but not users, the item-based component will boost the accuracy of the predictions and vice-versa. A confounding problem of this model is computational time as two separate models are being trained and tested to get the dual prediction arrays, the time of the hybrid will be compared to the other models in this study to make a fair conclusion on overall performance.

The design for the model can been seen in Figure 7 and the prediction arithmetic can be seen in the Listing 2 and the full code for the class can be found in Appendix B.

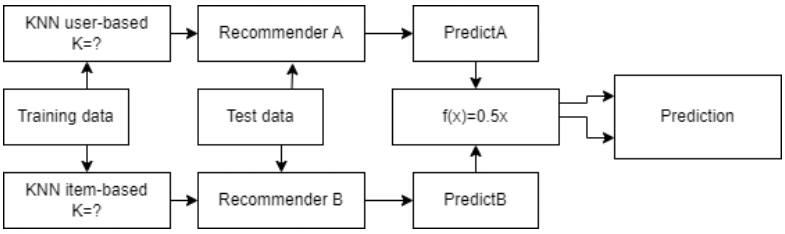

Figure 7: KNN hybrid component block diagram

```
def estimate(self, u, i):
    # generate predictions for each model and combine
    user_prediction = self.user_algo.predict(u, i).est
    item_prediction = self.item_algo.predict(u, i).est
    return self.u_weight * user_prediction + (1 - self.i_weight) *
```

```
    item_prediction
```

Listing 2: Estimate function to generate rating prediction for hybrid model

3.5.4. Baselines

All of the KNN algorithms that have been considered so far could possibly suffer from sparsity problems. Surprise offers two algorithms to combat this problem, Zmeans and Baseline. In this work, the KNNBaseline will be optimised on the ml-100k using a process called GridSearch. See 3.6.1 for further details. The parameter grid that will be used is :

```
param_grid = {'k': [10, 20, 30, 40, 50, 60],
        'sim_options': {'name': ['cosine',
            'pearson_baseline'],
                'min_support': [1, 5],
                'user_based': [True, False]}}
```

Once all combinations have been cross validated, the lowest RMSE score producing model will retrained and tested on the ML-1M to get the final accuracy score. The results of the KNN model analysis will determine which algorithm is chosen for the final hybrid model.

3.6. Matrix Factorisation Modes Analysis

Surprise also offers algorithms for matrix factorisation models. The first algorithm SVD follows the implementation of one of the algorithms made popular by the Netflix Prize Koren et al. (2009), predictions are computed as $\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u$ and the algorithm using a straightforward stochastic gradient descent algorithm as a loss function to minimise error. The second matrix factorisation algorithm is essentially an extension of SVD. SVD++ computes predictions as : $\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left(p_u + |I_u|^{-\frac{1}{2}} \sum_{j \in I_u} y_j \right)$ where y_j is a new set of item factors that capture implicit ratings. See Koren (2010) for details.The third and final algorithm for this section of analysis is NMF (Non-negative Matrix Factorisation) a very similar algorithm again to SVD as it computed as $\hat{r}_{ui} = q_i^T p_u$, but where user and item factors are kept positive; the Surprise implementation follows the suggestions of Luo et al. (2014).

Using similar methodology to the KNN analysis, the resuts gained from this part of the study will decide the algorithm used to build the second and final hybrid recommender.

3.6.1. Grid Search Cross Validation

To optimise the algorithms for testing, a similar process to the custom algorithm used for KNN will be used, however, due to the additional complexity of matrix factorisation and the sheer number of tuneable parameters that exist for these algorithms, an automated process will be used in place.

Grid search is a technique in machine learning that is commonly used to optimise the parameters of models for the problem at hand. The process works by systematically searching through a predefined set of hyperparameter values to find the combination that achieves the best performance in a specific metric for that model. In this case RMSE will be used. Each unique combination is trained and evaluated using a K-fold validation and the best parameter setup is returned by the function. The matrix factorisation have a large parameter space which can make this type of optimising computationally expensive however grid search was chosen due to the fact can take advantage of all available cores heavily speeding up the process. For example, to evaluate an algorithm using 6 different parameters with 6 different values for each using 5 fold validation, would require $6*6*5(180)$ runs training and testing the model.

Again, parameter optimisation will use the Ml-100k dataset, and once the optimised models are returned, they will be retrained and tested on the ml-1m dataset in order to confirm cross dataset engineering will work and to ensure the model can generalise. Figure 8 provides a graphical representation of the optimisation process.

3.6.2. Parameter Grids

Listing 3 shows the code for each parameter grid that will be used for the grid searches.

```
1    #Grid for SVD
2    param_grid= {'n_factors':[20,50,150,200,250,300],
3             'lr_all':[0.005,0.01, 0.015, 0.02, 0.05],
4             'reg_all':[0.05, 0.075, 0.1, 0.15, 0.2],
5             'n_epochs':[30,40,50,60,70]}
6
7    #Grid for SVD++
```

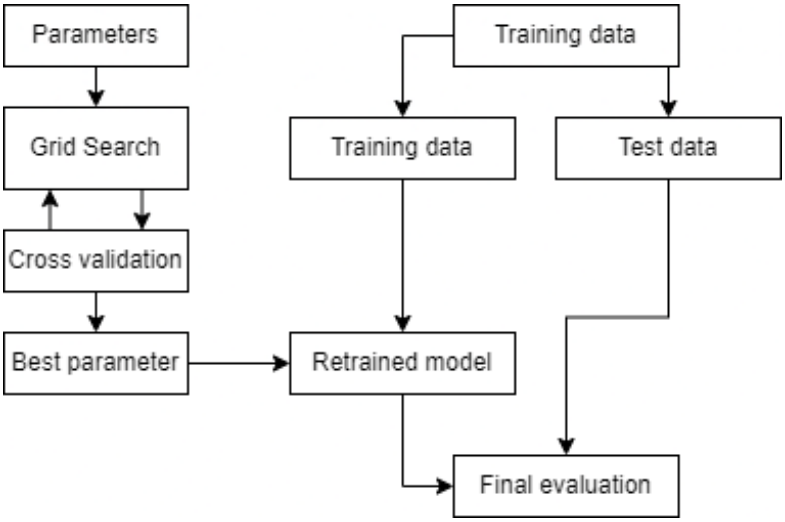

Figure 8: Grid search

```
    param_grid= {'n_factors':[20,50,100,200,300,400],
                'lr_all':[0.005, 0.01, 0.02, 0.05],
                'reg_all':[0.05,0.1,0.2, 0.3],
                'n_epochs':[30,40,50,60,70]}

    #Grid for NMF
    param_grid = {'n_factors':[20,50,100,200],
                  'lr_bu':[0.005, 0.01, 0.02],
                  'lr_bi':[0.005, 0.01, 0.02],
                  'reg_bu':[0.05,0.1,0.2],
                  'reg_bi':[0.05,0.1,0.2],
                  'n_epochs':[40,50,60,70],
                  'biased':[False, True]}
```

Listing 3: Grids for hyperparamater tuning.

3.7. Combining KNN and Matrix Factorisation

Using the data collected from the previous two experiments to aid the design, the final algorithm will again use a weighted architecture similar to the KNN hybrid seen in section3.5.3 and Figure 9 shows the adaptations for this model. The motivation behind this design is that the KNN and matrix factorisation can complement each other's strengths; for example, a user with a low amount of neighbours will benefit from the weighting

from the other component. The aim is to be able to achieve a lower RMSE than both singular algorithms individually.

The algorithm will be comprised of the best overall performing in both categories, and the weights will be decided by lowest RSME scoring model from the following Algorithm 2.

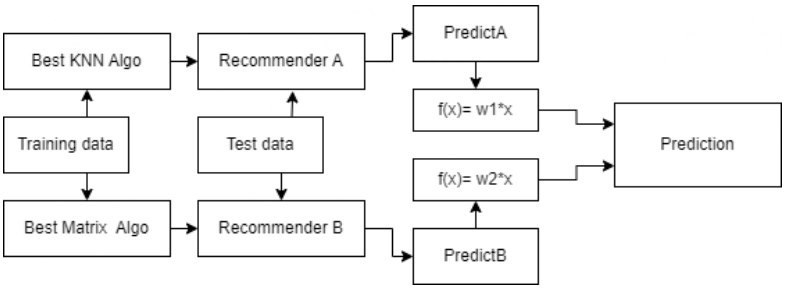

Figure 9: Hybrid

Algorithm 2 findBestWeights(*knn*, *matrix*)

Initiate weights: $weights \leftarrow \text{np.arange}(2, 8, 0.5)$
Initialize lowest RMSE: lowest_rmse $\leftarrow \infty$
Initialize best weights: best_weights $\leftarrow$ None
for $i \leftarrow 0$ to $length(weights)$ **do**
    $w_1 \leftarrow \text{weights}[i]$
    $w_2 \leftarrow 1 - \text{weights}[i]$
    Build *knn*
    Build *matrix*
    Perform prediction using *knn* :$\hat{R}_{\text{pred\_knn}} \leftarrow$ predictKNN
    Perform prediction using *matrix*: $\hat{R}_{\text{pred\_matrix}} \leftarrow$ predictMatrix
    Calculate combined prediction: $\hat{R}_{\text{pred}} \leftarrow (\hat{R}_{\text{pred\_knn}} \cdot w_1) + (\hat{R}_{\text{pred\_matrix}} \cdot w_2)$
    Calculate RMSE: rmse $\leftarrow$ compute_rmse$(\hat{R}_{\text{pred}})$
    if rmse $<$ lowest_rmse **then**
        Update lowest RMSE: lowest_rmse $\leftarrow$ rmse
        Update best weights: best_weights $\leftarrow \text{weights}[i]$

3.8. Extra Algorithms

To complete a full comprehensive study of recommendation models. the study will analyse addition algorithms that have not been covered by KNN or Matrix Factorisation. The algorithms covered in this section will not need any parameter tuning and hence not undertake any validation but will be trained and tested directly on the ml-1m data.

Slope One is a simple but effective collaborative model (Lemire and Maclachlan, 2005) computed as $\hat{r}_{ui} = \mu_u + \frac{1}{|R_i(u)|} \sum_{j \in R_i(u)} \text{dev}(i,j)$, where $\text{dev}(i,j) = \frac{1}{|U_{ij}|} \sum_{u \in U_{ij}} r_{ui} - r_{uj}$ which is equal to the average difference between the ratings of i and those of j.

4. Implementation and Evaluation

4.1. Determining Baseline Score

The first experiment conducted was an evaluation of the NormalPredictor algorithm and pure random custom algorithm discussed in 3.8. Table 3 show the results of both algorithms on the ml-1m dataset showing a RMSE of approximately 1.5. A poor result was expected and the aim of this section was to establish a baseline RMSE to refer back to in the further studies.

Table 3: RMSE scores of Surprise Normal Predictor and custom PureRandom algorithms tested on ml-1m

Algorithm	RMSE
NormalPredictor	1.5045
PureRandom	1.5067

4.2. Evaluating Nearest Neighbor models

4.2.1. User and Item Based Models

This testing was conducted on the ml-100k using the custom algorithm to compute average accuracy scores shown in section 3.5.1. The aim of this testing was to determine whether one of the two models would perform better than the other, which in turn would

impact the weighting chosen for the first KNN hybrid model. The configuration options used for this are shown in Listing 4, where anything not defined would be defaulted [2].

```
algo = KNNBasic(sim_options={'user_based': False}, verbose=False)
algo = KNNBasic(sim_options={'user_based': True}, verbose=False)
```

Listing 4: Model setups used in testing

Figure 10 shows the results of 5 fold cross validation where the mean scores have been taken and plotted. As the graph shows, the accuracy for both models are almost identical and close to 1, which is an improvement over the random predictor. The reason such similar RMSE scores could be down to the small dataset with similar user and item interactions or possibly due to both models using default values for K (40) and it is possible using different k values could benefit the different models.

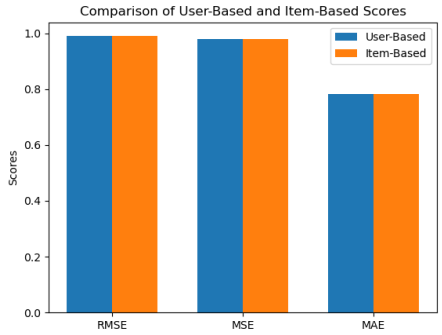

Figure 10: Mean accuracy results from 5 fold cross validation of KNNBasic using item and user relationships on ml-100k

4.2.2. Hybrid Solution

Due to the results in section 4.2.1, for the first KNN hybrid it was decided to used was a 0.5-0.5 weighting split. Figure 11 and Table 4 shows the results of the hybrid compared to its individual components. The results show a better score in all RMSE, MSE and MAE which would suggest the algorithm is an improvement. But it is also clear that the using the hybrid solution results in a dramatic time increase for both training and testing due to the fact the model must first train each component then generate predictions

[2] *https://surprise.readthedocs.io/en/stable/knn_inspired.html*

sequentially and then factor in the additional cost of the array arithmetic used to combine both prediction objects.

Table 4: Results

Algorithm	RMSE	MSE	MAE	Fit time	Predict time
User-Based	0.989869	0.979859	0.783560	0.288202	1.676250
Item-Based	0.989914	0.979943	0.782299	0.207619	1.445565
Hybrid	0.969734	0.940407	0.771335	2.130322	5.040482

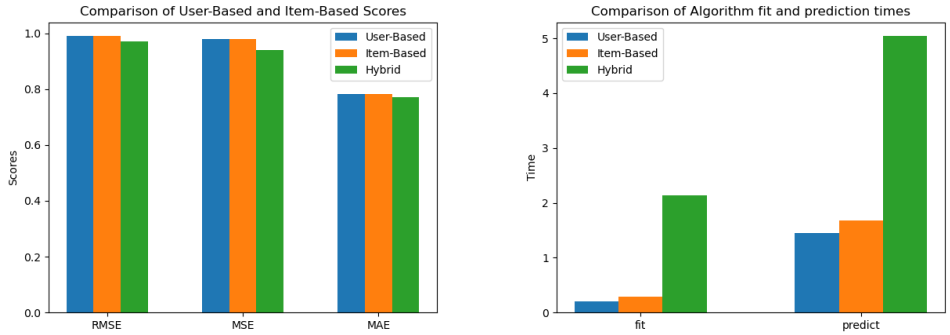

Figure 11: Results of hybrid algorithm in accuracy (left) and timing (right) against KNNBasic

Due to the fact this model sees such little improvement with a dramatic increase in time to fit and a less dramatic but still large in time to predict, it would be difficult to consider this first hybrid iteration a real success.

4.2.3. Improving the Components

The results from section4.2.2 show little improvement in the RMSE value, however this was expected and the following evaluation results will attempt to improve performance to a sufficent degree.

Using Algorithm 1, the RMSE average score for each k value and similarity measure was recorded and this was the repeated for both KNNBasic using user-based and item-based parameters. The results in Figure 12 show that for a item based model that using Cosine similarity yields better results for all values of k up until 100, it is possible they

may converge as this is the general trend however at a k value of 60 the lowest score is achieved at approximately 1.035.

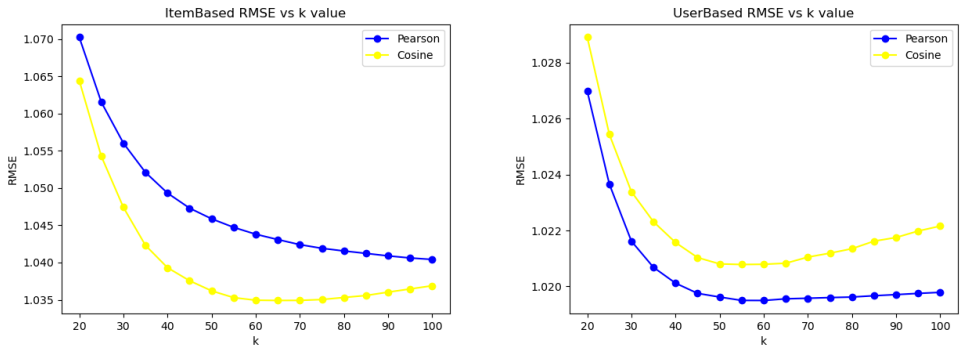

Figure 12: Results of k value and similarity measure testing for item-based (left) and user-based (right) KNN model

The figure also depicts the results of the evaluation for the user based component. These results show that the difference for similarity measures is very little, with the difference between the lowest and highest being approximately 0.1. With this being said, the results show that for user based, a Pearson similarity does perform better for this given model and the best score was achieved using a k value of 55.

Using the results of this experiment, the KNN hybrid was refactored using the two component setups in Listing 5, with the aim of achieve a significantly lower RMSE. Figure 13 shows the results of the evaluation in terms of RMSE and MAE and show that this is the case with the 2nd iteration hybrid lowering the RMSE to approximately 0.97 from almost 1 compared to an individual model.

```
# create sim option dictionarys
    self.sim_options_user = {'name': 'Pearson', 'user_based': True}
    self.sim_options_item = {'name': 'cosine', 'user_based': False}
# create reccomendation components
    self.user_algo = KNNBasic(k=55, sim_options=self.sim_options_user
    , verbose=False)
    self.item_algo = KNNBasic(k=60, sim_options=self.sim_options_item
    , verbose=False)
```

Listing 5: Improved component setups derived from results shown by figure 12

Unfortunately although the accuracy scores were improved, looking at Table 5 it can be concluded that this hybrid brings no changes in the movies recommended and there-

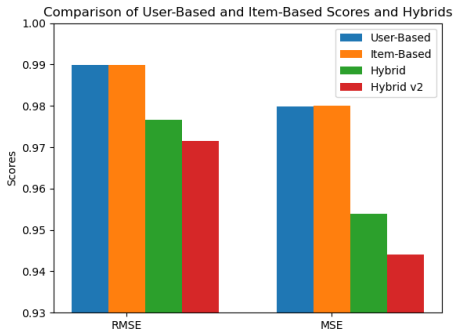

Figure 13: RMSE and MSE scores of original hybrid, improved model and original item, user based models

fore the results of this analysis suggest using the regular KNN algorithm would be more beneficial due the shorter computational times. Tables 16 and 15) in Appendix C support this claim for additional users.

Table 5: Top 5 reccomendations for user 1 returned by KNNBasic using default setup and Hybrid.

KNN Basic	Hybrid
Fargo (1996)	Fargo (1996)
Gigi (1958)	Gigi (1958)
Cinderella (1950)	Cinderella (1950)
One Flew Over the Cuckoo's Nest	One Flew Over the Cuckoo's Nest
Ben-Hur (1959)	Ben-Hur (1959)

4.2.4. Evaluating on Generalised Data

For the final experiment regarding KNN, the models were tested not using cross fold validation but instead on generalized unseen data. In this case 20 percent of the ml-100k that was left out and 25 percent of the ml-1m. In addition to this,a new model KNNBaseline was added in, the parameters for this model derived of the grid search detailed in section 3.5.4. (See Listing 6 for configuration).

```
{'k': 40, 'sim_options': {'name': 'pearson_baseline', '
min_support': 1, 'user_based': False}}
```

Listing 6: Paramaters for best KNNBaseline

Table 6: RMSE score results for KNN algorithms retrained on unseen data of the ml-100k and 1m datasets

Algorithm	RMSE	
	100k	1M
KNNBasic	0.964453	0.925107
Hybrid	0.964453	0.925949
KNNBaseline	0.926451	0.861641

The results of this experiment are depicted in Table 6 and Figure 14, the results confirm the hypothesis of the hybrid algorithm being redundant, as in the ml-100k results the RMSE values are identical suggesting that the models are predicting the same rating, for the ml-1m results the hybrid performs worse than the basic KNN which could be due to the relationships between users and items in the larger dataset weakening one of the components.

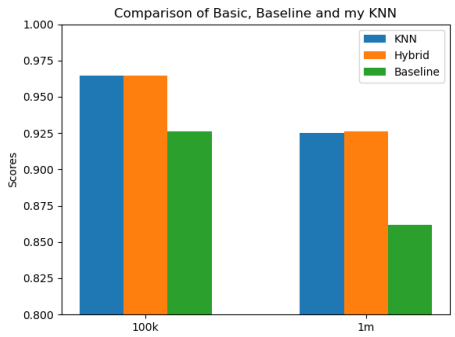

Figure 14: RMSE scores for KNN retrained on unseen data of the ml-100k and 1m datasets plotted

The best performing model was the KNNBaseline, which scored a low RMSE of approximately 8.6 on the ml-1m data which is a great improvement over the algorithms trialed beforehand. It can be also noted all three models produced a lower RMSE on the

ml-1m most likely due to a much larger training set which allowed for more accurate predictions.

Using the KNNBaseline model also produced different prediction lists. Table 7 shows the predictions for user 1's and comparing those to the predictions in Table 5 this model produces a different set of movies to the previously tested; it can be assumed this prediction list is more suitable but without further investigation that surpasses the scope of this study.

Table 7: Top 5 recommendations for user 1 using the KNNBaseline model and grid search parameters

KNN Baseline
Star Wars: Episode IV - A New Hope (1977)
Schindler's List (1993)
My Fair Lady (1964)
Wizard of Oz, The (1939)
Driving Miss Daisy (1989)

From this evaluation, the evidence confidently suggest that the using the KNNBaseline algorithm yields the most accurate predictions of all the KNN models. This concludes exploration in nearest neighbors and the final model that will be used in the combination hybrid described in 3.7 will be as previously seen in Listing 6.

4.3. Matrix Factorisation Methods

4.3.1. Parameter Optimisation

To find the optimal parameters setups, a GridSeachCV method was used in accordance with section 3.6.1. The results from this will make up the configurations used to evaluate each algorithm on the ml-1m dataset. For SVD the setup can be seen in Table 8, for SDV++, Listing 9 and NMF, Listing 10. Note that for a full explanation of parameters and their purpose, see the Surprise documentation[3] .

[3] *https://surprise.readthedocs.io/en/stable/matrix_factorization.html*

Table 8: Best Parameter Setup for SVD Algorithm as returned by GridSearchCV

n_factors	**lr_all**	**reg_all**	**n_epochs**
250	0.01	0.1	50

Table 9: Best Parameter Setup for SVD++ Algorithm as returned by GridSearchCV

n_factors	**lr_all**	**reg_all**	**n_epochs**
100	0.005	0.1	70

4.3.2. Testing the Algorithms

Table 11 and Figure 15 show the results of testing on the unseen 20 percent of the ml-100k. The results first show that although for NMF grid search yields 'biased= true' to score better, these results contradict this heavily with the results as NMF(bias) scores RMSE of approximately 1.47 which is close to the random predictor (see Table 3 and NMF without biases scores a RMSE much lower at 0.96. The other algorithms all score similar in the 0.9-1 range and SVD++ scores the lowest, however the time to fit the SVD++ algorithm compared to regular SVD is almost 100 times longer and predict time is also drastically large.

Repeating the same experiment on the ml-1m yield a similar trend. Table 12 and Figure 16 show the results of this and similar to the KNN algorithms, the matrix factorization model set also produce a better RMSE, which reinforces that a reduction in sparsity and more training data improves model performance on a generalized level. Using this dataset SVD++ performs again slightly better than SVD with a lower RSME by 0.02 again with a very large fit time difference, specially 408 times and therefore it can be comfortably concluded that SVD++ is not fit for purpose in this study and will be omitted from further graphs to improve readability.

The results also show a much better performance for the NMF (bias) algorithm, with a dramatic change from 1.47 to 0.95 RMSE value it can be inferred that using biases

Table 10: Best Parameter Setup for NMF Algorithm as returned by GridSearchCV

n_factors	**lr_bu**	**lr_bi**	**reg_bu**	**reg_bi**	**n_epochs**	**biased**
20	0.02	0.02	0.05	0.2	70	True

Table 11: Comparison of matrix factorisation algorithm on the ml-100k dataset

Algorithm	RMSE	Fit Time	Predict Time
SVD	0.9106	6.3065	0.1500
SVDpp	0.9102	524.2002	4.8900
NMF (bias)	1.4630	2.4630	0.1470
NMF (unbias)	0.9573	2.2960	0.1880

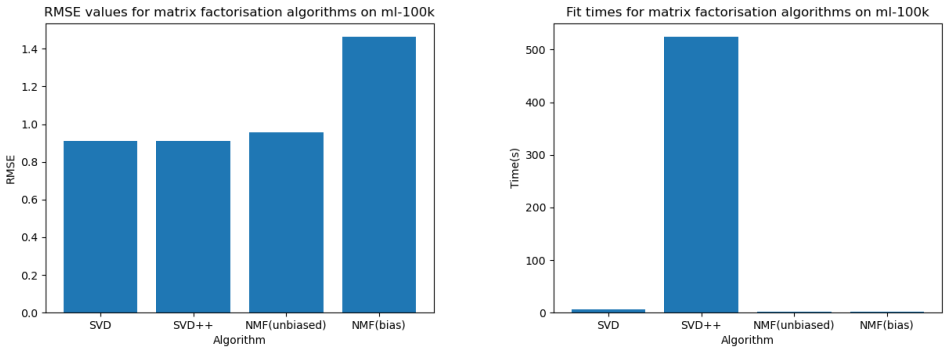

Figure 15: Results of matrix factorisation anaylsis on the ml-100k, it can noted the time for SVD++, that its impossible to interpret the results of the timing data (right)

are beneficial when the dataset is large and where users have rated a certain number of items.

Before experimentation, it had been predicted that the SVD++ would have been the best algorithm, this is due to the fact it takes implicit data such as user-item relationships rather than just explicit rating data. However the results showed that the difference was minimal and the time to process and analyze the extra data was extreme and therefore the SVD algorithm will be used as the second component to construct the hybrid detailed in section 3.7. Looking at Table 13, it is unlikely that using SVD++ would make any difference as both algorithm generated the same top predictions for user 1. The same predictions were produced for user 134 and 398 (See Table 17 and Table 18 in Appendix D.

Table 12: Comparison of Algorithms on RMSE, Fit Time, and Predict Time on the ml-1m dataset

Algorithm	RMSE	Fit Time	Predict Time
SVD	0.8680	31.5431	2.7326
SVDpp	0.8660	12664.3036	100.6409
NMF (bias)	0.9511	21.7209	2.4085
NMF (unbias)	0.9056	21.5159	2.2375

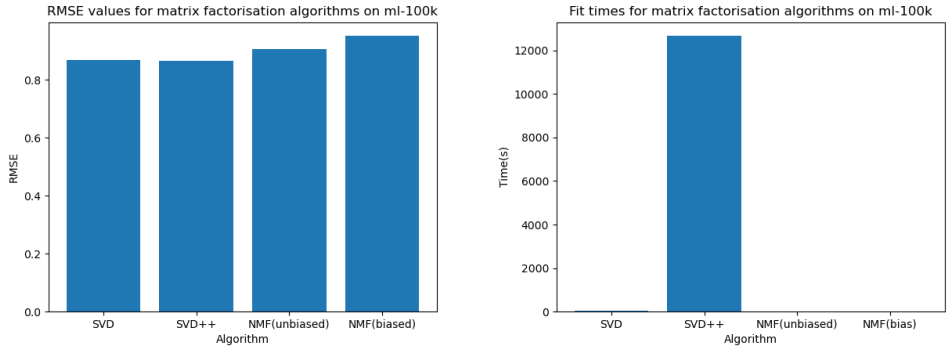

Figure 16: Results of matrix factorisation anaylsis on the ml-1m, it can noted the time for SVD++, that its impossible to interpret the results of the timing data (right)

4.4. Combining Different Models

After independent analysis of both Nearest Neighbours and Matrix Factorisation techniques for recommendation, enough data has been collected to build a combination hybrid. The hybrid *Combiner* will be built according to Figure 9 using KNNBaseline and SVD as these performed objectively best in their respective categories and the parameter setup for each model will be the same as seen prior (see Listing 6 and Table 8). To determine a weight combination, Algorithm 2 was used and the results can be seen in Table 14

Figure 18 shows the results of retrained and tested algorithms on the ml-1m with a 75-25 percent split. The graph shows that KNNBasic and the hybrid KNN performed the worst, with RMSE scores of above 0.9, the Hybrid lowers the RMSE but greatly increases the time to fit similar to what was observed for the KNN hybrid. Slope One and

Table 13: Top 5 recommendations for user 1 for SVD and SVD++ using their optimised parameters

SVD	SVD++
Toy Story (1995)	Toy Story (1995)
Secret Garden, The (1993)	Secret Garden, The (1993)
Mary Poppins (1964)	Mary Poppins (1964)
E.T. the Extra-Terrestrial (1982)	E.T. the Extra-Terrestrial (1982)
One Flew Over the Cuckoo's Nest	One Flew Over the Cuckoo's Nest

NMF(unbiased) return RSMEs close to 0.9 and both produce a quick fit time. SVD and KNNBaseline and Combiner all sit around the 0.86 mark with Combiner approximately the sum of both fit times, these algorithms take roughly 2-3 times as much as slope one and NMF while reducing RMSE by 0.5. Depending on the size of the training data, it could be beneficial to use a faster algorithm over a more accurate depending on the hardware available as time to fit will scale.

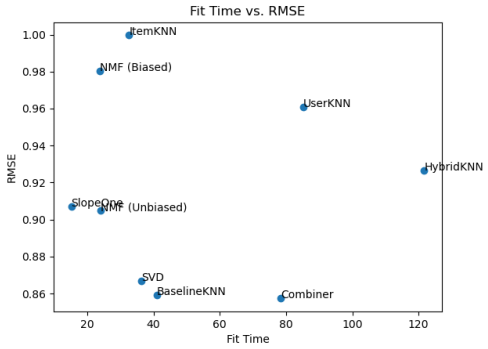

Figure 17: Results of RMSE against fit time for every algorithm used in this study

Figure 18 shows the same experiment results using prediction time instead of fit time. The results show a similar trend with KNN algorithms taking genuinely longer than the matrix factorization which would be due to calculate the distance to each instance in the trend set, once again a factor that will scale in line with dataset size. SVD appears to be the best algorithm for predictions once training is complete and combinations falls short due its KNN prediction component.

Overall the algorithm best fit for purpose for movie recommendations according to

Table 14: Results of weight combinations for combination algorithm

KNN_w	SVD_w	*RMSE*
0.25	0.75	0.8607
0.3	0.7	0.86
0.35	0.65	0.8594
0.4	0.6	0.8589
0.45	0.55	0.8585
0.5	0.5	0.8582
0.55	0.45	0.858
0.6	0.4	0.8579
0.65	0.35	0.8579
0.7	0.3	0.858
0.75	0.25	0.8582
0.8	0.2	0.8585

the results of this study is SVD. However, factors should be considered and a dataset that is constantly changing and requiring re training may benefit more from Slope One or NMF if the data is large where the fit time is significantly less for only a small rise in RMSE.

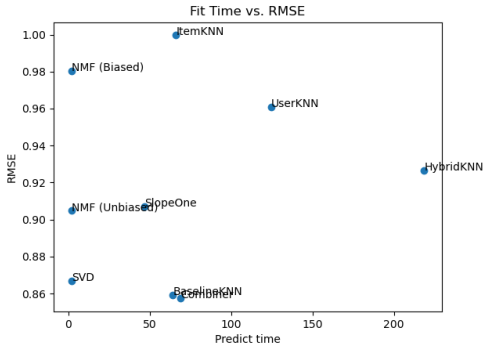

Figure 18: Results of RMSE against predict time for every algorithm used in this study

5. Conclusion and Future Work

This study has thoroughly examined the topic of machine learning techniques for recommendations systems, introduced the topic via the relevant background information required to understand on a high level, provided exploration of the MovieLens 100k to give a deeper understanding of the some of the demographic information that made up the userbase, the methodology section outlines the experiments that were conducted using various different algorithms and the evaluation section discuss the finding and decision making process in building various hybrids.

The background section introduces the different types of filtering that are most commonly used in recommendations, content based and collaborative filtering. The prior uses item features and user preferences to make recommendations, and the latter uses behavioral data such as ratings to based predictions of and therefore this study focused on collaborative algorithms as the movielens dataset provides explicit ratings and no item data and therefore content-based algorithms would not be appropriate in this scope. This section also provides a introduction to K- Nearest Neighbors algorithm as one of the primary models that would be evaluated and discussed different ways the algorithm can be figured such as using different similarity measures.

Hybrid approaches are also introduced as a way to attempt to solve many of the issues are faced using singular models, the hybrid approach provides a key point into one of the studies main objectives and this background forms a basis for the design decisions used in the implementation and evaluation.

The methods section of this study outlines how the aims and objectives will be bet, first the data is explored deeper to give a comprehensive overview of certain information about the users. This included various distributions of ratings for both the movielens 100k and 1m datasets and this showed that the datasets were similar and therefore allowed for cross dataset testing to be used which reduced the computational time that was necessary to evaluate the algorithms.

An overview of the evaluation process was also discussed which was 5-fold cross validation for the KNN analysis and the design decision for building a KNN hybrid by finding the optimal setups for similarity measures and k values. GridsearchCV was then used later for the KNNBaseline algorithm and all the matrix factorizations due to the high number of parameters available to change which made automating this process key to evaluation.

The evaluation of the k nearest neighbors provide valuable insights to the performance of this type of algorithm. Both user and item-based models using k=40 achieved very similar results which unsurprisingly beat the random predictor baseline scores. Using a user-item hybrid did yield a lower RMSE score, deeming more accurate, however due to the nature of the hybrid having to train each model and combine predictions, a high increase in time was recorded which made it less practical for application. Refactoring the hybrid via experimentation and improved components did again lead to a drop in RMSE however did not change the actual movies that would be predicted for a top-10 list.

Furthermore, evaluation on the generalized data highlighted the supreriority of the KNNBaseline model, which achieved far lower RMSE scored for both the ml100k and 1m datasets while producing a different top10 prediction list .The results found here determined the KNNBaseline would be the model used In the final hybrid later on.

The matrix factorization methods were all optimized using grid search for each algorithm. Using these results the models were trained and tested on the ml-100k and 1m again. The results show that all of SVD, SVD++ and NMF achieved similar results with SVD++ topping the group with the demerit of significantly increased computation time, and on the 100k dataset NMF (bias) scored very poorly being very close to the random baseline algorithm which contradicted the results of grid search.

Based on these results the SVD algorithm was combined with KNN baseline to create the final hybrid, and experimentation was carried out the determine the best weight ratio. The algorithm performed in the same pattern as the previous with a slight increase over the singular component.

Overall, it was determined that matrix factorization algorithms perform slightly better than K Nearest Neighbors for providing movie recommendation using the movielens dataset. It can also be inferred that although using a combination of models can produce marginally better RMSE scores however due to the additional computation cost, it is unlikely to be a worthwhile in most use cases. However, looking at all the algorithms together show differences in predict times and fit times which could suggest that while SVD was considered to be the best algorithm, depending on the exact problem and dataset, simpler models such as SlopeOne may be better.

5.1. Future Work

While this study has provided a valuable insight into different machine learning techniques for movie recommendations, due to limitations and the scope of this study there are several avenues for further research and exploration.

Advanced hybrid approaches. This study explored simply hybrid approached and only used a weighted model, many other classes of hybrids exist such as feature combination and content boosted; using item data such as genre, actors or keywords could provide extra insighs to match movies to a user, this could specifically help deal with sparsity where a user hasn't rated many movies. The use of deep learning could also improve performance and accuracy and would allow for extra data to be used as inputs.

This study primarily used RMSE as score to evaluate each algorithm, while this is satisfactory for this study, future work could include extra metrics such as precision and recall to determine how well the algorithms generates predictions. This can be taking further by splitting up groups of users with similar likes and looking into demographic data to determine how well each algorithm handles diversity.

Additional work could incorporate a more exhaustive evaluation of the algorithms, while this study aimed to comprehensibly evaluate each algorithm, computation limitations meant, methods such as Leave One Out Validation could not be used, which could have changed the results for different algorithms, also for the grid search, many parameter grids were shrunk due to the time the took to run and therefore it is possible the best parameters setups for SVD, KNNBaseline and others existed outside the parameters this study conducted evaluation on.

Finally, future work could aim to include user feedback and iteration and work to involve users users in the recommendation by using feedback loops or active learning approaches could greatly improve accuracy and prediction.

References

Al Hassanieh, L., Abou Jaoudeh, C., Abdo, J. B., and Demerjian, J. (2018). Similarity measures for collaborative filtering recommender systems. In *2018 IEEE Middle East and North Africa Communications Conference (MENACOMM)*, pages 1–5. IEEE.

Bennett, J., Lanning, S., et al. (2007). The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York.

Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370.

Chen, L. and Sycara, K. (1998). Webmate: A personal agent for browsing and searching. In *Proceedings of the second international conference on Autonomous agents*, pages 132–139.

Das, D., Sahoo, L., and Datta, S. (2017). A survey on recommendation system. *International Journal of Computer Applications*, 160(7).

Duan, R., Jiang, C., and Jain, H. K. (2022). Combining review-based collaborative filtering and matrix factorization: A solution to rating's sparsity problem. *Decision Support Systems*, 156:113748.

Goldberg, K., Roeder, T., Gupta, D., and Perkins, C. (2001). Eigentaste: A constant time collaborative filtering algorithm. *information retrieval*, 4(2):133–151.

Gower, S. (2014). Netflix prize and svd. *University of Puget Sound*.

Harper, K. (2016). Harper fm, konstan ja. *The movielens datasets: History and context, ACM Transactions on Interactive Intelligent Systems (TIIS)*, 5(4).

Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., et al. (2020). Array programming with numpy. *Nature*, 585(7825):357–362.

Hijikata, Y., Iwahama, K., and Nishida, S. (2006). Content-based music filtering system with editable user profile. In *Proceedings of the 2006 ACM symposium on Applied computing*, pages 1050–1057.

Hug, N. (2020). Surprise: A python library for recommender systems. *Journal of Open Source Software*, 5(52):2174.

Koohi, H. and Kiani, K. (2016). User based collaborative filtering using fuzzy c-means. *Measurement*, 91:134–139.

Koren, Y. (2010). Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 4(1):1–24.

Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37.

Lemire, D. and Maclachlan, A. (2005). Slope one predictors for online rating-based collaborative filtering. In *Proceedings of the 2005 SIAM International Conference on Data Mining*, pages 471–475. SIAM.

Liu, H., He, J., Wang, T., Song, W., and Du, X. (2013). Combining user preferences and user opinions for accurate recommendation. *Electronic Commerce Research and Applications*, 12(1):14–23.

Luo, X., Zhou, M., Xia, Y., and Zhu, Q. (2014). An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems. *IEEE Transactions on Industrial Informatics*, 10(2):1273–1284.

McKinney, W. et al. (2010). Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56.

McLaughlin, M. R. and Herlocker, J. L. (2004). A collaborative filtering algorithm and evaluation metric that accurately model the user experience. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 329–336.

Melville, P., Mooney, R. J., Nagarajan, R., et al. (2002). Content-boosted collaborative filtering for improved recommendations. *Aaai/iaai*, 23:187–192.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830.

Peterson, L. E. (2009). K-nearest neighbor. *Scholarpedia*, 4(2):1883.

Rich, E. (1979). User modeling via stereotypes. *Cognitive science*, 3(4):329–354.

Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295.

Thorat, P. B., Goudar, R. M., and Barve, S. (2015). Survey on collaborative filtering, content-based filtering and hybrid recommendation system. *International Journal of Computer Applications*, 110(4):31–36.

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

A. Cross Validation Testing

The full code for the methods of testing algorithm can be seen in Listing 7

```
    def testing_algorithm(algo, data):

    # Initialize arrays to store the scores
    fit_times, predict_times = np.array([])
    predict_times = np.array([])
    rmse_scores = np.array([])
    mse_scores = np.array([])
    mae_scores = np.array([])

    # Define the K-fold cross-validation iterator with 5 splits
    kf = KFold(n_splits=5, random_state=1)

    # Perform K-fold cross-validation
    for trainset, testset in kf.split(data):
        # Train the algorithm on the training set and measure the
    time taken
        start_fit = time.time()
        algo.fit(trainset)
        fit_times = np.append(fit_times, time.time() - start_fit)

        # Predict the ratings for the test set and measure the time
    taken
        start_predict = time.time()
        predictions = algo.test(testset)
        predict_times = np.append(predict_times, time.time() -
    start_predict)

        # Compute the RMSE, MSE, and MAE scores for the predictions
        rmse_scores = np.append(rmse_scores, rmse(predictions,
    verbose=False))
        mse_scores = np.append(mse_scores, mse(predictions, verbose=
    False))
        mae_scores = np.append(mae_scores, mae(predictions, verbose=
    False))

    # Return the mean of the RMSE, MSE, MAE,fit time and predict time
    return np.array(
```

```
32          [np.mean(rmse_scores), np.mean(mse_scores), np.mean(
    mae_scores), np.mean(fit_times), np.mean(predict_times)])
```

Listing 7: Function to return statistic on a recommendation algorithm

B. KNN Hybrid Solution

The code for the KNN final hybrid can be seen in Listing 8.

```
class UserItemKNNv2(AlgoBase):
    def __init__(self, i_weight=0.5, u_weight=0.5):
        AlgoBase.__init__(self)

        # create sim option dictionarys
        self.sim_options_user = {'name': 'Pearson', 'user_based':
    True}
        self.sim_options_item = {'name': 'cosine', 'user_based':
    False}
        # set weight variables
        self.i_weight = i_weight
        self.u_weight = u_weight
        # create reccomendation components
        self.user_algo = KNNBasic(k=55, sim_options=self.
    sim_options_user, verbose=False)
        self.item_algo = KNNBasic(k=60, sim_options=self.
    sim_options_item , verbose=False)

    def fit(self, trainset):
        # train both components
        AlgoBase.fit(self, trainset)
        self.user_algo.fit(trainset)
        self.item_algo.fit(trainset)

    def estimate(self, u, i):
        # generate predictions for each model and combine
        user_prediction = self.user_algo.predict(u, i).est
        item_prediction = self.item_algo.predict(u, i).est
        return self.u_weight * user_prediction + (1 - self.i_weight)
    * item_prediction


    def predict(self, uid, iid, r_ui=None, clip=True, verbose=False):
        # return prediction object if possible
        try:
            est = self.estimate(uid, iid)
            return Prediction(uid, iid, r_ui, est, details=None)
```

```
33          except PredictionImpossible as e:
34              if verbose:
35                  print(str(e))
36              return None
```

Listing 8: Class for the KNN hybrid solution

C. Prediction Comparisons for KNN algorithms

Table 15: Top 5 recommendations for user 134 returned by for KNNBasic using default setup and Hybrid.

Hybrid	KNN Basic
Braveheart (1995)	Braveheart (1995)
In the Line of Fire (1993)	In the Line of Fire (1993)
Last of the Mohicans, The (1992)	Last of the Mohicans, The (1992)
Austin Powers: IMoM (1997)	Austin Powers: IMoM (1997)
Full Monty, The (1997)	Full Monty, The (1997)

Table 16: Top 5 recommendations for user 398 for KNNBasic using default setup and Hybrid.

Hybrid	KNN Basic
Taxi Driver (1976)	Taxi Driver (1976)
Godfather, The (1972)	Godfather, The (1972)
Clockwork Orange, A (1971)	Dial M for Murder (1954)
Full Metal Jacket (1987)	Clockwork Orange, A (1971)
Chinatown (1974)	Chinatown (1974)

D. Prediction Comparisons for SVD and SVD++

Table 17: Top 5 recommendations for user 134 for SVD and SVD++.

SVD	SVD++
Toy Story (1995)	Toy Story (1995)
Braveheart (1995)	Braveheart (1995)
Forrest Gump (1994)	Forrest Gump (1994)
Aladdin (1992)	Aladdin (1992)
Sneakers (1992)	Sneakers (1992)

Table 18: Top 5 recommendations for user 398 for SVD and SVD++.

SVD	SVD++
Twelve Monkeys (1995)	Twelve Monkeys (1995)
Blade Runner (1982)	Blade Runner (1982)
Annie Hall (1977)	Annie Hall (1977)
Sting, The (1973)	Sting, The (1973)
Stand by Me (1986)	Stand by Me (1986)