University of Cape Town

soMLier: A South African Wine Recommender System

Author:
Josh Redelinghuys
RDLJOS002

Supervisor:
Dr. Şebnem Er

A Dissertation Presented for the Degree of

Master of Data Science

in the Department of Statistics

October 14, 2022

Abstract

Though several commercial wine recommender systems exist, they are largely tailored to consumers outside of South Africa (SA). Consequently, these systems are of limited use to novice wine consumers in SA.

To address this, the aim of this research is to develop a system for South African consumers that yields high-quality wine recommendations, maximises the accuracy of predicted ratings for those recommendations and provides insights into why those suggestions were made. To achieve this, a hybrid system "soMLier" (pronounced "sommelier") is built in this thesis that makes use of two datasets.

Firstly, a database containing several attributes of South African wines such as the chemical composition, style, aroma, price and description was supplied by wine.co.za (a SA wine retailer). Secondly, for each wine in that database, the numeric 5-star ratings and textual reviews made by users worldwide were further scraped from Vivino.com to serve as a dataset of user preferences. Together, these are used to develop and compare several systems, the most optimal of which are combined in the final system.

Item-based collaborative filtering methods are investigated first along with model-based techniques (such as matrix factorisation and neural networks) when applied to the user rating dataset to generate wine recommendations through the ranking of rating predictions. Respectively, these methods are determined to excel at generating lists of relevant wine recommendations and producing accurate corresponding predicted ratings.

Next, the wine attribute data is used to explore the efficacy of content-based systems. Numeric features (such as price) are compared along with categorical features (such as style) using various distance measures and the relationships between the textual descriptions of the wines are determined using natural language processing methods. These methods are found to be most appropriate for explaining wine recommendations.

Hence, the final hybrid system makes use of collaborative filtering to generate recommendations, matrix factorisation to predict user ratings, and content-based techniques to rationalise the wine suggestions made. This thesis contributes the "soMLier" system that is of specific use to SA wine consumers as it bridges the gap between the technologies used by highly-developed existing systems and the SA wine market. Though this final system would benefit from more explicit user data to establish a richer model of user preferences, it can ultimately assist consumers in exploring unfamiliar wines, discovering wines they will likely enjoy, and understanding their preferences of SA wine.

Dedication

I dedicate this thesis to my father who taught me to love and appreciate wine and inspired the aim of this thesis. I could not have undertaken this research without the solid foundations laid by the wealth of knowledge he shared with me.

Acknowledgements

I would like to extend my deepest gratitude to my supervisor Dr. Şebnem Er for her invaluable feedback, continued support and guidance, without which completing this thesis would not have been possible. I must also thank Birkir A. Barkarson from Vivino.com and Kevin Kidson from wine.co.za for providing access to their data which this research is based on.

Accomplishing this research was possible thanks to the motivation and encouragement of my loving family for which I am endlessly grateful.

Contents

List of Figures

List of Tables

List of Acronyms

AHM	Ahmad distance measure
ALS	Alternating Least Squares
(AR)HR	(Average Reciprocal) Hit Rate
BP	Back Propagation
CB	Content-Based
CDF	Cumulative Distribution Function
COS	Cosine similarity measure
COV	Coverage quality metric
CPCC	Constrained Pearson Correlation similarity measure
EUC	Euclidean similarity measure
FPR	False Positive Rate
GOW	Gower distance measure
HAR	Harikumar distance measure
HUA	Haung distance measure,
(IB/ UB)CF	(Item-Based/ User-Based) Collaborative Filtering
IQR	Inter-Quartile Range
JAC	Jaccard similarity measure
JRL	Joint Representation Learning
***k*NN**	k Nearest Neighbour(s)
MSD	Mean Square Difference similarity measure
NHSM	New Heuristic Similarity Measure
NN	Neural Network
PCC	Pearson Correlation similarity measure
PER	Personalisation quality metric
POD	Podani distance measure
RMSE	Root Mean Square Error
ROC	Receiver Operating Characteristic curve
RS	Recommender System(s)
(S)GD	(Stochastic) Gradient Descent
SA	South Africa(n)
TF-IDF	Term Frequency-Inverse Document Frequency
TPR	True Positive Rate
WIS	Wishart distance measure
WPC	Weighted Pearson Correlation similarity measure

Chapter 1

Introduction

The main aim of this research is to build a wine recommendation system that intelligently recommends wines to a user by exploiting the intrinsic similarities between wines and explicit user preferences.

Obtaining wine recommendations in South Africa is an imprecise endeavour (Tassiopoulos et al., 2004); suggestions are limited, inaccurate, arbitrary, or based solely on expert opinions (Smith, 2019). As a result, purchasing unfamiliar wines with confidence that they will be enjoyed is challenging in South Africa.

Wine is unique in that it cannot be tasted before buying online or in stores; this forces novice consumers to purchase wine based on other factors such as the design of the bottle, its price, awards it may have won and descriptions on the back label. Purchasing wine based on those features, and not on the attributes of the wine (flavour, aroma, body etc.) can result in the consumer making the wrong purchasing decisions (Smith, 2019).

Current wine recommendation applications such as Vivino (2022), WineRing (2021), Sippd (2022) and HelloVino (2022) make use of proprietary algorithms to generate wine suggestions to solve this issue. These applications, however, are greatly focused on wines produced in other regions such as France, Italy, Spain, and America and are of limited use in South Africa (Kotonya et al., 2018).

Furthermore, 42% of popular South African wine retailers do not provide recommendations on any of their products. The wine recommendations made by the remaining retailers (58%) lack insight and can often seem meaningless to the customer[1]. When exploring wines, knowing that two bottles are simply "related" or "frequently bought together" is insufficient as it does not indicate how or why the wines may be similar.

Wine, by nature, is extremely varied and nuanced. As a result, wines recommended simply because they were produced by the same farm or because they are of the same varietal ignore many of the features of wine that cause them to be similar (Johnson, 1991). Then, in other

[1]Data collected from a survey of 24 South African online wine retailers. 10 retailers offered no product recommendations, 6 offered "related product" suggestions, 4 indicated products which the user "might like", 3 recommended more products from the same producer, and 1 suggested other products which were "frequently bought together".

cases where wine suggestions are made using more complex strategies, little insight is provided into why that wine was recommended.

Therefore, wine suggestions can often seem arbitrary as they are unlike those produced by other recommendation systems in which the suggestions can be logically understood by the user (such as books written by the same author or movies with a similar cast, for example). This is significant when purchasing new wine as the customer should have confidence that they will enjoy it; wine need not be purchased solely on the notion that it is simply "related" to another wine – more data can and should be supplied when making purchasing decisions (Smith, 2019).

Beyond type and producer, wines can be considered similar based on their compositions, styles, flavour profiles, colours, aromas, descriptions, awards received, user ratings, reviews, food pairings and price. These features can be used to better explain why a certain wine was recommended than the less interpretable "you may also like" types of recommendations made currently.

To our knowledge, as of now, there are no recommendation systems that produce specifically South African wine suggestions that are accompanied by detailed information on why that suggestion was made. To address this, the proposed solution in this thesis incorporates three types of recommendation methodologies in a hybrid system named "soMLier" inspired by the terms "sommelier" and Machine Learning (ML):

1. **Collaborative Filtering Methods**

   Item-based collaborative filtering (IBCF) can be used to generate recommendations for a user based on their previous wine ratings recorded by the system (i.e., recommendations are made using a user's *implied* preferences) (Lü et al., 2012). The IBCF approach determines which wines are most similar to the wines previously enjoyed by a user and utilises them to generate wine suggestion lists (Melville and Sindhwani, 2010).

2. **Content-Based Methods**

   Content-based methods use the structured (such as price) and unstructured (such as descriptions) features of the wines to determine their similarity to one another; firstly, this is achieved through comparing the multiple numeric and categorical features (such as a wine's acidity and region of origin) using various equations which measure similarity. Next, the similarity between the textual description of two wines is determined using natural language processing methods (Pazzani and Billsus, 2007). Through ensembling the results of these similarity calculations, a database of wines which are closely related across multiple features can be generated and used to explain why certain wines are recommended.

3. **Model-Based Methods**

   Model-based methods such as matrix factorisation and neural networks utilise latent patterns in the ratings made by users to establish models that can accurately predict a user's future wine ratings. Matrix factorisation is used to compute user and wine latent factors which capture the affinity of the users to each of the wines in the database. Neural networks, on the other hand, receive and combine user rating and wine attribute data to extract latent representations (Zhang, Yao, et al., 2019). These latent entities are then

used to predict ratings that serve as estimations of the likelihood that a user would enjoy a given wine recommended to them (Ricci et al., 2011).

The optimal configurations of each of these methods are combined into the final hybrid system that leverages the advantages of each methodology; IBCF suggests wines to a user, model-based methods predicts the ratings the user would award them, and CB methods explain why those wines are recommended.

Each component of the final recommendation system is combined in a Shiny (Chang et al., 2021) web application that allows users to interact with the system by rating wines, receiving wine suggestions and discovering the reasoning behind their recommendations. Uniquely, this system provides explicit feedback on why a set of wines are recommended; specifically, a user may explore both textual and visual depictions of the similarities between their rated and recommended wines. The visualisations provided are in the form of networks that illustrate the relationships between wine attributes, three-dimensional plots of wine chemistry and depictions of the words most commonly used to describe their recommendations.

In this thesis, various recommender system methods are applied to the user rating and wine datasets; their strengths and weaknesses when used within the context of wine data are investigated before combining the optimal models into the final system. This thesis contains 5 remaining chapters. Chapter 2 summarises recommender system methodologies and existing wine recommender applications. Chapters 3 and 4 cover the data and methods used in this research, respectively. Chapter 5 presents the results and discussion of each method and the details of the final proposed wine recommender. Chapter 6 contains conclusions and future work.

Chapter 2

Recommender Systems

Recommender Systems (RS) have been used for decades to generate recommendations of items that a user may be interested in. Recently, data on users' demographics, consumption and preferences have become more readily available to large companies as shopping, networking and entertainment have moved online. As a result, data such as users' preferences and interactions with products has become more easily accessible. Hence, many RS have been developed to leverage this data to make personalised item suggestions to improve the user experience (in the case of movie recommendations such as those made by Netflix) or increase sales (in the case of suggested products made by online retailers like Amazon) (Lü et al., 2012).

Some of the earliest RS relied on knowledge engineering or explicit recommendations made by experts to produce a set of suggestions for a user (Burke, 1999b). These were quickly superseded by systems that exploited the intrinsic similarity between users or items to make recommendations. Assuming that a user's past and future preferences are correlated, RS are now able to generate accurate recommendations for a diverse array of users and items using various techniques (Melville and Sindhwani, 2010).

This chapter summarises the core concepts used in recommender systems and their applications in the wine industry. The synopsis of wine recommenders begins with rudimentary systems developed in the late 1990s and concludes with summaries of advanced, hybrid applications which are popular at the time of this research.

2.1 System Types

As illustrated in Figure 2.1, there exists three primary types of recommender systems: Collaborative Filtering (CF) systems, Content-Based (CB) systems and Hybrid systems (Lü et al., 2012). Each of these may utilise explicit preference data, such as user ratings, to generate recommendations.

User ratings are numeric assessments made of items, typically on a scale from 1 to 5, 10, or 100. These can be recorded in a sparse ratings matrix $\boldsymbol{R}_{n \times m}$ consisting of n users and m items as seen in Definition 1 where each r_{ij} corresponds to the rating made by user i on item j.

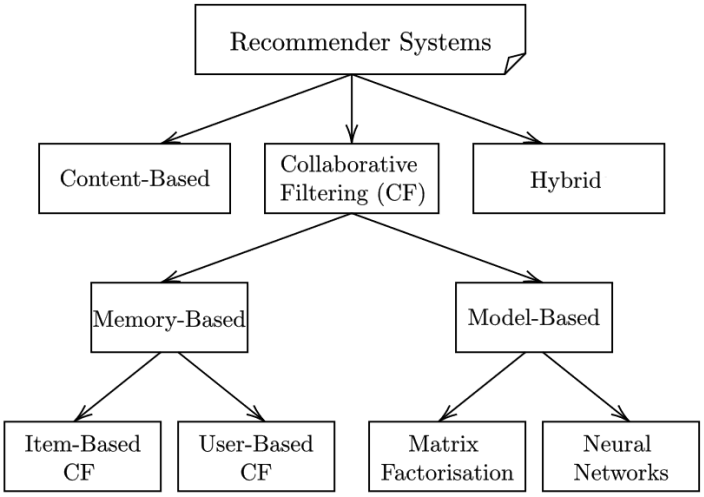

Figure 2.1: Categorisation of types of recommender systems.

Definition 1. The sparse rating matrix:

$$\boldsymbol{R}_{n\times m} = \begin{bmatrix} r_{11} & \cdots & r_{1m} \\ \vdots & \ddots & \vdots \\ r_{n1} & \cdots & r_{nm} \end{bmatrix}$$

n represents the number of users who have previously interacted with a system by providing some form of explicit rating feedback and could be as large as hundreds of millions. m is the number of items that a user may rate and that a system may recommend (such as the inventory size of a business) which may account for millions of items. This has practical implications as performing operations on such huge matrices are too computationally expensive. Hence, in this research, the restrictions $n < 100,000$ and $m < 2,000$ are applied.

$\boldsymbol{R}$ is sparse as users typically rate very few items and thus many values are missing and recorded as NA. Hence, the set of observed ratings $\boldsymbol{R}^+ = \{(i,j) : r_{ij} \text{ is observed}\}$ and the set of missing ratings $\boldsymbol{R}^- = \{(i,j) : r_{ij} \text{ is not observed}\}$ are defined for all users i and items j. Note that $\boldsymbol{R}^+ \subset \boldsymbol{R}$, $\boldsymbol{R}^+ < \boldsymbol{R}^-$ and $\boldsymbol{R}^+ \cap \boldsymbol{R}^- = \varnothing$.

$\boldsymbol{R}_{i\cdot}$ denotes the i^{th} row of $\boldsymbol{R}$ which corresponds to the ratings made by user i for all items. Thus, $\boldsymbol{R}_{i\cdot}^+$ denotes the set of observed ratings made by user i and therefore only contains numeric values and no missing observations. Likewise $\boldsymbol{R}_{\cdot j}$ is the j^{th} item column of $\boldsymbol{R}$ which contains missing values and $\boldsymbol{R}_{\cdot j}^+$ is the complete set of observed ratings for item j.

The purpose of the following systems described is to predict the unobserved ratings in $\boldsymbol{R}$ to generate recommendations. Typically, each system applies a unique set of methods to make predictions $\hat{r}_{ij}$ of the observed ratings and, assuming that the error between the observed and predicted ratings is low, the same techniques can be applied to predict the missing values in $\boldsymbol{R}$. Then, the unrated items predicted to be awarded the highest ratings are recommended to the user.

The first commercial recommender system, *Tapestry* (Goldberg et al., 1992), originated the

term "collaborative filtering" as it required the collaboration of data from multiple users to generate recommendations. CF systems produce recommendations by exploiting similarities in the historical patterns of the ratings made by multiple users and can be categorised into either memory-based or model-based systems. Memory-based CF approaches make use of statistical methods to compute user or item similarities which are in turn used to generate recommendations (Aditya et al., 2017). Model-based CF, however, involves the training of models to identify user rating patterns which can be used to extract recommendations (Lü et al., 2012).

Contrarily, content-based methods make use of user and item *attributes* instead of ratings to generate recommendations. Structured CB systems are developed using a dense attribute matrix $\boldsymbol{A}_{m\times o}$ as seen in Definition 2 consisting of m item rows and o distinct item attributes where each a_{ij} corresponds to the value of attribute j for item i. Unstructured CB systems, on the other hand, make use of the free-text (such as descriptions or reviews) related to the items.

Definition 2. The dense item attribute matrix:

$$\boldsymbol{A}_{m\times o} = \begin{bmatrix} a_{11} & \cdots & a_{1o} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mo} \end{bmatrix}$$

Hybrid systems combine the various approaches of CF and CB systems in such a way that capitalises on the individual strengths and mitigates the limitations of the different methods used (Burke, 2007).

Both memory-based CF and CB systems operate on the assumption that there exists intrinsic similarities between user and item entities. On the other hand, the core assumption in model-based techniques is that similarities between entities are the result of lower-dimensional patterns that exist within rating data (Melville and Sindhwani, 2010).

2.1.1 Measuring similarity

Memory-based CF and content-based systems rely on the intrinsic similarity between users or items to make suggestions. The generic similarity matrix $\boldsymbol{S}_{p\times q}$ as seen in Definition 3 contains the similarities s_{ij} between every pair of elements i and j.

Definition 3. The dense similarity matrix:

$$\boldsymbol{S}_{p\times q} = \begin{bmatrix} s_{11} & \cdots & s_{1q} \\ \vdots & \ddots & \vdots \\ s_{p1} & \cdots & s_{pq} \end{bmatrix}$$

In the case of a *user* similarity matrix $\boldsymbol{S}^{u}_{n\times n}$, $p = q = n$ and the similarity values are symmetrical about the diagonal such that $s_{ij} = s_{ji}$ and $s_{ij} = 1$ when $i = j$. The same is true for an *item* similarity matrix $\boldsymbol{S}^{v}_{m\times m}$ where $p = q = m$, the diagonal values are 1 and $s_{ij} = s_{ji}$. A user-item similarity matrix $\boldsymbol{S}^{uv}_{n\times m}$ $(p = n, q = m)$ is not symmetrical as each value s_{ij} equates to the similarity of user i to item j (Aggarwal, 2016).

In CF systems, $\boldsymbol{S}^{u}_{n\times n}$ and $\boldsymbol{S}^{v}_{m\times m}$ may be computed by applying *similarity measures* to the rows and columns of $\boldsymbol{R}$, respectively. Alternatively, in the context of structured item attributes in CB systems, an item similarity matrix $\boldsymbol{S}^{v:s}_{m\times m}$ may be computed by applying *distance measures* to the structured rows of $\boldsymbol{A}$. Likewise, $\boldsymbol{S}^{v:\not s}_{m\times m}$ is an item similarity matrix that may be computed using natural language processing applied to unstructured item data. The combined user-item similarity matrix $\boldsymbol{S}^{uv}_{n\times m}$, however, requires slightly more CB computations which are detailed in Section 2.1.3.

Importantly, similarity measures are used to establish the relationships between numeric vectors whereas distance measures are used with mixed data types. A selection of eight popular similarity measures used to calculate user and item similarity based on rating patterns are investigated in this thesis. These measures explored are the Euclidean (EUC), Pearson Correlation (PCC), Constrained Pearson Correlation (CPCC), Cosine (COS), Mean Square Difference (MSD), Jaccard (JAC), Weighted Pearson Correlation (WPC), and the New Heuristic Similarity Measure (NHSM). Further details regarding the formulation of these measures can be found in Appendix A.1.

The similarity between items can be determined using their attributes (or content) instead of rating patterns. Often, both the factor and numeric variables recorded for each item need to be compared together to determine the relationships between the items. As only numeric data can be used with the similarity measures mentioned above, other distance measures are required to establish similarity between mixed data consisting of both factor and numeric data. The specifics of the Gower (GOW), Wishart (WIS), Podani (POD), Haung (HUA), Ahmad (AHM), and Harikumar (HAR) distance measures explored in this thesis can be found in Appendix A.2.

2.1.2 Memory-based collaborative filtering

As seen in Figure 2.1, memory-based CF may be approached from a user perspective (user-based CF or UBCF) where items recommended are those which are preferred by similar users, or from an item perspective (item-based CF or IBCF) where items that are similarly rated to those preferred by the user are recommended (Melville and Sindhwani, 2010). Henceforth, *CF* will refer to memory-based CF exclusively, while model-based methods will be explicitly specified.

CF systems produce recommendations through the ranking of predicted ratings. The system computes the ratings a user would award to items not yet rated and those with the highest predicted ratings are then recommended to the user. For both user and item-based CF systems, these predicted ratings can be calculated using a similarity vector $\boldsymbol{s}$ and a rating vector $\boldsymbol{r}$, both of which are different depending on the choice of the user or item-based approach (Aggarwal, 2016).

To illustrate how predicted ratings are computed, consider the ratings matrix in Example 2.1.1. The ratings made by users $i \in \{1, 2, 3\}$ on items $j \in \{1, 2, 3, 4\}$ are stored in $\boldsymbol{R}$ where r_{34} is missing and to be predicted.

Example 2.1.1.

$$\boldsymbol{R} = \begin{array}{c} \\ i=1 \\ i=2 \\ i=3 \end{array} \begin{array}{c} \begin{array}{cccc} j=1 & j=2 & j=3 & j=4 \end{array} \\ \begin{bmatrix} 5 & 4 & 1 & 4 \\ 1 & 2 & 3 & 1 \\ 4 & 4 & 2 & \text{NA} \end{bmatrix} \end{array}$$

2.1.2.1 User-based CF

In user-based CF, the similarity matrix is computed by comparing the ratings that each user made (the rows $\boldsymbol{R}^+_{1\cdot}$, $\boldsymbol{R}^+_{2\cdot}$ and $\boldsymbol{R}^+_{3\cdot}$) for the set of items that they rated in common using any similarity measure discussed in Appendix A.1. Using the Euclidean similarity measure in equation (A.1.1), the symmetrical user similarity matrix $\boldsymbol{S}^u$ is calculated as:

Example 2.1.2.

$$\boldsymbol{S}^u = \begin{array}{c} \\ i=1 \\ i=2 \\ i=3 \end{array} \begin{array}{c} \begin{array}{ccc} i=1 & i=2 & i=3 \end{array} \\ \begin{bmatrix} 1 & 0.15 & 0.41 \\ 0.15 & 1 & 0.21 \\ 0.41 & 0.21 & 1 \end{bmatrix} \end{array}$$

noting that s^u_{13} and s^u_{23} are computed using the ratings made for the co-rated items $j \in \{1, 2, 3\}$ only.

The vector $\boldsymbol{s}^u_i = \{s^u_{i1}, \ldots, s^u_{ik}\}$ is defined as that which contains the similarities between the target user i (the user for which the system will produce recommendations) and the k most similar other users in the system. Optimal values for k are to be determined through experimentation but smaller values are preferred to larger as increases in k may result in dissimilar users being included in $\boldsymbol{s}^u_i$ which, in turn, diminishes the accuracy of predictions made. The values of $\boldsymbol{s}^u_i$ are extracted from $\boldsymbol{S}^u_{i\cdot}$ such that $\{s_{i1} \geq s_{i2} \geq \cdots \geq s_{ik}\}$. From Example 2.1.2, $\boldsymbol{s}^u_3 = \{0.41, 0.21\}$ when $k = 2$.

Next, the rating vector $\boldsymbol{r}^u_j$ is extracted from $\boldsymbol{R}^+_{\cdot j}$ and contains the ratings for item j awarded by each user in $\boldsymbol{s}^u_i$. Once $\boldsymbol{s}^u_i$ is formed, it must be weighted to produce $\dot{\boldsymbol{s}}^u_i$ such that $\sum \dot{\boldsymbol{s}}^u_i = 1$. This ensures that the product of $\dot{\boldsymbol{s}}^u_i$ and $\boldsymbol{r}^u_j$ is a weighted average of $\boldsymbol{r}^u_j$. For measures which generate similarity values in the range [0,1], weighting is achieved by dividing by the sum of the similarities, $\dot{\boldsymbol{s}}^u_i = \boldsymbol{s}^u_i / \sum \boldsymbol{s}^u_i$. For the Pearson measures (and the cosine measure in the case of negative ratings) which yield similarity values in [-1,1], $\dot{\boldsymbol{s}}^u_i$ is weighted using the softmax function instead:

$$\dot{\boldsymbol{s}}^u_i = \frac{e^{\boldsymbol{s}^u_i}}{\sum e^{\boldsymbol{s}^u_i}} \tag{2.1.1}$$

When applying softmax, negative similarity scores are scaled to very small values (typically 0.1) and positive scores are scaled to values closer to 1.

Then, the similarity vector is linearly combined with the rating vector to produce the user-based predicted rating $\hat{r}^u_{ij}$ that user i would give to an item j (Aggarwal, 2016):

$$\hat{r}^u_{ij} = \dot{\boldsymbol{s}}^u_i \cdot \boldsymbol{r}^u_j \tag{2.1.2}$$

Assuming rating values in $\boldsymbol{r}$ are between 1 and 5, the dot product $\dot{\boldsymbol{s}}^u_i \cdot \boldsymbol{r}^u_j$ would produce a weighted rating value between 1 and 5. This rating is equivalent to the average rating awarded to item j by the users most similar to i, weighted by how similar their rating patterns are to

i. This predicted rating can then be interpreted as the likelihood that i would enjoy item j as defined by the preferences of users similar to i (Aggarwal, 2016).

The predicted rating for r_{34} is then computed as in Example 2.1.3.

Example 2.1.3.

$$\begin{aligned} \boldsymbol{r}_4^u &= \{4, 1\} \\ \boldsymbol{s}_3^u &= \{0.41, 0.21\} \text{ then,} \\ \dot{\boldsymbol{s}}_3^u &= \{0.41, 0.21\}/(0.41 + 0.21) = \{0.66, 0.34\} \\ \therefore \hat{r}_{34}^u &= \dot{\boldsymbol{s}}_3^u \cdot \boldsymbol{r}_4^u = \{0.66, 0.34\} \cdot \{4, 1\} = 3 \end{aligned}$$

2.1.2.2 Item-based CF

Item-based collaborative filtering applies the same principles as user-based CF, except that similarities are calculated between *items* and not users. In this approach, items are determined to be similar if the users who co-rated them awarded them similar ratings. As a result, item recommendations for the target user are those which are most similar to the items already rated by the user (Melville and Sindhwani, 2010).

The item-based similarity matrix $\boldsymbol{S}^v$ is calculated by applying the similarity measures in Appendix A.1 to pairs of columns $\boldsymbol{R}_{\cdot j}^+$. Applying the Euclidean similarity to items in Example 2.1.1, produces the similarity matrix $\boldsymbol{S}^v$ in Example :

Example 2.1.4.

$$\boldsymbol{S}^v = \begin{array}{c} \\ j=1 \\ j=2 \\ j=3 \\ j=4 \end{array} \begin{array}{c} \begin{array}{cccc} j=1 & j=2 & j=3 & j=4 \end{array} \\ \begin{bmatrix} 1 & 0.41 & 0.17 & 0.5 \\ 0.41 & 1 & 0.21 & 0.5 \\ 0.17 & 0.21 & 1 & 0.22 \\ 0.5 & 0.5 & 0.22 & 1 \end{bmatrix} \end{array}$$

noting that s_{j4}^v values are computed using the ratings made by users $i \in \{1, 2\}$ as they were the only users who rated item 4.

In the item-based approach, the similarity vector $\boldsymbol{s}_j^v = \{s_{j1}^v, \ldots, s_{jk}^v\}$ contains the similarities between an item j and the k most similar other items rated by i and is extracted from $\boldsymbol{S}^v$ such that $\{s_{j1}^v \geq s_{j2}^v \geq \cdots \geq s_{jk}^v\}$.

Next, the rating vector $\boldsymbol{r}_i^v$ is extracted from $\boldsymbol{R}_{i\cdot}^+$ such that it contains all the ratings i made for the k items in $\boldsymbol{s}_j^v$. After weighting $\boldsymbol{s}_j^v$ so that $\sum \dot{\boldsymbol{s}}_j^v = 1$ as in Section 2.1.2.1, the item-based predicted rating $\hat{r}_{ij}^v$ is calculated as

$$\hat{r}_{ij}^v = \dot{\boldsymbol{s}}_j^v \cdot \boldsymbol{r}_i^v \tag{2.1.3}$$

and is interpreted as a measure of how likely i would enjoy j as influenced by their preferences of items similar to j (Aggarwal, 2016). In a similar manner to Example 2.1.3, if $\dot{\boldsymbol{s}}_4^v = \{0.5, 0.5\}$ when $k = 2$ and $\boldsymbol{r}_3^v = \{4, 4\}$, then $\hat{r}_{34}^v = 4$.

Although this method is likely to lead to more obvious recommendations than its user-based counterpart as it considers only the user's ratings (as opposed to those of a neighbourhood of similar users), it has been shown to be more accurate than user-based CF (Aggarwal, 2016).

This is in part due to the focus placed on the user's own ratings and in part due to the robustness of item-based CF to rating changes.

The similarity between two items is more static than the similarity between two users. Typically, in systems where there are many more users than items, two items will likely share a large number of users who co-rated them. Therefore, as more ratings are made for a pair of items, the similarity between the items is not likely to change significantly. However, when comparing users in the common case that few items have been rated by both users, new rating data will have a larger impact on the similarity between the users (Lü et al., 2012).

Additionally, item-based CF is more computationally efficient (Melville and Sindhwani, 2010). In user-based CF every user has to be compared to every other user - in real-life systems with millions of users, this quickly becomes infeasible (especially when user similarities should be updated regularly to account for new rating data). Contrarily, the set of items need only be compared to themselves in item-based CF, a much smaller set of calculations that needs to be updated far less frequently.

2.1.3 Content-based methods

In contrast to collaborative filtering methods, CB methods generate predictions by mapping the preferences of users to the attributes of items (Lops, 2011). Where CF locates similar users or items through rating data, CB systems only examine the content of items to determine similarity (Mooney and Roy, 2000).

As CF systems recommend the items that were well-rated by peers (user-based) or which were similar to the items well-rated by the user (item-based), often the most popular items overall are recommended. The reliance on user rating history thus prevents unpopular, or rare items from being recommended in CF systems (Mooney and Roy, 2000).

This is not the case for CB systems; as only the user's preferences and item attributes are required, niche items can be recommended to new users along with rich explanations as to why those items were suggested (Lops, 2011).

CB systems depend on item attributes to establish their similarity to one another in order to generate recommendations. Many item domains are considered semi-structured, where some attributes are described by a known set of finite values (such as car colour, shoe size, etc.), and others are completely unstructured - mainly free text consisting of unlimited, undefined values. While structured attributes can be managed neatly and compared with well-defined distance measures (see Appendix A.2 for details), unstructured text must be converted into a structured form before any further analysis (Pazzani and Billsus, 2007).

For structured item attributes[1], an item similarity matrix $\boldsymbol{S}^{v:s}$ may be created by comparing pairs of rows of the item attribute matrix $\boldsymbol{A}$ as in Definition 2 using the distance measures described in Appendix A.2. While those distance measures are capable of handling mixed data types (numeric and categorical variables), they cannot handle unstructured data.

In the case of unstructured item data such as textual descriptions, Term Frequency - Inverse Document Frequency (TF-IDF) calculations can be used to produce a structured equivalent.

[1]The superscript $^{v:s}$ represents a similarity matrix created with structured data and $^{v:\not s}$ reflects a similarity matrix created with unstructured data.

The procedure used to convert a text document d to a numeric vector $\boldsymbol{d}$ which represents the relevance of a set of terms to that document is detailed in Appendix A.3. Once each item is represented by a structured TF-IDF vector, they can be compared using any similarity measure described in Appendix A.1. The similarity between all pairs of documents describing items is then combined in the unstructured item similarity matrix $\boldsymbol{S}^{v:\not s}$.

Producing content-based recommendations

Once an item similarity matrix has been established, it may be used to generate recommendations in two ways. Firstly, the $\boldsymbol{S}^{v:s}$ and $\boldsymbol{S}^{v:\not s}$ matrices can be used as in the IBCF approach (instead of $\boldsymbol{S}^{v}$ built using rating data in Section 2.1.2.2) to predict ratings for unrated items and recommend those with the greatest predictions. Secondly, clustering and nearest neighbour methods can be applied to $\boldsymbol{S}^{v:s}$ and $\boldsymbol{S}^{v:\not s}$ to determine which items are most similar. Then, the items most similar to those highly rated by a user can be suggested (Pazzani and Billsus, 2007).

Alternatively, items can be recommended by pairing user *profiles* to items in a user-item similarity matrix $\boldsymbol{S}^{uv}$. Generating a user profile involves establishing the user's preferences; this may be done explicitly through fillable forms and personalisable profiles, or implicitly by analysing a user's rating history. A user profile $\boldsymbol{x}_i$ for user i who rated the set of items V_i may be calculated as:

$$\boldsymbol{x}_i = \sum_{j \in V_i} r_{ij} \boldsymbol{y}_j \tag{2.1.4}$$

where r_{ij} is the rating awarded by user i to item j and $\boldsymbol{y}_j = \{\omega_{1_j}, \omega_{2_j}, \ldots, \omega_{Pj}\}$ is an item profile where each value $\omega_{\varrho j}$ corresponds to the relevance of word t_ϱ in the document describing item j (refer to the formation of TF-IDF vectors detailed in Appendix A.3) (Ricci et al., 2011).

Then, $\boldsymbol{S}^{uv}_{n \times m}$ can be established by computing the similarity (using a measure defined in Appendix A.1) between each user profile $\boldsymbol{x}_i \forall i \in \{1, \ldots, n\}$ and every item $\boldsymbol{y}_j \forall j \in \{1, \ldots, m\}$. Using $\boldsymbol{S}^{uv}$, recommendations can be made by suggesting the items with the highest similarity values in $\boldsymbol{S}^{uv}_{i\cdot}$. However, performing $n \times m$ similarity calculations is very computationally expensive and a procedure which is not easily scalable (Aggarwal, 2016).

2.1.4 Model-based methods

Model-based methods also make use of the ratings matrix $\boldsymbol{R}$ but they do not exploit any forms of underlying user/ item similarities. Instead, rating predictions are computed using statistical models and the items suggested are those with the greatest predicted ratings. Two popular approaches, matrix factorisation (or decomposition) and neural networks have been shown to produce strong recommendation systems (Aggarwal, 2016).

2.1.4.1 Matrix factorisation

Matrix factorisation is the process of decomposing one matrix into (typically) two other latent matrices, the product of which is an approximation of the original. These decomposed matrices are considered lower-dimensional representations of the original matrix which capture an underlying structure in the data. This approach to recommender systems was largely popularised by the winners of the Netflix prize (Bennett et al., 2007) who used matrix factorisation methods to

produce a system which predicted a set of user ratings with a root mean square error of 9.46% less than Netflix's existing recommendation engine (Bell et al., 2009).

Matrix factorisation is a technique well-utilised by recommender systems as though the original matrix $\boldsymbol{R}$ (see Definition 1) contains many missing values, the decomposed matrices are complete. Thus, when recomposing the latent matrices, estimations of the missing values in $\boldsymbol{R}$ are produced, which in turn serve as predicted ratings (Aggarwal, 2016).

Formally, the ratings matrix $\boldsymbol{R}_{n\times m}$ can be approximately factorised into the matrices $\boldsymbol{U}_{n\times x}$ and $\boldsymbol{V}_{m\times x}$ such that

$$\boldsymbol{R} \approx \boldsymbol{U}\boldsymbol{V}^T \tag{2.1.5}$$

where

$$\boldsymbol{U} = \begin{array}{c} \\ \boldsymbol{U}_{1\cdot} \\ \vdots \\ \boldsymbol{U}_{n\cdot} \end{array} \begin{array}{c} \begin{array}{ccc} c=1 & \dots & c=x \end{array} \\ \begin{bmatrix} u_{11} & \cdots & u_{1x} \\ \vdots & \ddots & \vdots \\ u_{n1} & \cdots & u_{nx} \end{bmatrix} \end{array} \qquad \boldsymbol{V}^T = \begin{array}{c} \begin{array}{ccc} \boldsymbol{V}_{\cdot 1}^T & \dots & \boldsymbol{V}_{\cdot m}^T \end{array} \\ \begin{array}{c} c=1 \\ \vdots \\ c=x \end{array} \begin{bmatrix} v_{11} & \cdots & v_{1m} \\ \vdots & \ddots & \vdots \\ v_{x1} & \cdots & v_{xm} \end{bmatrix} \end{array} \tag{2.1.6}$$

$\boldsymbol{U}_{i\cdot}$ represents the i^{th} row of $\boldsymbol{U}$ and is known as a user factor that captures user i's affinity to the x underlying concepts in $\boldsymbol{R}$. Likewise, $\boldsymbol{V}_{\cdot j}^T$ is the j^{th} column of $\boldsymbol{V}^T$ (an item factor) that represents the relationship between item j and the x concepts in $\boldsymbol{R}$. The dimension of these latent factors, x, is used to control the number of underlying components in the data which are captured; for example, in simpler cases, $x = 2$ might represent fiction and non-fiction concepts in a book recommendation system.

Assuming $\boldsymbol{U}$ and $\boldsymbol{V}$ approximate $\boldsymbol{R}$ accurately, they can be used to predict the ratings for the items that are unrated by the users. The predicted rating of user i on item j is calculated as:

$$\begin{aligned} \hat{r}_{ij} &= \boldsymbol{U}_{i\cdot}\boldsymbol{V}_{\cdot j}^T \\ &= \sum_{c=1}^{x} u_{ic} \cdot v_{jc} \\ &= \sum_{c=1}^{x} (\text{Affinity of user } i \text{ to concept } c) \cdot (\text{Affinity of item } j \text{ to concept } c) \end{aligned} \tag{2.1.7}$$

Using this, the unrated items which are predicted to be awarded the greatest ratings by each user can then be recommended.

Calculating the latent matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ involves minimising the loss function:

$$\begin{gathered} J(\boldsymbol{U}, \boldsymbol{V}) = \frac{1}{2}||\boldsymbol{R} - \boldsymbol{U}\boldsymbol{V}^T||^2 \\ \text{subject to } \boldsymbol{U}, \boldsymbol{V} \geq 0 \end{gathered} \tag{2.1.8}$$

which calculates the square errors between the observed and estimated matrices where $\boldsymbol{U}$ and $\boldsymbol{V}$ are restricted to non-negative values (Aggarwal, 2016).

As values in $\boldsymbol{R}$ are missing, the minimisation function is restricted to only the rating values which are observed, $\boldsymbol{R}^+$. The minimisation function then becomes (Kumar et al., 2021):

$$J(\boldsymbol{U}, \boldsymbol{V}) = \frac{1}{2} \sum_{(i,j)\in\boldsymbol{R}^+} (r_{ij} - \hat{r}_{ij})^2 = \frac{1}{2} \sum_{(i,j)\in\boldsymbol{R}^+} \left(r_{ij} - \sum_{c=1}^{x} u_{ic} \cdot v_{jc}\right)^2 \tag{2.1.9}$$

As the set of ratings observed $\boldsymbol{R}^+$ is typically very small, minimising the above function may often lead to overfitting of the training data. A regularisation parameter λ is introduced to the minimisation equation to reduce the magnitudes of the factors, thereby generating simpler, more generalisable latent representations:

$$\begin{aligned} J(\boldsymbol{U}, \boldsymbol{V}) &= \frac{1}{2} \sum_{(i,j) \in \boldsymbol{R}^+} \Big(r_{ij} - \sum_{c=1}^{x} u_{ic} \cdot v_{jc} \Big)^2 + \frac{\lambda}{2} (||\boldsymbol{U}|| \cdot ||\boldsymbol{V}||)^2 \\ &= \frac{1}{2} \sum_{(i,j) \in \boldsymbol{R}^+} \Big(r_{ij} - \sum_{c=1}^{x} u_{ic} \cdot v_{jc} \Big)^2 + \frac{\lambda}{2} \sum_{i=1}^{n} \sum_{c=1}^{x} u_{ic}^2 + \frac{\lambda}{2} \sum_{j=1}^{m} \sum_{c=1}^{x} v_{jc}^2 \end{aligned} \tag{2.1.10}$$

where $\lambda \geq 0$ controls the amount of regularisation (Aggarwal, 2016).

Solving for the $\boldsymbol{U}$ and $\boldsymbol{V}$ factors and minimising the loss function above can be achieved using Singular Value Decomposition (SVD) when the matrix $\boldsymbol{R}$ is complete. However, as $\boldsymbol{R}$ consists of mainly missing values, Gradient Descent (GD) techniques are used to solve the minimisation problem instead (Meira et al., 2018).

Gradient descent methods are used to compute optimal values for a set of parameters such that a minimum value of a loss function of those parameters is obtained. This process iteratively updates the parameter values using the gradient of the loss function for a given set of parameter values. By descending the loss function in increments of its gradient, the global minimum of the loss function can (theoretically) be found (Hastie et al., 2009).

For example, let $\boldsymbol{w}$ be the set of parameters to be optimised and $\nabla \boldsymbol{j}$ be a vector containing the gradient of the loss function for each parameter in $\boldsymbol{w}$. Then, the values of $\boldsymbol{w}$ are optimised iteratively according to:

$$\boldsymbol{w} \coloneqq \boldsymbol{w} - \alpha \cdot \nabla \boldsymbol{j} \tag{2.1.11}$$

where $\alpha > 0$ is the learning rate used to control the rate at which the gradient of the loss function is descended. Larger learning rates result in larger changes in parameter values between iterations which may cause the gradient descent to miss the global minimum. Alternatively, smaller learning rates slow the learning process, and thus α must be selected through hyperparameter tuning.

Determining $\nabla \boldsymbol{j}$, the gradient of the loss function for a given set of parameters, involves calculating the partial derivatives of the loss function J with respect to the parameters $\boldsymbol{w}$. In the context of matrix factorisation, $\boldsymbol{w}$ is of length $(nx + mx)$ and contains all the values u_{ic} and v_{jc} of the latent factor representations. Computing $\nabla \boldsymbol{j}$ for these values requires the partial derivatives of equation (2.1.10) with respect to both the user and item latent factor values (Aggarwal, 2016):

$$\begin{aligned} \frac{\partial J}{\partial u_{ic}} &= \sum_{(i,j) \in \boldsymbol{R}^+} \Big(r_{ij} - \sum_{c=1}^{x} u_{ic} \cdot v_{jc} \Big) \Big(- v_{jc} \Big) + \lambda u_{ic} \\ \frac{\partial J}{\partial v_{jc}} &= \sum_{(i,j) \in \boldsymbol{R}^+} \Big(r_{ij} - \sum_{c=1}^{x} u_{ic} \cdot v_{jc} \Big) \Big(- u_{ic} \Big) + \lambda v_{jc} \end{aligned} \tag{2.1.12}$$

noting that the derivatives are computed for a given user or item and concept c. Then, if $e_{ij} = r_{ij} - \sum_{c=1}^{x} u_{ic} \cdot v_{jc}$ is defined as the error between the observed rating and $\hat{r}_{ij}$, using

equation (2.1.11), the value updates for each u_{ic} and v_{jc} can be computed as (Aggarwal, 2016):

$$\begin{aligned} u_{ic} &\coloneqq u_{ic} + \alpha\left(\sum_{(i,j)\in \boldsymbol{R}^+} (e_{ij})(v_{jc}) - \lambda u_{ic}\right) \\ v_{ic} &\coloneqq v_{ic} + \alpha\left(\sum_{(i,j)\in \boldsymbol{R}^+} (e_{ij})(u_{ic}) - \lambda v_{jc}\right) \end{aligned} \tag{2.1.13}$$

The parameter updates in equation (2.1.13) are executed through *batch learning*. In this process, all parameters u_{ic} and v_{jc} are randomly initialised then, while the result of the loss function has not converged[2] to a minimum value, the parameters are iteratively updated. In batch learning, $\nabla \boldsymbol{j}$ is calculated for all the parameters before performing any updates. This procedure is inefficient for large datasets, however, as the error for all training samples (i.e. $\boldsymbol{R}^+$) is computed before any single parameter is updated (Aggarwal, 2016).

To address this, each parameter update can be estimated using *Stochastic* Gradient Descent (SGD) defined in Algorithm 1, whereby a parameter update is made using the partial derivative of a single training sample only. That is, each parameter is updated using the value of the loss function produced using a single rating r_{ij}.

Algorithm 1 SGD (Ratings Matrix $\boldsymbol{R}$, Learning Rate α, Regularisation λ)

```
Randomly initialise U and V
R⁺ ← {(i, j) : r_ij is observed}
repeat
    Shuffle training samples in R⁺
    for each (i, j) ∈ R⁺ do
        e_ij ← r_ij − Σ_{c=1}^{x} u_ic · v_jc
        for each c ∈ {1 . . . x} do
            u*_ic := u_ic + α(e_ij v_jc − λ u_ic)
            v*_jc := v_jc + α(e_ij u_ic − λ v_jc)
            u_ic := u*_ic and v_jc := v*_jc
        end for
    end for
until Convergence of J        ▷ Stop when the value of J does not change between iterations
```

An alternative approach to SGD is Alternating Least Squares (ALS) matrix factorisation (Meira et al., 2018). The loss function in equation (2.1.10) is non-convex, meaning it contains local minima which may be indistinguishable from the global minimum when using gradient descent methods. However, when fixing one of either latent matrices $\boldsymbol{U}$ or $\boldsymbol{V}$ as constant, the loss function becomes quadratic with a single, solvable minimum value. This is the approach of ALS factorisation as summarised by Algorithm 2; the values in $\boldsymbol{U}$ are held constant while the values for $\boldsymbol{V}$ are solved with simple least-squares regression. Then, in the next step, $\boldsymbol{V}$ is held

[2]Convergence may be enforced by setting a maximum number of parameter update iterations. Otherwise, convergence is reached when the value of the loss function changes by less than a certain tolerance value (such as 0.001) between iterations.

Algorithm 2 ALS (Ratings Matrix $\boldsymbol{R}$, Learning Rate α, Regularisation λ)

```
Randomly initialise U and V
repeat
    Fix all values in V as constants
    for each i ∈ {1...n} do
        Solve for U_i.
    end for
    Fix all values in U as constants
    for each j ∈ {1...m} do
        Solve for V_j.
    end for
until Convergence of J        ▷ Stop when the value of J does not change between iterations
```

constant whilst new values for $\boldsymbol{U}$ are calculated; these alternations in the solvable parameters ensures a steady decrease to the global minimum of the loss function (Meira et al., 2018).

The SGD approach to minimisation is theoretically less computationally intensive, yet the ALS method may be preferred over SGD due to the ease with which the algorithm can be parallelised (Aggarwal, 2016). With ALS, solving for the user and item factors are processes independent of each other and thus each minimisation can take place in parallel (Zhou et al., 2008). This leads to much-improved efficiency which overcomes the issues of scalability present in CF methods (Meira et al., 2018).

2.1.4.2 Neural networks

Neural Network (NN) recommender systems are capable of learning complex, non-linear relationships between users and items to provide accurate item suggestions (Hrnjica et al., 2020). In this context, given various inputs, NNs can be used to predict the ratings $\hat{r}_{ij}$ a user would award an item, the highest of which would be recommended to the user.

In its simplest form, a NN consists of a layer of input nodes (or neurons) which receive numeric representations of a set of independent variables. The values supplied to the input neurons are then propagated through a network of hidden layers, each performing specified mathematical operations, before being combined into a single value (the predicted rating) and returned at an output node.

Figure 2.2 illustrates this concept with a NN of three layers: one input, one hidden and one output. Each layer l is connected to the next layer $(l+1)$ through a series of weights $w_{ij}^{(l)}$ where i is the index of the origin node and j is the index of the destination node. Layer $(l+1)$ is said to be *densely* connected to layer (l) as all the nodes in the former layer are connected to all nodes in the latter. Each layer also contains a *bias* node $b^{(l)}$ which receives no input, has a node index of 0, is always set to a value of 1, and provides the same function as the intercept in regression problems.

In addition to the bias node $b^{(l)}$, the input layer consists of a set of input variables x_1 to x_p; these may be user attributes or profiles, item characteristics or descriptions, or even images. This input layer is connected to a hidden layer consisting of *derived features* $z_1^{(l)}$ to $z_q^{(l)}$ which are not directly observed and are created through a linear combination of their inputs (Hastie

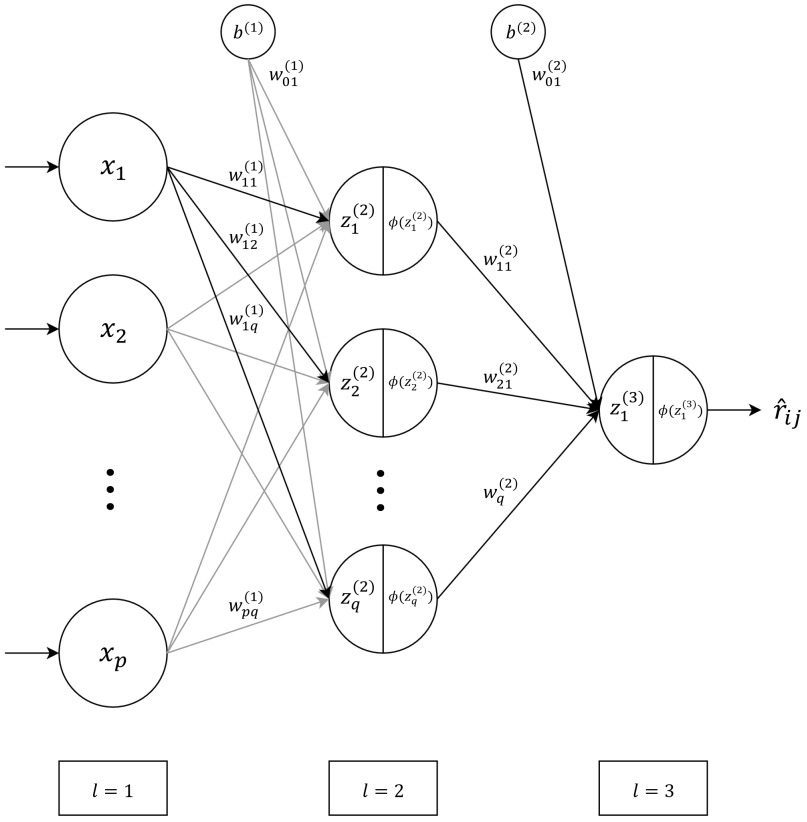

Figure 2.2: A Neural Network with layers $l \in \{1, 2, 3\}$, input nodes $x_i, i \in \{1, \ldots, p\}$, hidden nodes $z_j, j \in \{1, \ldots, q\}$, bias nodes $b^{(l)}$ and an output node $z_1^{(3)}$, all connected by weights $w_{ij}^{(l)}$.

et al., 2009).

The values of derived features in the hidden layer of Figure 2.2 are calculated as:

$$z_j^{(l+1)} = w_{0j}^{(l)} b^{(l)} + \sum_{i=1}^{p} w_{ij}^{(l)} x_i \quad \forall j \in \{1, \ldots, q\} \tag{2.1.14}$$

which combines the node's inputs and biases with their corresponding weights.

Next, the output of hidden layer nodes is then determined through an *activation function* $\phi(z)$. The choice of activation function influences the ability of the network to model different types of data; linear activation functions, $\phi(z) = z$ for example, reduce the network to a linear model. Non-linear activation functions such as the sigmoid and hyperbolic tan (tanh) functions can be used instead to model more complex data (Sharma et al., 2020).

There are no explicit guides to selecting an activation function as the success of the function depends on the system being built and experimentation is required to determine the most suitable. The Rectified Linear Unit (ReLU) function: $\phi(z) = \max(0, z)$ however, is a popular and efficient choice. As suggested by Sharma et al. (2020), ReLu is a sound initial activation function to use in hidden layers and others should be experimented with if satisfactory performance is not obtained.

Once the activation function is applied to the hidden layer nodes, the outputs of each of those nodes are then linearly combined in the output layer node ($z_1^{(3)}$ in Figure 2.2) with their corresponding weights as in equation (2.1.14). Typically the activation function applied in the output node of a regression problem is the identity function $\phi(z) = z$, which produces a predicted value $\hat{r}_{ij}$. In Figure 2.2, once a set of user or item inputs have been propagated through the hidden layer, the final predicted rating is calculated as:

$$\hat{r}_{ij} = w_{01}^{(2)} b^{(2)} + \sum_{j=1}^{q} w_{j1}^{(2)} \phi(z_j^{(2)}) \tag{2.1.15}$$

Equations (2.1.14) and (2.1.15) can be generalised to networks of more complex architectures than in Figure 2.2 with multiple hidden layers consisting of differing numbers of derived feature nodes (Hastie et al., 2009).

The accuracy of the rating predictions $\hat{r}_{ij}$ is dependent on the weight values which are combined with the input variables. The optimal weight values which produce the most accurate network are determined through gradient descent learning. In the case of regression NNs, the loss function to be optimised is:

$$J(\boldsymbol{w}) = \frac{1}{2} \sum_{(i,j) \in \boldsymbol{R}^+} (r_{ij} - \hat{r}_{ij})^2 + \frac{\lambda}{2} ||\boldsymbol{w}||^2 \tag{2.1.16}$$

which measures the average square error[3] of the predictions made for each observed rating $r_{ij} \in \boldsymbol{R}^+$, using the set of weights in the network $\boldsymbol{w}$ and is penalised by a regularisation term λ.

Then, using the weight update scheme in equation (2.1.11), the weights of the neural network are optimised using a gradient descent technique known as *Back Propagation* (BP) as described in Algorithm 3 (Khan and Sahai, 2012).

Algorithm 3 BP (Set of Neural Weights $\boldsymbol{w}$, Learning Rate α)

```
Randomly initialise weights w to values close to 0
repeat
    for each r_ij ∈ R^+ do
        Compute r̂_ij
    end for
    Compute J(w) = (1/2) Σ_{(i,j)∈R^+} (r_ij − r̂_ij)^2
    for all (i, j) and l do
        w_ij^(l)* := w_ij^(l) − α ∂J/∂w_ij^(l)
        w_ij^(l) := w_ij^(l)*
    end for
until max number of epochs        ▷ Repeat the learning for the specified number of epochs
```

[3] As the observed ratings may be restricted to integers or or pre-defined decimals, the square error of the predicted ratings may be inflated.

Algorithm 3 optimises the weight values by updating them according to the gradient of the loss function (2.1.16) at a given weight:

$$\alpha \frac{\partial J}{\partial w_{ij}} \tag{2.1.17}$$

where the learning rate α controls the magnitude of the weight updates as in Algorithms 1 and 2.

Weight decay introduces a penalty term λ to the loss function in the BP algorithm which is used to shrink weight values. For larger values of λ, the network weights tend to 0, thereby eliminating branches of the network which may overfit to the training data and are highly influenced by noisy samples present in the training data.

Alternatively, or in addition, *dropout* may be used to "drop" a random percentage of nodes from the network by setting their outputs to 0 per batch. Consequently, each weight update is applied to a slightly different version of the network leading to a model with improved generalisability. This produces a similar regularisation effect whereby the network does not become dependent on any particular branches as random portions of it are eliminated (Hastie et al., 2009).

The above summary of neural network architecture can be expanded through various modifications. For example, with slight changes to the network formulation, other inputs such as vectors may be propagated through the network. More complex data structures such as images or text, however, require a *convolutional* approach.

To process text in a NN, a set of convolutions is required; this process is detailed in Appendix A.4. In summary, *embedding* layers transform text input into numeric matrices which are then convolved. This process involves applying *filters* to the textual embeddings to extract the most meaningful latent features of the text. These latent features act as a representation of the text that can then be propagated through a NN as with any numeric vector (Liu, Li, et al., 2021).

Often, it may be beneficial to use both numeric and non-numeric variables as input to a NN. In such cases, independent branches of a NN need to be integrated.

Multiple-input neural networks can be used to process assortments of data together and have wide-reaching applications. Such networks are well suited to recommendation system problems as they can handle both rating and review data. Rating data indicates overall user preference, whereas textual reviews supply insights into specific item attribute preferences (Xi et al., 2021). Hence, when compared to collaborative filtering, using review text in addition to rating data can greatly improve recommendation quality (Catherine and Cohen, 2017). Additionally, the accuracy of latent user and item factors is known to decrease with increased rating data sparsity - textual reviews can be used to enrich these latent representations (Li et al., 2017).

Review and rating data can be combined in neural networks in two ways. Firstly, Joint Representation Learning (JRL), as described by Zhang, Ai, et al. (2017) and illustrated in Figure 2.3, can be used to integrate different data sources into unified latent representations. This technique, also applied by Xi et al. (2021), combines all reviews written by a given user and all reviews written for an item through dense layers and *fusion* operations into a review latent representation. Fusion is a method used to combine different latent factors together before

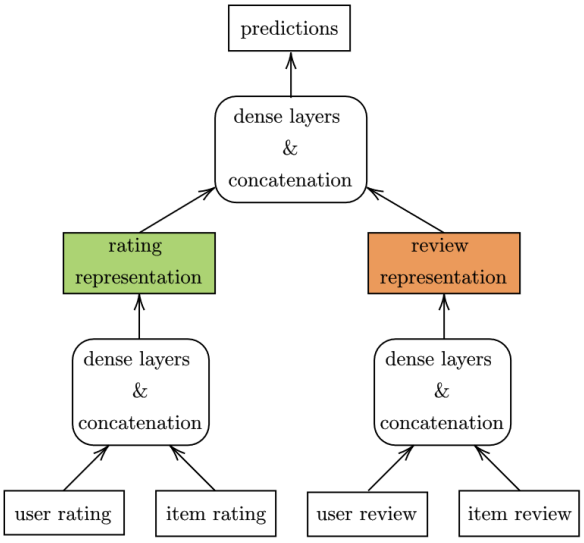

Figure 2.3: Structure of joint representation learning producing rating and review latent representations.

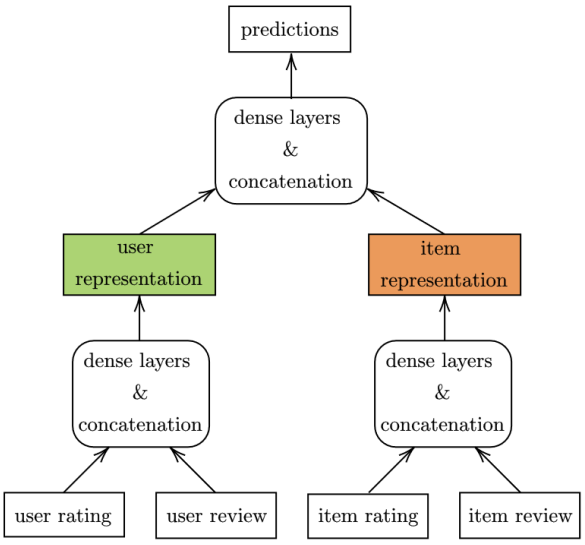

Figure 2.4: Structure of joint representation learning producing user and item latent representations.

propagating further through a network[4] (Liu, Li, et al., 2021). Then, in a parallel portion of the network, representations of the user's ratings are combined with the item's ratings in a similar fashion to produce a review latent representation. As seen in Figure 2.3 these latent rating and review representations are then combined and propagated through a series of nodes to produce a recommendation.

An alternative approach as seen in Figure 2.4, which can be thought of as a form of neural network matrix factorisation (Dziugaite and Roy, 2015), unifies user ratings and reviews into a user representation and item ratings and reviews into an item representation through dense layers and concatenation. As this approach bears similarities to matrix factorisation, it has achieved strong predictive accuracy applied to various recommendation scenarios (Seo et al., 2017; Liu, Li, et al., 2021; Liu, Wang, et al., 2020; Lu et al., 2018).

As research into the use of NNs for recommendation has grown in recent years, it has been found that these complex networks can be used to produce reliable, high-quality recommendations (in the form of predicted ratings or explicit item suggestions) which outperform more traditional methods such as matrix factorisation and item-based collaborative filtering (Zhang, Yao, et al., 2019). Primarily, in situations where complex user-item relationships can be extracted through an abundance of training data, NNs are considered an appropriate choice of RS system.

2.1.5 Strengths and weaknesses of standalone systems

As explained by Lü et al. (2012), choosing an appropriate recommender system methodology is largely dependent on experimentation. However, deciding between approaches can be influenced by the impact of each method's strengths and weaknesses on the recommendation scenario at hand. Table 2.1 summarises the main advantages and shortcomings of each method described

[4]Fusion may be in the form of element-wise addition, element-wise multiplication or concatenation of two vectors. Concatenation simply appends two vectors together (Liu, Wang, et al., 2020).

above.

Table 2.1: Strengths and weaknesses of standalone recommendation systems.

Method	Strength	Weakness
(IB)CF	No domain knowledge required. No need for user profile or item content data. Serendipitous recommendations.	Rating sparsity degrades performance. Limited by *cold-start*. Poor scalability. Slow reaction to change.
Content-Based	No rating data required, so no cold-start. All items are equally likely to be recommended.	Item content must be analysed. Items recommended are limited to the user's past consumption (not serendipitous).
TF-IDF	Provides a simple, understandable way to compare documents.	Significant pre-processing required. Word order and semantics are not preserved. Performance degrades with sparse documents.
Matrix Factorisation	Efficient and scales well. Proven to be highly accurate.	Limited by *cold-start*. Only applicable to numeric data. Can only capture linear relationships. Lacks interpretability.
Neural Networks	Can capture, complex non-linear relationships. Can accept varying inputs. Flexible design options.	Lacks interpretability. Requires rigorous architecture design and hyperparameter tuning.

Firstly, CF systems are diverse as they can be applied in any domain as only some form of rating data is required; this is beneficial in scenarios where user or item metadata are unavailable or would be too costly to extract. Furthermore, CF systems are known to produce serendipitous recommendations - items which are novel or surprising to the user (Barragáns-Martínez et al., 2010).

The primary challenge facing CF systems is the lack of user rating data which leads to both poor predictive accuracy and the *cold-start problem*. The cold-start problem applies to new items which have not been rated by any users, and new users who have not rated any items; in both cases, a CF system is unable to produce a recommendation. Then, even after users have made some ratings, the overlap between users and items is rarely dense and this diminishes predictive accuracy. Unfortunately, one solution to data sparsity - increasing the number of users in the system - diminishes the performance of CF systems as they scale poorly; computing the similarity between millions of users or thousands of items can quickly become infeasible (Lü et al., 2012). The issue of scalability then has further consequences - as user/ item similarity computations are expensive, they are often performed infrequently which results in a static representation of the user-item relationships in a system (Burke, 1999a).

Content-based systems, on the other hand, do not need to consider user ratings to produce

recommendations and therefore do not encounter issues with new users or sparsity. Likewise, item recommendations are not influenced by general popularity as the system is not swayed by underlying user preferences. This presents a weakness too, however, as CB systems will only recommend items similar to those previously enjoyed by a user - such items may appear obvious (and are not serendipitous). Additionally, CB systems are dependent on the knowledge of domain experts and rigorous modelling of item attributes - this cannot be applied in every scenario and further, may lead to over-simplified or obvious recommendations (Barragáns-Martínez et al., 2010).

To address the issues of knowledge engineering in CB systems, TF-IDF methods may be used to assist in representing the unstructured data that describes items in an understandable, numeric format (Pazzani and Billsus, 2007). Unfortunately, these methods require significant pre-processing efforts to produce a representation which does not retain any contextual information and is largely dependent on the quality of text provided (Liu, Li, et al., 2021).

Matrix factorisation is too limited by the cold-start problem as with CF - new user and item latent factors cannot be extracted as they do not exist in the original matrix $\boldsymbol{R}$. And like CF, such factorisation can only be applied to numeric data and is limited to linear user-item relationships only (see equations (2.1.2) and (2.1.7)). Despite this, matrix factorisation has been proven to be highly accurate and, through procedures such as Algorithm 2, highly efficient (Bell et al., 2009).

Matrix factorisation and neural networks are both limited by a lack of interpretability, however; latent factors may often seem arbitrary and NNs are considered "black boxes" whereby inputs are inexplicably converted to outputs without much insight. The ambiguity posed by NNs is accompanied by an arbitrary design process; unlike other techniques, the design possibilities for NNs are infinite and the choice of hyperparameters requires much tuning, often through grid searches, as the configuration and performance of a network is largely dependent on the scenario (Zhang, Yao, et al., 2019).

These limitations of NNs are not observed without accompanying benefits; primarily, the structure of neural networks allows for non-linear patterns in data to be modelled more accurately than linear approaches. The use of non-linear activation functions, flexibility in the network structure, and the ability to incorporate multiple different data sources lead to broad ranges of model expressiveness which results in more accurate representations of the underlying interaction between users and items in a given scenario (Zhang, Yao, et al., 2019).

Hence, NN systems are an appropriate choice when modelling complex relationships present in an abundance of different training input data sets (Zhang, Yao, et al., 2019). Matrix factorisation is popularly applied to scenarios involving sparse rating matrices (Aggarwal, 2016). CF systems tend to work best when item data is limited or unstructured and CB systems are best suited to domains with rich item data (Melville and Sindhwani, 2010). Notwithstanding, modern RS combine multiple methods in hybrid approaches that aim to avoid the weaknesses of standalone systems and reap the benefits of combining predictions (Lü et al., 2012).

2.1.6 Hybrid recommender systems

Burke (2007) defined and compared several approaches in which CF and CB systems could be combined into a hybrid system:

Weighted Two separate systems generate recommendations that are linearly combined (such as the system detailed by Mobasher et al. (2004)). Typically a CF system will establish a set of recommended products determined through user similarity, and a CB system will determine items that match a user's preferences. In both systems, suggested items are ranked according to the likelihood that the user would enjoy the item (via a predicted rating, perhaps). Then, the final set of items recommended is chosen as either the union or intersection of the CF and CB recommendations, where the new item rankings are calculated through a linear combination, for example, $\beta \text{rank}_{CF} + (1 - \beta)\text{rank}_{CB}$.

Mixed Predictions made by all systems are presented together (such as the system detailed by Smyth and Cotter (2000)). If both a CF and a CB system are used to predict a user's rating for unrated items, then the final set of items recommended in a mixed hybrid are the recommendations from both systems, ranked by their predicted rating calculated in each system.

Switching The type of recommender used in the hybrid is dependent on the situation at hand. This hybrid acknowledges situations in which the performance of different systems is not constant for a given domain. As illustrated by Billsus and Pazzani (2000), if their CB system could not produce any recommendations (or yielded recommendations with low confidence) first, then a CF system would be used next.

Feature augmentation This hybrid is similar to content-boosted CF where the predictions of one system can be used to reduce rating sparsity (Melville and Sindhwani, 2010) before another system is applied. As implemented by Melville, Mooney, et al. (2002), a CB model can be used to learn user preferences and impute missing ratings. Then, once the sparsity of the rating matrix has been reduced, CF can be applied with greater predictive accuracy.

Cascaded The items recommended by one system are refined by another (such as the system devised by Burke (2002)). This hierarchical approach is useful when breaking ranking ties; if a first system produces a set of items that are all predicted to be enjoyed equally, a second system can be used to establish different predicted ratings that can be used to re-rank the tied items.

Zhang, Ai, et al. (2017) noted an important distinction between models that make use of hybrid algorithms and those that use hybrid information sources. The hybrids described by Burke (2007) above are considered hybrid algorithms as they combine multiple distinct strategies. Multi-input neural networks described in Section 2.1.4.2, on the other hand, are an example of models which hybridise inputs - a single strategy that assembles different information sources to produce recommendations.

Regardless, as Lü et al. (2012) described, the success of different systems or hybrid approaches is dependent on the item domain and data available and requires experimentation.

2.2 The Evolution of Wine Recommender Systems

Typically, RS have been applied to sectors in which either the attributes of the items are easily discernible (such as the genre and cast of movies) or user consumption and preference records

are readily available (such as online purchase history) as such data can be used to simply match and recommend similar products to a user. Generating recommendations for items with complex features (for example, news articles) or when large amounts of user preference data are unavailable (for example, someone's car purchase history) presents more of a challenge (Bobadilla et al., 2013).

These data limitations are present in the wine industry; acquiring the features of a bottle of wine[5] is largely dependent on experts and, until recently, large databases of user wine preferences have been unavailable - this has restricted the development of wine RS in the past. Hence, without a robust and commonly available wine recommendation system, discovering and buying wine has presented a challenge for novice consumers (Smith, 2019).

Wine is unique in that it cannot be tasted before buying online or in stores; this forces the user to purchase wine based on other factors such as the weight and design of the bottle, its price, awards it may have won and descriptions on the back label (Sáenz-Navajas et al., 2013). Purchasing wine based on those measures, and not on the attributes (flavour, aroma, body etc.) of the wine can result in the user making the wrong purchasing decisions (Smith, 2019).

However, modern, hybrid recommendation systems, such as Vivino (2022), HelloVino (2022), Sippd (2022), and WineRing (2021), have been able to address this issue[6]. These applications provide widespread support to wine consumers by aiding them when purchasing wine by providing aggregated rating data, reviews and tasting insights. These applications can recommend wines from around the world, provide guides to help users when exploring wine and ultimately discover wine that should be of most interest to them.

Advanced wine recommender systems, however, were not created overnight and have much more humble beginnings. In the late 1990s and early 2000s, before access to the internet and mobile devices were common, consumers relied on the opinions of wine experts when deciding which wines to buy. Using the ratings of well-respected critics, such as Robert Parker (Parker and Rovani, 2002) and his widely used point system, allowed consumers to navigate the wine market without any prior knowledge. Purchasing wines solely based on their accolades and/or high ratings soon became ineffective as it was found that the preferences of wine experts do not always correlate with those of the casual drinker (Smith, 2019).

Basing purchasing decisions on the opinions of experts is unreliable as not only do they possess greater experience that influences their preferences (Cruz et al., 2018), but experts are fallible; it was found that critics can be biased by the appearance of wine and not its flavour (Morrot et al., 2001). This impacted consumers who would typically purchase a wine based on the interpretation that its high expert rating indicated the likelihood that they would enjoy the bottle. This may have led to disappointment as the opinions of experts should be used as an indication of *quality* and not as a bespoke recommendation.

Hence, using the judgements of professionals alone when purchasing a product as nuanced as

[5]Wine is a complex and highly variable drink consisting of multiple components (Goode, 2020), each of which is accompanied by plentiful amounts of wine-specific literature and scientific research. As a result, wine terminology has become less understandable to those who are inexperienced and can exclude those who do not drink from appreciating the intricacy of a glass of wine (Cruz et al., 2018). Hence, a brief overview of wine can be found in Appendix A.5 to contextualise the terms frequently used throughout this research.

[6]Barring Vivino which was founded in Denmark, these applications are based in America and thus are of limited use to South African consumers.

wine is insufficient; for example, published ratings may not be able to assist a novice consumer in deciding between a 2004 Cabernet Sauvignon and a 2019 Chardonnay, two massively different wines, both scoring 95 Parker Points (Parker and Rovani, 2002).

Herein lay the opportunity to develop a system built upon the consumer's preferences which would facilitate more meaningful recommendations - a process which began 3 decades ago.

2.2.1 Knowledge engineering and case-based reasoning

In the late 1990s Burke (1999b) proposed a shopping application, "The Wasabi Personal Shopper", which used knowledge engineering and case-based reasoning to find wines similar to a target wine. Knowledge engineering is the process of developing a system used to make decisions by capturing and utilizing the knowledge and thought processes of human experts (Studer et al., 1998). Case-based reasoning is a form of problem-solving which uses past scenarios to address a current, similar scenario (Kolodner, 1992). The combination of these methodologies can be interpreted as a rudimentary form of content-based recommendations.

CF was then used to bolster the wine suggestions made by aggregating user preferences to discover similar users and refine candidate items for recommendation. By matching users according to their implicit preferences, the system was able to identify clusters of similar users who shared similar tastes in wines. Further specifics of the system developed by Burke (1999b) can be found in Appendix A.6.1.

2.2.2 Other approaches to wine recommendations

Slightly later, Almasi and Lee (1999) proposed "Wine Guru", their approach to producing a "personalised wine list" which would be used to recommend wines that would best match a user's taste profile and suggest food-wine pairings. The user taste profiles were determined through clustering methods applied to wine consumption records and wines were clustered according to their attributes. Recommendations were then formed by matching user clusters to wine clusters - details covering this process may be found in Appendix A.6.2.

More recently, Michaelis et al. (2008) and Patton and McGuinness (2009) developed systems that relied on explicit user recommendations to provide suggestions for food and wine pairings. These stood out from previous applications as they were restricted to providing recommendations for specific occasions and not for a user's general preferences. In doing so, they avoided the challenges presented by data sparsity when using content and collaborative filtering as wine or food suggestions were found through simple queries to a network of existing recommendations.

These RS were limited by the networks of explicit recommendations that they relied upon - if no user had ever made a recommendation for a certain wine or food combination, it would never be presented to the user as no content or collaborative filtering was used to uncover such hidden patterns. A summary of the process involved in producing these recommendations is found in Appendix A.6.3.

2.3 Modern Wine Recommender Systems

Purchasing wine online is advantageous as it lowers *search costs*, the time, effort and money invested in searching for and deciding on buying a product. When the costs associated with finding and comparing reliable price and quality information across multiple retailers are lower, consumers have a better shopping experience. It was found that if retailers invest in providing detailed, transparent information about their inventories, consumer satisfaction increases (Lynch and Ariely, 2000).

Thus, as the online wine market has grown, so has the availability of wine applications (apps) to help consumers navigate the products available. Even though wines cannot be tasted when shopping online, the properties of wines are rarely revealed to a casual drinker in a single sip. Therefore, wine apps are invaluable as they remove consumer reliance on expert reviews and instead grow consumers' wine experience based on their personal preferences.

Modern wine apps make use of "wisdom of the crowd" algorithms to help consumers navigate the wines available to them, recommend new products to try, and help them explore various wines with greater discrimination (Smith, 2019). Current wine recommenders make use of proprietary algorithms to generate their wine suggestions and thus have closely guarded their approaches. Some of these applications include:

HelloVino (2022) makes use of content-based methods to make wine recommendations. A user specifies their wine preferences explicitly by selecting their preferred wine type (red or white), style (light, medium or bold), desired price range, and flavours (between options of fruits, flowers and spices). HelloVino then makes recommendations of wines that best suit these defined preferences.

Sippd (2022) offers functionality similar to HelloVino; a user begins by specifying their wine preferences explicitly. Firstly, a user selects their preferred varietals/ wine styles from a list. Then, the user is requested to rate at least 3 wines they have drank on a scale from 1 to 5. Then, using their "Taste Match Technology", wines are matched to the user, accounting for their specific preferences. A percentage match is generated for each wine in their database as an indication of how much the user would enjoy that wine.

Vivino (2022) can provide recommendations with a 93% confidence level using their "Match for you" recommender system whereby a "match score" is provided as an indicator of how likely the user will enjoy a bottle of wine. While the specific machine learning techniques are not publicly available, the recommendations are known to be made based on purchase history, flavour and variety preferences, as well as user ratings. As with the Sippd, Vivino emphasises that their recommendations become more accurate as the user rates more wine.

WineRing (2021) takes a unique, preference-based approach to recommending wines; using their patented "Preference Profile" methodology (Dillon et al., 2019), *rating dependency patterns* are established and used to build *user preference models*. These preference models represent the association between the ratings the user has awarded and the characteristics of the wines which drove those ratings. Further details of how these models are built and used to generate recommendations are found in Appendix A.6.4.

This synopsis covered various recommender methods and their applications in the wine industry. From this, it appears that a reliable way to produce robust recommendations is to use a combination of methods that capitalises on the benefits of each individual approach. The following chapter discusses the data used to build the wine recommendation system proposed in this thesis.

Chapter 3

Data

This chapter details the data collected and cleaned from online wine resources that are used to build various wine recommender systems. Further, the approach to data segmentation, the management of the cold-start problem, missing values and outliers as well as the effects of scaling are all explored.

3.1 Dataset Description

The datasets used to build the wine recommendation system consists of, firstly, a database of wines and secondly, a set of ratings and reviews for those wines. The first database contains attributes and descriptions for a set of 1,640 South African wines supplied by wine.co.za (WineNet, 2022). Each of the variables recorded for the wines are summarised in Tables 3.1 and 3.2.

The second dataset consists of 210,605 ratings made by 92,514 users for the 1,640 wines; all ratings (made in the interval [1, 5] in increments of 0.5) of these wines were scraped from Vivino (2022) during August 2021.

Figure 3.1 plots the decreasing frequency of ratings made for each wine where wine 1 received the maximum number of ratings, and wine 1,640 received the least. This illustrates that although some wines are rated thousands of times, most wines receive very few numbers of ratings with a mean rating frequency of 134.9.

This highlights the effect the *cold-start* problem has on recommender systems which make use of rating data to generate suggestions. The cold-start problem concerns users and items for which there is insufficient data (i.e. no ratings) required by rating-based systems to make recommendations. Hence, when dividing users between training, validation and test sets, rating data may be hidden in such a way that users or items with no rating history (i.e. new users or items) may be present in the sets. This poses complications for generating recommendations and thus two protocols are implemented to avoid the cold-start problem.

Table 3.1: Summary of the free text and factor variables recorded for each wine in the dataset.

Variable	Type	Description
Name	Free text	The name of the wine.
Description	Free text	A description of the wine's flavours and aromas as determined by wine.co.za.
Winery	Factor	Which of the 115 wine farms in the dataset that bottled the wine.
Location	Factor	Which of the 37 regions produced the wine.
Year	Factor	The vintage of the wine, ranging from 1973 to 2020. Blends of numerous vintages are Non-Vintage (NV).
Grape	Factor	The predominant grape variety used in the wine; if more than 85% of the wine consists of a single grape type, that wine can be considered single-varietal. Wines composed of grapes in proportions less than 85% are "Blends". A total of 49 grape varietals are present, ranging from "Chardonnay" to "Merlot".
Type	Factor	The wine type may be any one of: "Red", "White", "Rosé", "Sparkling", "Cap Classique", "Dessert", and "Fortified".
Style	Factor	A description of how sweet a wine is perceived to be, taking on values, "Dry" "Off-Dry" and "Sweet".
Body	Factor	A term used to categorise the intensity and mouth-feel of a wine, from "Light" to "Medium" to "Full".

Table 3.2: Summary of the numeric variables recorded for each wine in the dataset.

Variable	Min	Med	Mean	Max	Description
Price	33.17	148.00	201.66	2700.00	The price of the wine in Rands (R).
Alcohol (Alc)	0.00	13.64	13.50	43.00	The percentage of alcohol by volume (ABV) of wine.
Residual Sugar (RS)	0.00	3.00	12.04	275.00	The grams per litre of residual sugar which remain in the wine after fermentation - a measure of sweetness.
pH	0.00	3.47	3.44	6.70	The acidity of the wine measured on the pH scale.
Tannins (TA)	0.00	5.84	5.90	10.20	The gram per litre amount of Tannins (a polyphenol) present in the wine.

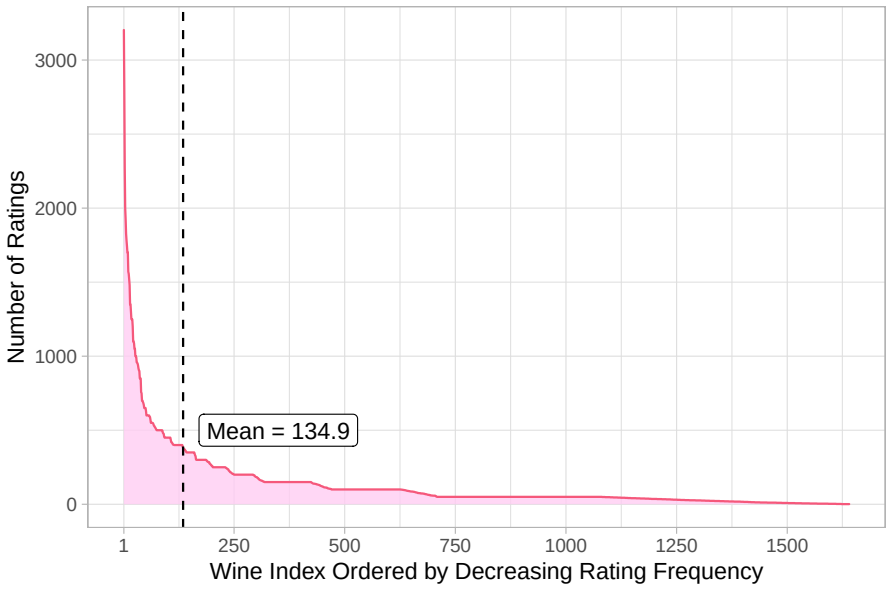

Figure 3.1: Frequency of ratings made per wine.

3.2 The Training/ Validation/ Test Split and the Cold-Start Problem

To evaluate the performance of the various RS models built, the original rating data is split into training, validation and test sets[1] in the proportions 70%, 20% and 10% respectively. As this thesis aims to explore multiple system types, all requiring rigorous hyperparameter tuning, the use of cross-validation is infeasible due to the additional computational complexity of training using numerous hold-out sets. Thus, the validation set is used for hyperparameter tuning, evaluating each system and selecting an optimal model. Then, the performance of the optimal model is assessed using the test set.

Developing each system involves training models to accurately predict the set of observed ratings $\boldsymbol{R}^+$. Then, these models may be used to make predictions of the unobserved ratings $\boldsymbol{R}^-$. Achieving this involves allocating each $r_{ij} \in \boldsymbol{R}^+$ to a single training, validation or test set. But, as each r_{ij} corresponds to a unique user-wine pair, when creating the training and evaluation sets by selecting ratings at random, problems may arise for users who have made few ratings.

In the case of users who have made only one single rating (referred to as single-users henceforth), their rating and consequently their presence may only exist in one of the training or evaluation sets. This simulates the cold-start problem; if a single-user is not present in the training set, no models can learn how to generate recommendations for them. Likewise, if the user is only present in the training set, there is no means to measure the quality of their recommendations as there is no data for that user in any evaluation set.

In total, 68,160 users made only one rating on 1,566 unique wines and so roughly 74% of the dataset consists of single-users. To address this simulated cold-start problem, two protocols are

[1]The validation and test sets are collectively referred to as evaluation sets.

implemented and compared. Solving the cold-start problem requires that a user exists in the system before generating recommendations; in other words, each user must exist in the training set for any models to learn their rating behaviour. Thus, maintaining a random partitioning scheme mandates that each user has made at least two ratings and at least one of which exists in the training set.

Both protocols implemented generate a secondary rating for all single-users, given their first rating, to serve as a rating history. Hence, 68,160 new ratings are generated using the following protocols, individually, for each single-user to ensure that sufficient data for every user exists in the system. No more than one rating is made per user to avoid major changes to the original data and making too many assumptions about a user's rating behaviour. Although these protocols may be viewed as data manipulation, they should rather be considered as the approaches that the following systems would make use of to generate predictions for new, unseen users.

The *target wine* is the wine most similar to the single wine rated by a user (the *source wine*) and is assumed to be a wine the user would likely rate in the future. The most similar wine is determined using the simple and widely used Gower distance (Gower, 1971) calculated on the non-free text variables of each wine (listed in Tables 3.1 and 3.2); more details regarding this distance measure can be found in Section A.2. The predicted rating for this target wine is generated in two ways:

3.2.1 Protocol 1 - empirical CDF sampling

Choosing the mean rating of the target wine as a predicted rating does not capture the variability of the ratings made by a user. Instead, the empirical Cumulative Distribution Function (CDF) of the ratings for the most similar wine is used with inverse sampling to generate a predicted rating.

A random number y is drawn from the Uniform distribution between 0 and 1, $y \sim U(0, 1)$. This number can then be substituted into the inverse CDF of a wine's ratings to generate x, a simulated rating which shares the same distribution as the ratings for that given wine. To illustrate, if wine a was determined to be most similar to the user's source wine, and $y = 0.65$, then using the empirical CDF of the ratings made for a, as seen in Figure 3.2, the generated rating for the target wine is $x = 4$.

Through this inverse sampling, values close to the mean rating are likely to occur often as the mean rating often dominates the CDF. Yet, the random component incorporates the natural variation that exists in users' rating patterns, thereby serving as a more realistic simulated rating than simply using the mean.

3.2.2 Protocol 2 - k nearest neighbour imputation

k Nearest Neighbour (kNN) methods can be used for data imputation and have been shown to perform well on various datasets when compared to other complex methods (Jadhav et al., 2019). With kNN imputation, calculating a secondary rating for a user involves finding their k most similar users and computing their average rating for the target wine (Patra and Ganguly, 2019).

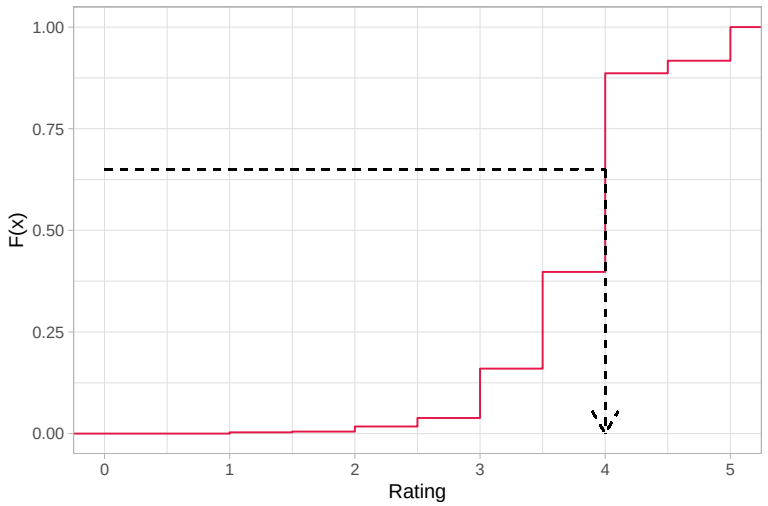

Figure 3.2: Empirical CDF of ratings for a wine.

Firstly, determining the k most similar users is computationally expensive as each user is to be compared to every other user. This computational effort is unnecessary, however, as when determining a neighbourhood of similar users for those who have made one rating, only users who have rated the source wine may be compared. Illustratively, users who have rated just red wines cannot be compared to users who have rated only white wines as their rating vectors are perfectly dissimilar; this reduces the number of comparisons to be made as only users who co-rated at least one wine are compared.

Therefore, the k most similar users for a given single-user are those who awarded the source wine the same rating; comparing the ratings of other wines is futile if the user has only made one rating. In the case that no other users awarded the source wine the same rating as the single-user, the k most similar users are those who awarded the source wine the next closest rating. Note, the value of k is dependent on each user - some single users may have many similar users while others may only be similar to $k = 1$ other user.

Then, the generated rating is calculated as the average rating made by the k most similar users for the target wine. In the case that none of the k users rated the target wine, the average rating of the most similar user k_1 is used instead (Bobadilla et al., 2013).

Applying these protocols involves generating secondary ratings for 1,010 unique target wines, the barplot of which can be seen in Figure 3.3. In Figure 3.3, the frequency of the ratings made by the single-users are the *original* ratings, plot in blue. From this, each protocol roughly captures the original rating frequency distribution. The main discrepancies are that ratings of 3.5 are much more frequently generated than they appear in the original data and ratings of 5 are generated less than they appear in the original.

Once the new ratings are added to the original rating data, the ratings are unified into two matrices $\boldsymbol{R}^{\text{CDF}}_{n\times m}$ and $\boldsymbol{R}^{k\text{NN}}_{n\times m}$ corresponding to the datasets created after applying the CDF and kNN protocols, respectively. As seen in Definition 1, each matrix consists of $n = 92,514$ user rows and $m = 1,640$ wine columns. Representing the rating data in such a manner results in sparse, very low-density matrices where approximately 0.2% of all elements in either $\boldsymbol{R}$ matrix contain a value.

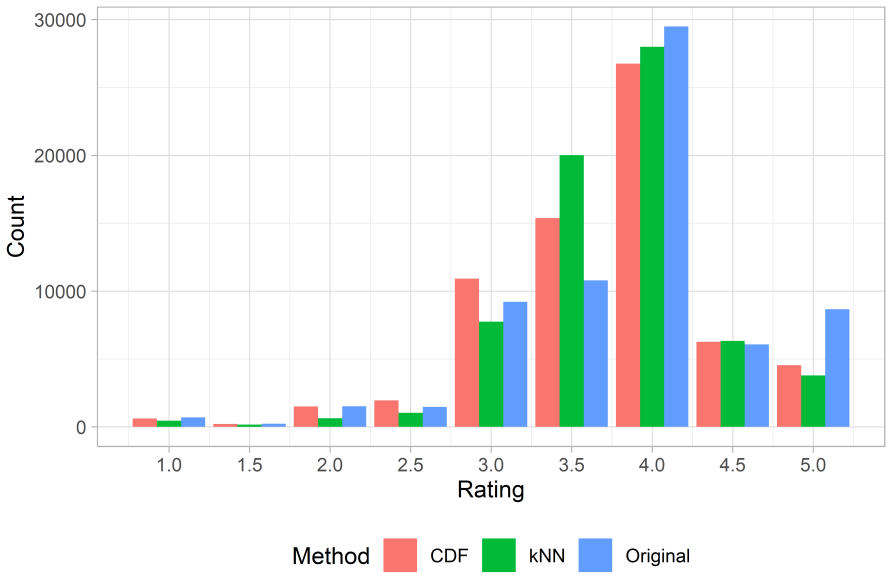

Figure 3.3: Frequency of ratings made by single-users and ratings generated by CDF and kNN protocols.

In each matrix, 30% of ratings are hidden to form the training sets; the ratings are hidden such that at least one random rating per user remains in the training set. Of the 30% of hidden ratings, two-thirds are used for validation (hyperparameter tuning and model selection) and a third is used as a test set to evaluate the performance of the final model. In all, 195,135 ratings are used for training, 55,753 are used for validation and 27,877 for testing. Where available, the reviews associated with each rating are similarly divided among the sets.

Henceforth, $\boldsymbol{R}^{\text{CDF}}$ and $\boldsymbol{R}^{k\text{NN}}$ may be referred to as either the CDF and kNN (imputed) datasets, respectively. Before building RS systems, both the rating data and wine database require data cleaning.

3.3 Missing Values

The majority of the wine attribute data (summarised in Tables 3.1 and 3.2) is collected from wine.co.za and naturally some variables contained missing values which are imputed. Firstly, missing `Year` values (0.28% missing) are taken from the wine's name (which often includes the vintage) or are otherwise non-vintage (NV).

The `Grape` variable is constructed from the wine's composition - a list of the percentages of grape varietals present in the wine recorded by wine.co.za. If a wine consists of more than 85% of a single grape variety, that becomes the predominant grape, otherwise, it is considered a "blend". For example, the `Grape` of a wine with composition 95% Cabernet Franc, 5% Merlot is Cabernet Franc, and a wine with a composition of 45% Sauvignon Blanc, 33% Chardonnay, 22% Viognier is a blend. When a wine's composition is unavailable (1.88% missing), the predominant grape type recorded for that wine by Vivino (2022) is used instead.

Similarly, `Price` values unavailable on wine.co.za (43.29% missing) are taken from Vivino (2022)

too.

Missing values for the "chemistry" variables (Alcohol, Residual Sugar, pH and Tannin each missing ±1%) are imputed manually by finding the corresponding missing value as reported by other online stores. And finally, the `Body` variable (8.36% missing) is imputed by taking the mode value corresponding to that wine's grape value. For example, the mode body for the grape Cabernet Sauvignon is "Full", and so it is assumed that wines with missing body values of that same grape variety would share a similar body.

No values were missing for the `Description` variable but, for wines which had descriptions of less than 10 words, the free text is supplemented using a description of the predominant grape from Vivino (2022). This is done to enhance the description of wines which would otherwise be incomparable to wines with rich, lengthy descriptions.

After imputing all missing values, only outliers in the numeric variables remain to be corrected.

3.4 Outliers

Obvious errors in the numeric variables are simple to locate. Alcohol in wine ranges between 0% and 22%, most wines that are considered dry have between 0 and 10 g/l of residual sugar but sweet wines can exceed 200 g/l, acidity is measured on the pH scale which ranges from 1 to 14 and Tannins in wine rarely exceed 10 g/l. Values for the chemical variables that largely exceeded the normal limits are considered data entry errors and manually corrected (e.g. an alcohol percentage of 142% is clearly meant to be 14.2%). Apart from those errors, outliers are defined using the interquartile range (IQR) method and are those values which exist outside the range $[Q_1 - 1.5(IQR), Q_3 + 1.5(IQR)]$ for each variable (Walfish, 2006). Table 3.3 summarises the number of outliers present in each numeric variable.

Table 3.3: Number of outliers present in the numeric wine variables.

	Price	**Alcohol (Alc)**	**Residual Sugar (RS)**	**pH**	**Tannins (TA)**
Number of Outliers	115 (7%)	128 (7.6%)	228 (14%)	24 (1.5%)	87 (5.3%)

Outlier values are often minimised or eliminated as they can negatively affect the performance of models, especially those which are sensitive to the mean and variance of the data. However, as the models built in this paper that make use of the numeric wine variables make no assumptions about the distributions of the variables (and thereby are not heavily influenced by outliers), the outliers were not modified.

Moreover, these extreme values represent important aspects of the data which are not erroneous and are significant to a wine's characteristics. When comparing two wines, a dry wine with 5 g/l of residual sugar should be considered the opposite of dessert wine with 250 g/l of residual sugar. Likewise, wines that cost R50 or R500 should by no means be considered similar to a wine priced at over R1500. Therefore, the presence of outliers is important in the context of recommending items as they are necessary for differentiating between the majority of wines and those with more unique characteristics. For completeness, a brief visual exploration of the numeric variables in the wine dataset can be found in Appendix A.7.

3.5 The Effects of Normalisation and Standardisation

Variables X measured with different units in varying ranges often require *scaling* before model building. The two forms of scaling, namely *Normalisation* defined as:

$$\eta(X) = \frac{X - X_{\min}}{X_{\max} - X_{\min}} \tag{3.5.1}$$

which shifts values such that all scaled values are in [0,1], and *Standardisation* defined as:

$$\zeta(X) = \frac{X - \bar{X}}{s_X} \tag{3.5.2}$$

(where s_X is the standard deviation of X) which scales values such that the transformed mean and variance are 0 and 1, respectively.

Normalisation is typically used when the distribution of X is not Gaussian or the proposed model does not make assumptions about the underlying distribution of the variable. Standardisation is typically used on normally distributed data and merely shifts the distribution - it does not enforce any bounds like normalisation.

For rating data specifically, classic standardisation cannot be applied; instead, user mean-centring is implemented. Standardisation is applied to the columns of a dataset - in the case of the ratings matrix $\boldsymbol{R}$, this would have the effect of scaling each item independently. This is counter-intuitive as it assumes that each item is measured on a different scale. A more plausible assumption, rather, is that each *user* (row) awards ratings according to their own biases and scale.

To account for this, a modified form of standardisation is applied. For each user, their ratings are centred by subtracting the user (row) mean $\bar{r}_i$ from each rating ($\zeta(r_{ij}) = r_{ij} - \bar{r}_i$). This has the effect of removing the biases from each user's rating scale; all items that are enjoyed will have positive ratings, and disliked items will yield negative ratings.

Complete row-wise standardisation is not possible (i.e. dividing by the users' rating standard deviation) as many users have only made one rating, or have made the same rating numerous times. In these common cases, the user standard deviation is either undefined or 0, thereby rendering classic standardisation unfeasible. Henceforth, the term *standardisation* should be understood as user mean-centring when discussed in the context of rating data.

After applying each scaling technique[2], the normalised and standardised ratings matrices are denoted $\eta(\boldsymbol{R})$ and $\zeta(\boldsymbol{R})$ and the similarity matrices determined using these scaled ratings are denoted $\boldsymbol{S}^v_\eta$ and $\boldsymbol{S}^v_\zeta$, respectively. In the CF approach, the similarity matrix used should match the scale of the rating data that generated it; i.e. $\eta(\boldsymbol{R})$ and $\boldsymbol{S}^v_\eta$ should only be used together. Then, predicted ratings made with scaled data are denoted $\eta(\hat{r}_{ij})$ and $\zeta(\hat{r}_{ij})$ - this does not, however, imply that the predicted rating itself was scaled.

Importantly, after a rating prediction is made using scaled data, the value needs to be re-transformed to the original rating scale for correct interpretation of results. In the case of

[2]Scaling is applied **after** the training/ evaluation data segmentation to ensure underlying patterns such as the users' mean validation ratings do not leak into the training set and influence proceeding results.

normalised data, the final predicted rating is: $\hat{r}_{ij} = \eta(\hat{r}_{ij}) \cdot (\max r_i - \min r_i) + \min r_i$ and for standardised data: $\hat{r}_{ij} = \zeta(\hat{r}_{ij}) + \bar{r}_i$.

However, it can be shown that scaling $\boldsymbol{R}$ has no effect when using content-based similarity matrices $\boldsymbol{S}^{v:s}$ or $\boldsymbol{S}^{v:\not s}$:

$$
\begin{aligned}
\hat{r}_{ij} &= \dot{\boldsymbol{s}}_j^{v:s} \cdot \boldsymbol{r}_i && \text{the rating prediction for unscaled data, where } \sum \dot{\boldsymbol{s}}_j^{v:s} = 1 \\
\zeta(\boldsymbol{r}_i) &= \boldsymbol{r}_i - \bar{r}_i && \text{the standardised rating vector} \\
\zeta(\hat{r}_{ij}) + \bar{r}_i &= \dot{\boldsymbol{s}}_j^{v:s} \cdot \zeta(\boldsymbol{r}_i) + \bar{r}_i && \text{the standardised rating prediction after transforming} \\
&= \dot{\boldsymbol{s}}_j^{v:s} \cdot (\boldsymbol{r}_i - \bar{r}_i) + \bar{r}_i \\
&= \dot{\boldsymbol{s}}_j^{v:s} \cdot \boldsymbol{r}_i - \dot{\boldsymbol{s}}_j^{v:s} \cdot \bar{r}_i + \bar{r}_i \\
&= \dot{\boldsymbol{s}}_j^{v:s} \cdot \boldsymbol{r}_i - \bar{r}_i + \bar{r}_i && \text{as } \dot{\boldsymbol{s}}_j^{v:s} \cdot \bar{r}_i = \bar{r}_i \\
&= \dot{\boldsymbol{s}}_j^{v:s} \cdot \boldsymbol{r}_i = \hat{r}_{ij}
\end{aligned}
$$

This proof demonstrates that after transforming a standardised content-based prediction to the original scale, the predicted rating is equal to that produced with unscaled data. However, the effects of scaling are more crucial for other RS methods. Models which make use of deep learning (via gradient descent, minimising some loss function, or computing distances) are greatly impacted by the scale of the variables; variables measured on larger scales will undoubtedly be more influential to model learning than variables measured on smaller scales, despite equal importance. Contrarily, models which consider variables independently when training are impervious to scaling.

To illustrate, the original Example ratings matrix 2.1.1 can be considered where its normalised and standardised equivalents are seen in Example 3.5.1 where $\eta(\boldsymbol{R})$ is formed by applying equation (3.5.1) to the columns of $\boldsymbol{R}$ and $\zeta(\boldsymbol{R})$ is formed by subtracting the row means of $\boldsymbol{R}$.

Example 3.5.1.

$$
\eta(\boldsymbol{R}) = \begin{bmatrix} 1 & 0.75 & 0 & 0.75 \\ 0 & 0.25 & 0.5 & 0 \\ 0.75 & 0.75 & 0.25 & \text{NA} \end{bmatrix} \qquad \zeta(\boldsymbol{R}) = \begin{bmatrix} 1.5 & 0.5 & -2.5 & 0.5 \\ -0.75 & 0.25 & 1.25 & -0.75 \\ 0.67 & 0.67 & -1.33 & \text{NA} \end{bmatrix}
$$

When applying the COS similarity equation in (A.1.4) to the pairs of columns of each $\boldsymbol{R}$ matrix in Examples 2.1.1 and 3.5.1, the item-based similarity matrices in Example 3.5.2 are produced. From Example 3.5.2, it is clear how scaling impacts the measurement of similarity. Note how the similarity between wines $j = 3$ and $j = 2$ vary from 0.80 in $\boldsymbol{S}^v$ to 0.51 in $\boldsymbol{S}^v_\eta$ to -0.68 in $\boldsymbol{S}^v_\zeta$. Hence, it is likely that scaling will impact both rating predictions made and the ranking of wines when producing recommendations.

Example 3.5.2.

$$\boldsymbol{S}^v = \begin{bmatrix} 1.00 & 0.98 & 0.66 & 1.00 \\ 0.98 & 1.00 & 0.80 & 0.98 \\ 0.66 & 0.80 & 1.00 & 0.54 \\ 1.00 & 0.98 & 0.54 & 1.00 \end{bmatrix}$$

$$\boldsymbol{S}^v_\eta = \begin{bmatrix} 1.00 & 0.96 & 0.27 & 1.00 \\ 0.96 & 1.00 & 0.51 & 0.95 \\ 0.27 & 0.51 & 1.00 & 0.00 \\ 1.00 & 0.95 & 0.00 & 1.00 \end{bmatrix} \quad \boldsymbol{S}^v_\zeta = \begin{bmatrix} 1.00 & 0.64 & -1.00 & 0.87 \\ 0.64 & 1.00 & -0.68 & 0.12 \\ -1.00 & -0.68 & 1.00 & -0.87 \\ 0.87 & 0.12 & -0.87 & 1.00 \end{bmatrix}$$

Note, the EUC and MSD measures in equations (A.1.1) and (A.1.5) are not affected by scaling as the subtraction of the constant user mean cancels out in the summation term:

$$\begin{aligned} &\sum_{u \in U_{ab}} \left((r_{ua} - \bar{r_u}) - (r_{ub} - \bar{r_u}) \right)^2 \\ &= \sum_{u \in U_{ab}} \left(r_{ua} - \bar{r_u} - r_{ub} + \bar{r_u} \right)^2 \\ &= \sum_{u \in U_{ab}} (r_{ua} - r_{ub})^2 \end{aligned}$$

and the JAC measure in equation (A.1.6) does not consider the magnitude of ratings at all and is too robust to scaling.

Model-based methods, such as matrix factorisation, involve minimising a loss function. Processes such as these are far more sensitive to the scale of data used. Consider again the unscaled matrix example 2.1.1; User_1 is expected to be represented by a user latent factor of greater magnitude than User_2 as the learning process preserves the scale of the ratings made by the users. This is problematic when making predictions as user vectors of greater magnitude are somewhat restricted to predicting greater ratings when combined with an item factor; this reduces the model's ability to produce varied predictions. As a result, scaling is seen as an important pre-processing step as large differences in the magnitudes of ratings in the matrix $\boldsymbol{R}$ are reduced for smoother gradient descent learning.

In summary, scaling likely affects the predictions made by IBCF and model-based methods. To explore this, all model building was applied to the unscaled, normalised and standardised data independently to assess the effects and importance of scaling.

The following chapter outlines the methods used to produce recommendations using the wine data described above.

Chapter 4

Methods

This chapter details how various methods are applied to the South African wine database, user ratings and reviews described in Chapter 3 (collectively referred to as the "wine dataset") to build recommendation systems. Firstly, the procedures used to generate recommendations are outlined; then, the methods used to evaluate the accuracy and quality of the recommendations produced by each model are described. Next, elaborating on the techniques discussed throughout Section 2.1, the approaches used to build item-based collaborative filtering, content-based, model-based and hybrid systems are outlined.

Unless stated otherwise, each methodology is used to build 6 different systems: 3 of which are systems that use the unscaled, normalised and standardised versions of the CDF-imputed ratings matrix $\boldsymbol{R}^{CDF}$, $\eta(\boldsymbol{R}^{CDF})$ and $\zeta(\boldsymbol{R}^{CDF})$. The other 3 are built using the various scaled versions of the kNN ratings matrix, $\boldsymbol{R}^{kNN}$, $\eta(\boldsymbol{R}^{kNN})$ and $\zeta(\boldsymbol{R}^{kNN})$. For brevity, the variables $\boldsymbol{R}, \boldsymbol{S}, \boldsymbol{s}, \boldsymbol{r}, \boldsymbol{U}, \boldsymbol{V}$ without any super or subscripts represent a generalised version of that variable and can be substituted for their specific counterparts denoted with the sub/superscripts $CDF, kNN, i, j, u, v, s, \not{s}, \eta, \zeta$ (refer to Table A.1 in the Appendix for details).

4.1 Generating Recommendations

As discussed in Section 2.1, memory-based CF and CB methods rely on a form of either user or wine similarity to produce recommendations, whereas model-based methods make use of statistical models instead. Thus, the techniques used to produce recommendations using wine similarity and wine models differ.

In each case, two forms of recommendations are made, the first being a predicted rating made by a user for a wine and secondly, a list of the top-Q wines which a user is most likely to enjoy. The approach to generating predicted ratings and top-Q lists for systems that make use of either similarity or models is discussed below.

4.1.1 Predicted ratings using wine similarity

Algorithm 4 illustrates the method used to predict ratings given user ratings and wine similarities and is taken from the IBCF approach described in Section 2.1.2.2. This Algorithm

details how predictions for a wine j can be generated using the ratings made by a user i and the similarity matrices $\boldsymbol{S}^{v}$, $\boldsymbol{S}^{v:s}$ and $\boldsymbol{S}^{v:\not s}$ created with IBCF and CB methods, respectively.

In Algorithm 4, the predicted rating $\hat{r}_{ij}$ is produced as a combination of user i's ratings and the similarities between wine j and the other wines rated by i. In this algorithm, the set of wines rated by user i (V_i) is fixed; hence, the predicted rating for j is dependent on $\boldsymbol{R}$ and $\boldsymbol{S}$, the rating and similarity matrices used.

The ratings matrix $\boldsymbol{R}$ used in Algorithm 4 differs by the protocol used to address the cold-start problem (CDF or kNN detailed Section 3.2) and the scaling method applied ($\eta(\boldsymbol{R})$ or $\zeta(\boldsymbol{R})$). The wine similarities determined with rating data ($\boldsymbol{S}^{v}$, $\boldsymbol{S}^{v}_{\eta}$, $\boldsymbol{S}^{v}_{\zeta}$), structured wine attributes ($\boldsymbol{S}^{v:s}$) and unstructured wine descriptions ($\boldsymbol{S}^{v:\not s}$) are based on different elements of the wine dataset, and are bound to be different. Therefore, the values of $\hat{r}_{ij}$ are likely to vary based on the choice of ratings and similarity matrix used in Algorithm 4.

Importantly, when using Algorithm 4 with IBCF, the scaling applied to $\boldsymbol{R}$ decides the similarity matrix; i.e. $\eta(\boldsymbol{R})$ must be used with $\boldsymbol{S}^{v}_{\eta}$ and so on.

Algorithm 4 CF Rating Prediction (User i, wine j, Ratings matrix $\boldsymbol{R}$, Similarity Matrix $\boldsymbol{S}$)

$V_i \leftarrow$ the set of wines rated by i
Select $\boldsymbol{s}_j$ from $\boldsymbol{S}_{V_i j}$ ▷ the similarities between j and the other wines rated by i
Retain only the top k most similar wines in $\boldsymbol{s}_j$
Select $\boldsymbol{r}_i$ from $\boldsymbol{R}_{i\cdot}$ ▷ only the ratings i made for the wines in $\boldsymbol{s}_j$
Weight $\boldsymbol{s}_j$ to produce $\dot{\boldsymbol{s}}_j$
Compute the predicted rating as $\hat{r}_{ij} = \dot{\boldsymbol{s}}_j \cdot \boldsymbol{r}_i$
Transform $\hat{r}_{ij}$ to the original rating scale ▷ only when using $\eta(\boldsymbol{R})$ and $\boldsymbol{S}^{v}_{\eta}$ or $\zeta(\boldsymbol{R})$ and $\boldsymbol{S}^{v}_{\zeta}$

Furthermore, the predicted rating $\hat{r}_{ij}$ is influenced by the number of similar wines k contained within $\boldsymbol{s}$ and $\boldsymbol{r}$. Basing predictions on too few wines may lead to inaccuracies as the prediction is biased towards the rating of a small subset. Likewise, calculating a predicted rating based on a large number of wines may too be suboptimal as $\boldsymbol{r}$ may contain ratings for wines which are not highly similar to the wine j.

The choice of the k most similar wines, however, is only applicable to users who have rated a large number of wines. If the number of wines rated by a user $|V_i|$ is less than k, the $\boldsymbol{s}$ and $\boldsymbol{r}$ vectors are limited to the number of ratings made. The effects of this are most significant for users who have made only one rating (single-users). When $|V_i| = 1$, $\boldsymbol{r}$ contains the single rating made by that user, and despite the contents of $\boldsymbol{s}$, after weighting, $\hat{r}_{ij}$ will equate to the user's existing single rating.

Similarly, when $k = 1$ and $|V_i| > 1$, the predicted rating will be the rating that the user awarded to the most similar wine in V_i. This can be seen as the main flaw of Algorithm 4 - as a result of sparse user ratings, predictions are severely biased to past ratings only. A potential solution in the case that either $k = 1$ or $|V_i| = 1$ is to skip the weighting of $\boldsymbol{s}$ and use unscaled similarity scores to generate a prediction. For example, if $\boldsymbol{s} = \{0.6\}$ and $\boldsymbol{r} = \{3.5\}$, rather than weighting $\boldsymbol{s}$ such that it sums to 1 (producing $\dot{\boldsymbol{s}} = \{1\}$) and predicting $\hat{r}_{ij} = 1 \times 3.5 = 3.5$, the predicted rating could be calculated as $\hat{r}_{ij} = 0.6 \times 3.5 = 2.1$. Alternatively, the mean rating of the wine j could be returned in cases where $k = 1$ or $|V_i| = 1$.

Despite this, Algorithm 4 is not altered for single-users or for $k = 1$. When a predicted rating is chosen as the mean of a wine's ratings, the prediction ignores the biases in the user's own rating scale and does not account for users who often rate below and above the mean rating. Additionally, predicting ratings using unweighted similarity will not lead to realistic predictions as many unscaled similarity scores are close to 0 and this would irrationally bias rating predictions to values of 0 and 1.

4.1.2 Top-Q wine recommendations using wine similarity

In systems built on the premise of user or wine similarity, recommending a list of the top-Q wines involves returning a list of wines with the greatest predicted rating that are unrated by the user. Typically, this involves populating $\boldsymbol{R}$ with predicted ratings such that no observations are missing; this is straightforward for model-based approaches (see Section 4.1.3), but requires more consideration for CF and CB systems.

Algorithm 4 is adequate when predicting a single rating at a time, but is inefficient when predicting ratings in bulk. If Algorithm 4 is used to produce a list of top-Q wines for a user, it would be repeated $m = 1,640$ times to generate a predicted rating for each wine before ranking and selecting the Q highest predicted ratings. In the wine dataset, producing top-Q lists with Algorithm 4 would involve $92,514 \times 1,640 = 151,722,960$ rating predictions which are infeasible if it is to be repeated numerously to compare multiple systems.

Furthermore, Algorithm 4 is incapable of producing a ranked list of wines for single-users and users with a rating standard deviation of 0. Recall that $\boldsymbol{r}_i \subseteq \boldsymbol{R}_{i\cdot}^+$, thus $\min \boldsymbol{R}_{i\cdot}^+ \leq \boldsymbol{r}_i \leq \max \boldsymbol{R}_{i\cdot}^+$. As $\sum \dot{\boldsymbol{s}}_j = 1$, a predicted rating cannot be greater than the maximum or less than the minimum rating that user i has previously made, as $\hat{r}_{ij}$ is defined to be bound by the user's previous ratings.

Hence, when $|V_i| = 1$ or when $\min \boldsymbol{R}_{i\cdot}^+ = \max \boldsymbol{R}_{i\cdot}^+$ (i.e. the user has only awarded 1 rating multiple times), $\hat{r}_{ij}$ will be the same for all wines. In these common cases, wines cannot be ranked to produce top-Q lists as they all share the same $\hat{r}_{ij}$. Thus, Algorithm 5 is used instead.

From Algorithm 5, single-users are recommended the top-Q wines which are most similar to the single wine they rated. For users who awarded multiple wines the same rating, their top-Q list consists of the Q wines most similar to the set of wines they rated. For all other users, the top-Q list is determined using their favourite wines only. Favourite wines are defined as those which a user i awarded a rating greater than or equal to their mean rating $\bar{r}_i$. Hence, top-Q recommendations are those which are most similar to the user's favourite wines.

As Algorithm 5 is dependent on similarity alone, the measure used to determine similarity is what determines the quality of a top-Q list. Similarity and distance measures which do not discriminate well between wines, and award many wine pairs the same similarity score, for example, will result in tied top-Q lists. Such lists are less informative as the first wine recommended should be of more interest to the user than the Q^{th} wine.

This highlights how the ranking produced by a similarity measure should not be arbitrary and directly influences the quality of recommendation lists produced with Algorithm 5. The measures used to evaluate top-Q recommendations are discussed further in Section 4.2.2.

Algorithm 5 CF Top-Q Recommendations (User i, wine j, Ratings matrix $\boldsymbol{R}$, Similarity Matrix $\boldsymbol{S}$)

$V_i \leftarrow$ the set of wines rated by i
$\boldsymbol{R}_{i\cdot}^{+} \leftarrow$ the ratings made by i
if $|V_i| = 1$ **then** ▷ single-users
  $j \leftarrow$ the single wine rated
  Select $\boldsymbol{s}_j$ from $\boldsymbol{S}_{\cdot j}$ ▷ the similarities between j and all other wines
  Rank $\boldsymbol{s}_j$
  return the top-Q wines in $\boldsymbol{s}_j$
else
  if $\min \boldsymbol{R}_{i\cdot}^{+} = \max \boldsymbol{R}_{i\cdot}^{+}$ **then**
    Select $\boldsymbol{s}_{V_i}$ from $\boldsymbol{S}_{\cdot V_i}$ ▷ the similarities between the wines in V_i and all other wines
    Rank $\boldsymbol{s}_{V_i}$
    return the top-Q wines in $\boldsymbol{s}_{V_i}$
  else
    $\bar{r}_i \leftarrow \text{mean}(\boldsymbol{R}_{i\cdot}^{+})$ ▷ the mean rating made by i
    $\bar{V}_i \leftarrow$ the set of wines rated by i with a rating $\geq \bar{r}_i$ ▷ i's favourite wines
    Select $\boldsymbol{s}_{\bar{V}_i}$ from $\boldsymbol{S}_{\cdot \bar{V}_i}$ ▷ the similarities between the wines in $\bar{V}_i$ and all other wines
    Rank $\boldsymbol{s}_{\bar{V}_i}$
    return the top-Q wines in $\boldsymbol{s}_{\bar{V}_i}$
  end if
end if

4.1.3 Model-based recommendations

In the case of matrix factorisation, predicting a user's rating for a wine is as simple as linearly combining the user and wine latent factors $\hat{r}_{ij} = \sum_{c=1}^{x} u_{ic} \cdot v_{jc}$. Likewise, producing top-Q recommendations is achieved by recomposing the ratings matrix using the latent factors $\boldsymbol{U}\boldsymbol{V}^T$. Recommendation lists are the Q wines with the greatest ratings per row of the recomposed matrix.

Contrarily, neural network recommender systems trained to predict ratings can only produce a single $\hat{r}_{ij}$ value at a time. As there is no underlying similarity component, Algorithms 4 and 5 cannot be used and producing top-Q recommendations can only be achieved by predicting all of the ratings missing in $\boldsymbol{R}$. Although inefficient, once a matrix of dense rating predictions is formed, the Q wines with the highest predicted ratings are returned as a list of recommendations.

4.2 Evaluation Metrics

The efficacy of a recommender system can be approached from both an accuracy and quality perspective. *Accuracy* is a measure of the correctness of a single recommendation determined through the error between an observed and predicted rating. The *quality* of recommendations, however, more subjectively measures how *good* a list of recommendations is. Measures of quality are of course dependent on the user receiving recommendations but can be estimated empirically. Quality metrics concern issues such as "are all wines recommended equally?"

(coverage), "how different are the recommendations to each other?" (personalisation), and "are the most important wines recommended first?" (average reciprocal hit rate).

4.2.1 Accuracy of a predicted rating $\hat{r}_{ij}$

Accuracy is measured as the Root Mean Square Error (RMSE) between a predicted rating $\hat{r}_{ij}$ and an observed rating r_{ij}, calculated as:

$$RMSE = \sqrt{\frac{\sum_{(i,j)\in\nu}(r_{ij} - \hat{r}_{ij})^2}{|\nu|}} \tag{4.2.1}$$

where $\nu = \{(i,j) : r_{ij} \text{ is observed but hidden}\}$ contains the ratings not used for training - observations in either a validation or testing set (Aggarwal, 2016).

The main criticism of the RMSE is that it is arguably an unreliable measure of the mean error. The squared error term ensures that large differences between observed and predicted ratings are penalised more greatly. Large differences between observed and predicted ratings may be exceedingly common for sparsely rated wines as the less data available for a model to base a prediction on, the more likely the resulting prediction will be inaccurate. The use of RMSE may then be undesirable in cases where outlier predictions should not be more punitive to the measured accuracy of a system than minor errors (Ricci et al., 2011).

The Mean Absolute Error (MAE), alternatively, penalises all errors equally by considering the absolute difference between predictions and observations only:

$$MAE = \frac{\sum_{(i,j)\in\nu} |r_{ij} - \hat{r}_{ij}|}{|\nu|} \tag{4.2.2}$$

The greatest possible single error produced by the systems built in this thesis would yield an RMSE of $(5-1)^2 = 16$ and an MAE of $|5-1| = 4$. It is decided that large errors should be considered a more serious issue than smaller errors. If a user who rated a wine with a 1 was recommended that bottle with a predicted rating of 5, the system should be considered largely inaccurate; hence, the RMSE metric is chosen over MAE as the measured accuracy of a system should be strongly diminished by large errors.

Importantly, RMSE values can only be compared when computed with data measured on the same scale. The RMSE between the unscaled ratings $\{5, 1\}$ is not equal to the RMSE between the equivalent normalised ratings $\{1, 0\}$. Thus, to avoid biased results, the RMSE values reported are calculated using ratings in the original scale, unless stated otherwise.

4.2.2 Quality of top-Q recommendations

Many metrics exist to evaluate the quality of a list of recommendations - the following used in this thesis operates on the concept of relevancy. **Relevant wines** are those that a user has rated positively in ν, the set of ratings not available to train with. Positive ratings for a user are those $\geq$ the user's mean rating $\bar{r}_i$. Consequently, **irrelevant wines** are those which the user has not positively rated and **relevant users** are those with relevant wines in the validation and testing sets.

The metrics are applied to sets of relevant wines only and therefore are only computed for relevant users. This greatly reduces the number of calculations to perform as there are only 32,915 relevant users out of the total 92,514. In the following equations, U is the set of relevant users, V is the set of wines, and V_i^Q is a list of Q wines recommended to user i.

- **Coverage**

  Coverage measures the percentage of the wine catalogue recommended to users and indicates if a system recommends a diverse array of wines or just the most popular. Using the list of Q wines recommended to user i, V_i^Q, coverage is computed as:

  $$\text{COV} = \frac{|\bigcup_{i=1}^{|U|} V_i^Q|}{|V|} \tag{4.2.3}$$

  which is the length of the union of all recommendations for all users as a fraction of the number of wines. Coverage assesses the diversity of recommendations between users to an extent as to achieve 100% coverage, each wine in V should be recommended at least once (Aggarwal, 2016).

- **Personalisation**

  Personalisation explicitly measures how unique each top-Q list is to each user; instead of receiving the same set of recommendations, users should receive a list which is bespoke. The personalisation formula devised computes the fraction of wines shared between the top-Q lists V_a^Q and V_b^Q of two users a and b for all pairs of users:

  $$\text{PER}(Q) = 1 - \frac{1}{|U|}\left(\sum_{(a,b)\in U, a\neq b} \frac{|V_a^Q \cap V_b^Q|}{Q}\right) \tag{4.2.4}$$

  where $(a, b) \in U, a \neq b$ are all distinct pairs of users a and b in U. When the number of wines shared by V_a^Q and V_b^Q is large proportional to Q, the personalisation decreases. The more dissimilar V_a^Q and V_b^Q are for all pairs of users, the greater the personalisation score.

  Computing the personalisation score for all relevant users would impractically require the repeated comparison of $n(n-1)/2 = 32,915(32,914)/2 = 541,682,155$ top-Q lists for increasing values of Q. Instead, personalisation scores are approximated by taking the mean personalisation scores for user subsets of size $10,000$, which greatly reduces the computational load.

- **Receiver Operating Characteristic (ROC) curve**

  Classification errors are categorised in confusion matrices which record true or false, positive and negative classifications. In a recommendation context, the confusion matrix in Table 4.1 indicates how different recommendations in a top-Q list may be categorised. Importantly, a relevant recommendation is recorded as a true positive and the relevant wines not recommended are false negatives.

Table 4.1: Confusion matrix of recommendations in a top-Q list.

	Relevant	Irrelevant
Recommended	True Positive (TP)	False Positive (FP)
Not Recommended	False Negative (FN)	True Negative (TN)

Using Table 4.1, the True Positive Rate (TPR) and False Positive Rate (FPR) of the recommendations made can be calculated as:

$$\begin{aligned} TPR &= \frac{TP}{TP + FN} \\ FPR &= \frac{FP}{FP + TN} \end{aligned} \tag{4.2.5}$$

TPR values close to 1 are ideal as the majority of relevant wines are recommended; likewise, FPR values closer to 0 are preferred as fewer irrelevant wines are recommended. TPR and FPR values for increasing values of Q plot together produces a Receiver Operating Characteristic (ROC) curve. A perfect RS would produce a ROC curve which lies as close to the y-axis (low FPR) and as far from the x-axis as possible (high TPR). Such plots are useful in measuring, for different sizes of top-Q lists, how well the recommendations suit the user's wine preferences (Schröder et al., 2011).

- **Average Reciprocal Hit Rate (ARHR)**

  A *hit* is counted when a relevant wine is recommended, or a true positive value as in Table 4.1 is recorded. The hit rate is then the average number of hits per user:

$$\text{Hit Rate} = \frac{\#\text{ hits}}{|U|} \tag{4.2.6}$$

  The hit rate does not account for the ranking of the wines within the recommended list, however. The position of recommendations is incorporated in the Average Reciprocal Hit Rate (ARHR):

$$\text{ARHR} = \frac{\sum_{\text{hit} \in \text{hits}} \text{rank(hit)}^{-1}}{|U|} \tag{4.2.7}$$

  Using this metric, the rank-quality of top-Q lists is established as both the presence and position of relevant wines in the recommendations are measured. Hence, higher ARHR values correspond to top-Q lists that suggest relevant items before irrelevant ones. For example, a hit ranked first in a recommendation list will increase the ARHR value, whereas a hit with a rank of 20 would marginally increase the value. Likewise, when few hits are recorded, the ARHR value is reduced by the large denominator (Shi et al., 2012).

  It is important to note that values for the ARHR and TPR metrics are expected to be very low. It is very challenging, given the high sparsity of this wine rating dataset, for any RS to obtain large numbers of hits as the number of relevant wines is very low. Using equation (4.2.7), if all of the top-Q wines recommended to each user in $|U|$ are hits (i.e. $\#\text{ hits} = |U| \cdot Q$), then the maximum possible ARHR is:

$$|U|\Big(\frac{1}{1} + \frac{1}{2} + \cdots + \frac{1}{Q}\Big)/|U| \tag{4.2.8}$$

In the wine dataset, there are 32,915 relevant users - a perfect RS would then theoretically only be able to achieve a maximum ARHR of 3.6 when $Q = 20$.

This highlights how the results of these metrics are diminished by the sparsity of the wine data and can only be used compared between systems. That is, recommendations produced by a system yielding an average hit rate close to 0 should not be thought of as low quality, but should first be compared to those produced by other systems to accurately gauge performance.

These evaluation metrics can be used together to compare the overall performance of various recommender systems. For all methodologies, the best-performing systems are selected based on their validation RMSE. Further examination of the quality of recommendations made is used to determine the optimal system for each method.

4.3 Item-Based Collaborative Filtering

As noted by Sarwar et al. (2001), and as a result of the sparsity of the ratings matrix, item-based collaborative filtering is applied instead of user-based CF for increased accuracy in rating predictions. IBCF leverages the fact that wines can be more accurately compared based on the groupings of users who rate them when the number of users far exceeds the number of wines.

The mean number of wines rated per user is 2 and the mean number of ratings awarded per wine is 119; as a result, more meaningful comparisons can be made between the ratings per wine (the columns of $\boldsymbol{R}$) than ratings per user (the rows of $\boldsymbol{R}$). It follows that the similarity between wines is well-defined by their consumers, but the similarity between users is not as well-defined by their rating preferences alone.

Another motivation for the use of IBCF over UBCF is scalability. The pairwise calculations of similarity matrices are computationally expensive operations; computing the upper-triangle[1] of the user similarity matrix $\boldsymbol{S}^u$ would involve $n(n-1)/2 = 92,514(92,513)/2 = 4,279,373,841$ calculations - an extremely lengthy procedure.

To address this, the number of similarity calculations required can be reduced through clustering (like that described by Almasi and Lee (1999) in Appendix A.6.2) where user similarity is only calculated within smaller clusters. However, as explored by Xue et al. (2005), the performance of a simple cluster-based approach to UBCF was more time-efficient but compromised on predictive accuracy. Consequently, clustering as a solution to poor UBCF scalability is not explored further.

Considering that computing $\boldsymbol{S}^u$ is expensive and sensitive to rating updates, UBCF provides an unfavourable basis for an RS model in this context. Therefore, the similarity is calculated between wines by applying the similarity measures described in Section A.1 to the wine columns of $\boldsymbol{R}$ to form the item-based CF similarity matrix $\boldsymbol{S}^v$. To investigate the performance of each similarity measure and to establish an optimal value for k in the rating prediction function (see Algorithm 4), each measure is used to predict the ratings in the validation set for values of $k \in \{1, 5, \ldots, 20\}$ and the accuracy of these rating predictions is determined through the RMSE (see Section 4.2.1 for details).

[1] $\boldsymbol{S}^u$ is symmetric about a diagonal of 1's, hence only the upper or lower region needs to be computed.

Then, to explore the effects of scaling, wine similarity is calculated for unscaled, normalised and standardised rating matrices, $\boldsymbol{R}$, $\eta(\boldsymbol{R})$ and $\zeta(\boldsymbol{R})$. When calculating the RMSE of the predictions made using scaled ratings, values are transformed to the original scale beforehand so that errors are comparable between the methods of scaling. Moreover, the validation set of ratings is predicted for outcomes of both cold-start management protocols $\boldsymbol{R}^{\text{CDF}}$ and $\boldsymbol{R}^{k\text{NN}}$.

Likewise, to assess the quality of recommendations yielded for varying similarity measures, values of k, scaled datasets and cold-start protocols, the top-Q recommended wines are determined using Algorithm 5 for relevant users in the validation set for $Q \in \{1, \ldots, 20\}$ and evaluated using the quality metrics detailed in Section 4.2.2.

4.4 Content-Based Methods

Wine similarity can be determined through the content of the wine instead of their rating patterns; the rating prediction method outlined in Algorithm 4 can be applied as normal, the only difference being the use of the wine similarity matrices $\boldsymbol{S}^{v:s}$ and $\boldsymbol{S}^{v:\not s}$. Structured content-based similarity is determined using the numeric and factor attributes of the wines and unstructured CB similarity is computed with the descriptions of the wines.

4.4.1 Structured wine attribute similarity

The structured numeric variables and factor variables of the wines (`Winery`, `Year`, `Grape`, `Type`, `Style`, `Body`, `Price`, `Alcohol`, `Residual Sugar`, `pH`, and `Tannins`) are compared[2] using the distance measures described in Appendix A.2 to compute the wine similarity matrix $\boldsymbol{S}^{v:s}$.

$\boldsymbol{S}^{v:s}$ is then used with the rating prediction function (Algorithm 4) to generate recommendations as with the IBCF method described above. It is worth noting that, unlike IBCF, rating scaling does not affect the predictions made as the rating data is not involved in computing content-based similarity. Hence, the effects of scaling are not measured in the content-based approach.

To determine the optimal structured CB approach, for each dataset, $\boldsymbol{R}^{CDF}$ and $\boldsymbol{R}^{kNN}$, the validation set ratings are predicted for varying numbers of similar wines $k \in \{1, 5, \ldots, 20\}$ and distance measures. As above, the accuracy of these predictions is measured using RMSE, and the top-Q recommendations determined with Algorithm 5 are evaluated using the quality metrics in Section 4.2.2.

4.4.2 Unstructured wine description similarity

To compute $\boldsymbol{S}^{v:\not s}$, the similarity between the unstructured descriptions of wines, the free text must be converted to a vectorised representation. Term Frequency - Inverse Document Frequency (TF-IDF) calculations (as described in Appendix A.3) are used to convert the descriptions of each wine to a numeric vector of fixed length as follows.

Firstly, stop words (e.g. "is", "and", "the") are removed, followed by common wine terms which are already captured by other variables. For example, terms such as "wine", "taste" and

[2] As the numeric variables in the wine dataset are all measured on different scales and are not assumed to be normally distributed (refer to Section 3.4), the numeric variables are normalised before applying the distance measures.

"style" appear frequently but are not useful in discriminating between wines and are removed from the descriptions.

Next, stemming is applied to the descriptions, reducing each word to a stem representation. For example, words such as "balanced", "appley" and "concentrated" are reduced to "balanc", "appl" and "concentr", respectively. This process only slightly diminishes the readability of the description to generalise the descriptions further.

After that, each description is broken into two representations, one which contains each word as a separate entity and one which contains each bigram (pair of words) of the description. Dividing the descriptions into words is important in capturing standalone terms like "elegant" and "rich"; bigrams, however, are needed to capture concepts better represented by word pairs such as "medium body" and "black cherry". Thus, a combination of both words and bigrams is required in the vectorised representation of the wine descriptions to maximise how representative they are. Henceforth, words or bigrams are collectively referred to as tokens.

Naturally, some words and bigrams represent the same concept which introduces redundancies; to account for this, the more appropriate token in a pair of semantically similar tokens is retained while the redundant token is removed. For example, the bigram "berry fruit" is more suitably represented by the word "berry" as a berry is a fruit and it is more likely that most wine descriptions would refer to the word "berry". Likewise the bigram "passion fruit" is a more appropriate token when compared to the words "passion" and "fruit" on their own. On the other hand, bigrams such as "red fruit" are kept along with words like "red" as "red fruit" is a term that could be found in the descriptions of sparkling, rosé and red wines, whereas the word "red" is more likely to describe the grape, colour or type of wine.

Once each wine description is reduced to a set of tokens, the vectorised representations should be limited to contain only the most common and most discriminative tokens such that the vectors can be reliably compared. Pazzani and Billsus (1997) found that the optimal number of tokens used to represent a body of text should be between 50 and 300. Importantly, the optimal number of tokens should be selected such that each wine's description contains at least one of the chosen tokens, otherwise, not all wines could be represented by that token set.

Metrics such as the "Gini Index" and "Normalised Deviation" measure how well a token can discriminate between ratings assigned to bodies of text (see Appendix A.3 for more details) and can be used to select the top tokens used to represent a description. After calculating these metrics, the tokens which yield the most discriminative power are found to be niche, uncommon terms. When selecting 300 tokens with the lowest Gini index or greatest normalised deviation respectively, only a subset of the wine descriptions contains those tokens. Therefore, using those token selection methods would limit the number of wines to compare; so instead, tokens are selected based on frequency.

The frequency that each token appears throughout all the wine descriptions is used to select the optimal number of tokens. The top 166 most frequently used tokens appear in all wine descriptions at least once. In other words, each wine description contains at least one of the top 166 most frequently used tokens and thus those tokens are chosen to represent all wine descriptions.

Figure 4.1 is a word cloud representing the 166 most frequently used tokens with size indicating the relative frequency. For each of these tokens, the TF-IDF weight per wine description is

calculated. This produces is a matrix of 1640 wine rows and 166 token columns with each value corresponding to the TF-IDF weight of that term within that wine's description.

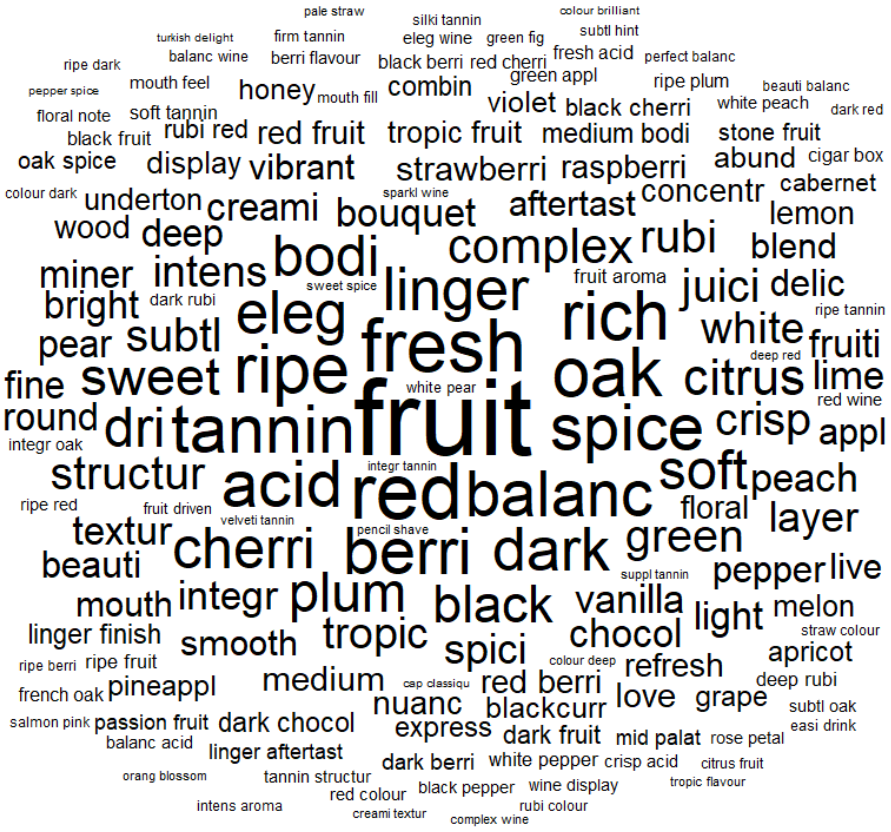

Figure 4.1: Word cloud of the 166 most frequently used tokens in the wine descriptions.

Then, using the similarity measures described in Appendix A.1, the similarity between each wine is calculated[3] using their vectorised representations to form the unstructured wine similarity matrix $\boldsymbol{S}^{v:\not\phi}$.

As detailed in Section 2.1.3, user profiles could be computed as $\boldsymbol{x}_i = \sum_{j \in V_i} r_{ij} \boldsymbol{y}_j$, where $\boldsymbol{x}_i$ is a profile consisting of user i's affinity towards each of the tokens describing the wines, $\boldsymbol{y}_j$. A user-wine similarity matrix $\boldsymbol{S}^{uv}$ can then be computed by comparing user profiles to the vectorised representations of wines allowing for a user-based approach to rating predictions. This method is avoided, however, as due to the computational complexity of computing such a similarity matrix ($92,514 \times 1,640 = 151,722,960$ calculations), some form of user clustering would be required which is posed to diminish predictive accuracy (Xue et al., 2005).

Instead, $\boldsymbol{S}^{v:\not\phi}$ is computed by comparing the wine description vectors for use in Algorithm 4. Then, for the various similarity measures applied to the wine vectors used to compute $\boldsymbol{S}^{v:\not\phi}$ and for increasing numbers of similar wines $k \in \{1, 5, \ldots, 20\}$, the RMSE of the predicted ratings of the validation set. Using Algorithm 5 and the quality metrics in Section 4.2.2, the top-Q recommendations are recorded and evaluated too.

[3] As the WPC and NHSM measures are used explicitly for rating data, they are not applied in this context.

4.5 Matrix Factorisation

The `R recosystem` package (Qiu et al., 2021) provides stochastic gradient descent functionality to solve for the latent user and wine matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ as detailed in Algorithm 1. When solving for these matrices, hyperparameters to tune include the dimension x of the latent matrices, L1 and L2 regularisation for both $\boldsymbol{U}$ and $\boldsymbol{V}$ as well as the learning rate α applied in the gradient descent algorithm.

The training data is split into 10 folds and a grid search is performed using `recosystem` to determine the optimal hyperparameter combinations which yielded the lowest cross-validated RMSE when decomposing the training set matrices $\boldsymbol{R}^{CDF}$ and $\boldsymbol{R}^{kNN}$. Hyperparameter values compared are: $x \in \{1, 5, 10, \ldots, 30\}$, learning rate $\alpha \in \{0.1, 0.01, 0.05\}$, L1 and L2 regularisation for both user and wine matrices $\in \{0, 0.01, 0.05, 0.1\}$.

To explore the effects of scaling the ratings matrix before decomposition, the SGD algorithm is applied to the unscaled, normalised and standardised versions of the training sets. The cross-validated RMSE values of each hyperparameter combination are used to determine the optimal model configurations and the latent matrices for the unscaled, normalised and standard versions of $\boldsymbol{R}^{CDF}$ and $\boldsymbol{R}^{kNN}$ are saved.

Using the optimal latent matrices, each matrix $\boldsymbol{R}$, $\eta(\boldsymbol{R})$ and $\zeta(\boldsymbol{R})$ is recomposed and transformed to the original rating scale where necessary. As the recomposed matrices contain no missing values, they can be used to compute both a training and validation RMSE as a measure of the efficacy with which SGD can recompose the training matrix. Obtaining the top-Q recommendations per user involves extracting the Q highest values per row from the recomposed matrices - these are then used to measure the quality of the recommendations with the metrics in Section 4.2.2.

Next, the `R sparklyr` package (Luraschi et al., 2022) is used to perform efficient ALS matrix decomposition as described in Algorithm 2. In ALS, the hyperparameters tuned using a grid search are the dimension of the latent factors $x \in \{1, 5, 10, \ldots, 30\}$ and total regularisation applied to the factors $\lambda \in \{0.01, \ldots, 0.05, 0.1, \ldots, 0.5\}$. Unlike the cross-validation used in `recosystem`, the ALS algorithm is applied to the training ratings for a maximum of 10 iterations (or until convergence is reached) and the optimal hyperparameter combination is determined using the validation RMSE calculated when recomposing using the resulting latent matrices.

Again, the ALS method is applied to both $\boldsymbol{R}^{CDF}$ and $\boldsymbol{R}^{kNN}$ for each version of data scaling. Likewise, the RMSE of the training and validation set are computed using the optimal latent matrices and the quality of the recommendations is calculated by applying the quality metrics (Section 4.2.2) to the top-Q wines of each row in the latent matrices.

Note that when applying either the SGD or ALS algorithm to the standardised data, the non-negative constraints of the loss function in equation (2.1.8) are dropped as $\zeta(\boldsymbol{R})$ contains negative values.

4.6 Multiple-Input Neural Networks

To maximise the potential of complex NN designs, several networks receiving multiple inputs are developed and compared. In the case of the wine dataset, four potential inputs exist: a user rating input, a wine rating input, and both a structured and unstructured wine data input.

The user and wine rating inputs are used to express the rating patterns made by a user and awarded to a wine, respectively. The structured and unstructured wine inputs, therefore, correspond to the numeric, categorical and textual descriptions which are available to describe each wine (see Tables 3.1 and 3.2).

Taking inspiration from the multi-input architecture illustrated in Figure 2.4, these user and wine inputs are combined in a NN to produce user and wine latent representations which are used to predict ratings, as seen in Figure 4.2.

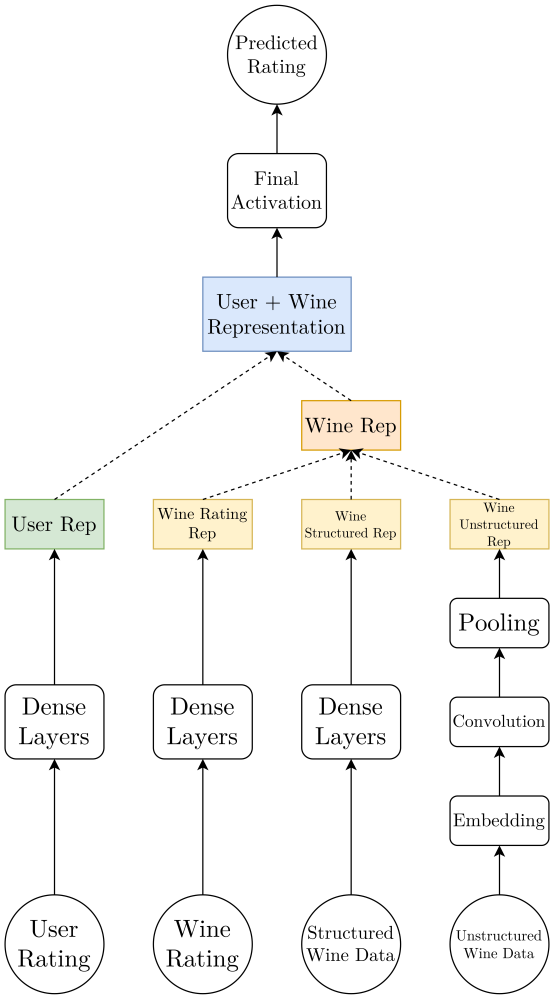

Figure 4.2: The multi-input neural network wine recommender system architecture.

The network in Figure 4.2 consists of four parallel networks (or portions), each of which receives a different input, propagates it through a series of densely connected layers and produces a corresponding representation ("rep" for short). The output representations of the wine portion

of the network are concatenated into a single wine representation, Wine Rep = {Wine Rating Rep, Wine Structured Rep, Wine Unstructured Rep}, indicated by the dashed arrows in Figure 4.2. The wine representation is then concatenated with the user representation (User+Wine Rep = {User Rep, Wine Rep}), created by propagating user ratings through a separate set of dense layers. The combined User+Wine representation allow for the outputs of each parallel component to be propagated together through one final activation function before producing a predicted rating for a given user and wine.

As each parallel component of the network operates on different data, the design of each varies and is detailed below. Importantly, as the structure of the multiple inputs is vastly different, measured on different scales and cannot be assumed to be normally distributed, the inputs are normalised to ensure stable learning.

4.6.1 User and wine rating component

To model specific user preferences and global wine preferences, both the ratings made by a user i and all the ratings made for wine j can be used as respective inputs. Simply, the user and wine rating inputs could be the entire user row and wine column of the ratings matrix $\boldsymbol{R}_{i\cdot}$ and $\boldsymbol{R}_{\cdot j}$, respectively. This approach poses two issues - firstly, in the wine dataset, each $\boldsymbol{R}_{\cdot j}$ contains 92,514 values corresponding to each user. This presents memory issues when propagating such a large vector through the network. Secondly, the vectors $\boldsymbol{R}_{i\cdot}$ and $\boldsymbol{R}_{\cdot j}$ contain mostly missing values (which would be converted to 0's) which limits the expressiveness of those inputs. This is a particular issue for single users, where their rating input vectors would only be distinguished between each other by a single value in a vector of 1,639 0's.

Hence, smaller, more expressive user and wine rating inputs are required. One approach considered is reducing the user rating input to those made for the 50 most popular wines and reducing the wine rating input to those awarded by the 50 most active users. Initial experimentation yielded poor results as this reduction discards information in the case where users did not rate any of the top 50 wines or wines were rated by more than just the most active users.

Instead, a form of *feature augmentation* (see Section 2.1.6) is applied whereby the optimal user and wine latent factor representations determined through matrix factorisation in Section 4.5 are used as inputs which reliably capture the underlying user preferences and wine rating distributions. These latent representations are shown to accurately reduce the dimension of the large, sparse ratings matrix and as a result, the user and wine input rating input vectors share the same dimension x as determined by the optimal latent representation of $\boldsymbol{R}$.

4.6.2 The structured wine data component

The structured (numeric and categorical) variables in the wine dataset (see Tables 3.1/ 3.2) are numerically encoded before being fed to the NN. Numeric variables are normalised and, in the case of the `Year` variable, NV (non-vintage) values are replaced as the median year of 2018 of all wines. The remaining categorical variables are then one-hot encoded as integer encoding would erroneously imply an ordering to the categories. Additionally, to reduce the size of the structured wine vector input, the number of factor levels is reduced.

The numerous factor levels of the `Location` and `Grape` variables are reduced according to

classifications provided by Puckette and Hammack (2018), from 37 to 8 ("Breede River Valley", "Cape South Coast", "Cedeberg", "Coastal Region", "Klein Karoo", "Northern Cape", "Olifants River" and "Western Cape") and 49 to 8 ("Blend", "Citrus White", "Golden White", "Herbal White", "Perfume White", "Soft Red", "Spicy Red", "Tannic Red"), respectively. As these new grape classifications capture the grape varietal, all the levels of the `Type` variable are no longer needed.

As "red" and "white" is captured by the new grape classifications, the `Type` variable is reduced to "Rosé", "sparkling" and "Dessert". After creating indicator variables for `Body` and `Style` and including the numeric variables, the resulting structured wine input is a vector consisting of 31 values.

4.6.3 The unstructured wine data component

To capture all of the unstructured wine data available, the written descriptions of the wines, any food pairing suggestions, and user reviews are combined to create documents that reliably represent the wines. User reviews could not be used independently as in the training set, only 77,310 users made a review with an insufficient median length of 6 words. Instead, when considering all user reviews written for a single wine together, the median total review length is far more informative at 358 words.

Further, combining wine reviews and descriptions allow for a more complete document describing a wine. Although reviews and descriptions have different sentiments and cover different topics, where a description may lack detail, wine reviews may impute any missing detail and vice versa.

After combining all the user reviews for a wine along with their description and any food suggestions (in that order) into a single document, the following pre-processing steps were taken[4].

Stop words are removed from the document, but stemming is not applied as appropriate embeddings do not exist for word stems. Though the embeddings of the wine documents could be learnt, the pre-trained GloVe embeddings determined using a Twitter dataset consisting of 2 billion tweets and a vocabulary of 1.2 million words are used instead (Pennington et al., 2014). These embeddings are far more reliable than those which could be learnt using the (comparably) small wine dataset and are appropriate as the semantics of tweets are similar to those of short reviews when compared to other embeddings such as those pre-trained on Wikipedia, for example.

Next, words for which no GloVe embedding exists are removed. Then, context-specific words defined as those with a document frequency greater than 0.5 are removed as they are too commonly used to be helpfully distinguishable (Kim et al., 2016). The TF-IDF weights of the words in each document and the total TF-IDF weight of each word is computed; the words with the greatest total TF-IDF weight are then the most informative and should be retained in the vocabulary.

Unlike applying TF-IDF as detailed in Section 4.4.2, the set vocabulary used when embedding does not have to be as small as a few hundred words. When using TF-IDF, each document is

[4]Similar to the approaches of Wu et al. (2019), Kim et al. (2016), and Wang et al. (2015).

represented by vectors $\boldsymbol{d}$ of equal length where each value corresponds to the weight of each word in the vocabulary in that document. When using embeddings, each document is represented by a matrix $\boldsymbol{D}$ of a fixed number of rows ρ as well, but each row corresponds to any word in a much larger vocabulary. Not only does this allow for more freedom when selecting words for a vocabulary (as the vocabulary size does not explicitly dictate the size of the matrix), but the ordering of words is maintained.

After stop word removal, the median wine document length is 312 words. The number of rows in the embedding matrix is then set to $\rho = 300$ and, for each wine document, is populated with a vocabulary of the 5,000 words with the greatest TF-IDF weights. Documents with less than 300 words are padded with 0's when embedding, and those documents with more than 300 words are amputated in the same manner as Wu et al. (2019).

The depth of the embedding matrices b is determined through hyperparameter tuning along with the number of filters applied when convolving $\boldsymbol{D}$ in the rightmost portion of the network in Figure 4.2.

4.6.4 Hyperparameter tuning

Developing an optimal NN recommender involves tuning the network size (i.e. the number of nodes and hidden layers in each parallel portion of Figure 4.2), the learning rate applied in the back propagation (see Algorithm 3), the regularisation of the network, the dimension b of the embedding matrices, the width h and number of the filters applied in the convolution layer, the activation functions used, and finally the batch size and number of epochs used during learning.

Combinations of the above hyperparameters can result in endless different network configurations. To reduce the search space for the optimal network, some hyperparameters are fixed whilst others are compared through grid searches.

4.6.4.1 Fixed parameters

Experimentation was conducted with a baseline model to determine the appropriate batch size, number of epochs, number of filters and embedding dimension to remain constant before conducting grid searches for the remaining hyperparameters. Performance of the baseline NN is measured using the validation RMSE and corresponding plots may be found in Appendix A.9.

The baseline model is designed such that is resembles the simplest form of Figure 4.2. Hence and taking inspiration from Zhang and Wallace (2015), the baseline consists of 1 hidden layer with 15 nodes using the reLu activation function in each of the user and wine rating input and structured wine input portion of the network (the sections labelled "Dense Layers" in Figure 4.2). The convolutional layer applies 100 filters of width $h = 3$ and an embedding dimension $b = 50$. The learning rate is set at 0.001, no regularisation is applied and the final activation function is reLu as well.

Holding all other hyperparameters constant, the baseline model is trained with batch sizes $\in \{50, 100, 500\}$ and the number of epochs $\in \{1, \ldots, 20\}$. From Figure A.4, it is found that smaller batches (50 and 100) find a lower training RMSE in fewer epochs, but produce more fluctuating validation errors. The larger batch size of 500 produces high errors in the first epoch

which decrease rapidly and consistently with each proceeding epoch. Despite the performance on the first epoch, a batch size of 500 leads to the most stable validation errors for 10 epochs (after which the chances of overfitting to the training data increase) and is selected as optimal.

Next, the baseline model is trained with the number of filters applied $\in \{100, 200, 300\}$ when convolving. Though 300 filters yields the best results in Figure A.5, the lower errors are negligible compared to the model trained with 100 filters. Such large numbers of filters strongly impacts the training time as the number of trainable parameters greatly increases with each filter. Hence, faster training with 100 filters is chosen as the most appropriate.

Similarly, in Figure A.6, when comparing the baseline model with embedding dimensions $b \in \{50, 100, 200\}$, the larger embedding performs best. However, the performance observed is comparable to that of the model trained with $b = 50$. Thus, for the sake of simplicity and faster model training, the embedding dimension is set to 50.

Other fixed hyperparameters include the activation function used within dense layers set to reLu which has been shown to perform well in a variety of NNs and the use of a max-pooling strategy which leads to best results with text CNNs (Zhang and Wallace, 2015). Additionally, the use of a convolutional layer followed by max-pooling is a common approach (Xi et al., 2021; Liu, Li, et al., 2021; Liu, Wang, et al., 2020), and although could it could be extended to include multiple rounds of convolutions, the structure is kept simple. Likewise, concatenation is chosen as the best fusion strategy over other methods (such as addition) as shown by (Liu, Wang, et al., 2020).

4.6.4.2 Varying parameters

Using the structure in Figure 4.2 and the fixed hyperparameters: batch size 500, 10 epochs, 100 filters, an embedding dimension of 50 and reLu hidden layer activation functions, the following hyperparameters are combined and compared. The hyperparameter combinations that produce networks yielding the lowest validation errors are then chosen as optimal.

Table 4.2 summarises the hyperparameter values compared; there exists 864 unique combinations of hyperparameters and therefore 864 different model configurations are compared. Firstly, the sizes of the user rating, wine rating and structured wine portions of the network can each be small or large. In Table 4.2, $x \rightarrow y \rightarrow z$ represents a network structure of 3 layers: x nodes densely connected to y nodes connected to z output nodes.

All of the network portions before concatenation produce an output of 15 nodes as inspired by Liu, Wang, et al. (2020) and chosen on the basis that matrix factorisation latent representations of dimension 15 captures the underlying structure of the rating data well (refer to the results of Section 5.5). Large network sizes add an additional hidden layer of 20 nodes over the small network sizes. Naturally, endless network topologies could be compared, but the combination of small and large networks provides insight into the performance of networks with thousands versus millions of trainable parameters.

Next, various learning rates are compared along with the amount of dropout to apply. Dropout is the chosen regularisation technique - while regularisation penalty terms may be included in addition, dropping a proportion of nodes at random is decided to sufficiently regularise the networks.

Table 4.2: Wine neural network hyperparameter grid search values.

Hyperparameter	Grid Search Values
User Rating and Wine Rating Hidden Layer Size	Large: $30 \rightarrow 20 \rightarrow 15$ Small: $30 \rightarrow 15$
Structured Wine Hidden Layer Size	Large: $31 \rightarrow 20 \rightarrow 15$ Small: $31 \rightarrow 15$
Learning Rate α	0.001, 0.002, 0.01
Dropout	0, 0.1, 0.2, 0.3, 0.4, 0.5
Filter Width h	3,4,5
Final Activation Function	reLu, tanh

Dropout is applied equally throughout the network. That is, after every hidden layer, the same portion of nodes are deactivated to prevent any bias from forming in the network. Again, dropout could be applied in a more fine-grain manner where each hidden layer receives a different amount of dropout, but this would drastically increase the number of network configurations to compare.

Lastly, as discussed in the "Practitioner's Guide to Text CNNs" (Zhang and Wallace, 2015), determining an optimal filter width requires grid searching and thus differing filter widths applied in the convolutional layer are compared as well as the final activation function applied. Xi et al. (2021) noted the benefits of using tanh as a final activation function, whereas reLu is often applied in other systems (Liu, Li, et al., 2021; Paradarami et al., 2017). Hence the performance of both activation functions in the penultimate layer is noted.

Each of these 864 network configurations are trained using the `R keras` package (Allaire and Chollet, 2022) for 10 epochs using both the CDF and kNN imputed rating data $\boldsymbol{R}^{CDF}, \boldsymbol{R}^{kNN}$ and both the training and validation RMSE is recorded. The best network configuration is chosen as that which does not overfit to the training data and which predicts the validation set accurately.

Using the optimal models for each dataset $\boldsymbol{R}^{CDF}$ and $\boldsymbol{R}^{kNN}$, the final validation RMSE is recorded. Producing top-Q lists for each user can only be achieved by predicting and ranking the ratings for all wines for the validation users. After which, the quality of the NN recommendations can be evaluated using the metrics outlined in Section 4.2.2.

4.7 Weighted Hybrids

β Hybrids Different wine similarity matrices are likely to be correlated; wines which are naturally very similar as a result of their attributes will likely be rated and described similarly. Therefore, these multiple similarity matrices may capture redundant information. This is explored by linearly combining the various similarity matrices $\boldsymbol{S}$ to determine an optimal model of wine similarity. This may be considered a form of a weighted hybrid which combines the results of CF and CB similarity calculations.

The most accurate measure of wine similarity denoted $\boldsymbol{S}^{\text{optim}}$ is computed as:

$$\boldsymbol{S}^{\text{optim}} = \beta_1 \boldsymbol{S}^{v} + \beta_2 \boldsymbol{S}^{v:s} + \beta_3 \boldsymbol{S}^{v:\not s} \tag{4.7.1}$$

where each β is a weight coefficient such that $\sum_{i=1}^{3} \beta_i = 1$ and $0 \leq \beta_{1,2,3} \leq 1$. Using the `R optim` function (R Core Team, 2020), the β values are determined through Nelder-Mead minimisation (Olsson and Nelson, 1975) of the RMSE calculated when $\boldsymbol{S}^{\text{optim}}$ is used to predict the validation set. Initial β values are set to 0.33 each so as to equal weight the proportion of each similarity matrix at the start of the minimisation.

Using $\boldsymbol{S}^{\text{optim}}$ determined for each rating matrices, $\boldsymbol{R}$, $\eta(\boldsymbol{R})$ and $\zeta(\boldsymbol{R})$ and both CDF and kNN protocols, the validation set can be predicted using Algorithm 4 and the final RMSE can be determined. Then, the top-Q recommendations can be computed and evaluated as with the standard IBCF method described in Section 4.3.

θ Hybrids An alternative approach to weighted hybrids involves linearly combining the predicted ratings of various approaches to lower the error of predictions. An optimal matrix of rating predictions $\hat{\boldsymbol{R}}^{\text{optim}}$ is defined as:

$$\hat{\boldsymbol{R}}^{\text{optim}} = \theta_1 \hat{\boldsymbol{R}}^{\text{IBCF}} + \theta_2 \hat{\boldsymbol{R}}^{s} + \theta_3 \hat{\boldsymbol{R}}^{\not s} \tag{4.7.2}$$

where $\hat{\boldsymbol{R}}^{\text{IBCF}}$, $\hat{\boldsymbol{R}}^{s}$ and $\hat{\boldsymbol{R}}^{\not s}$ are the predicted ratings matrices produced using IBCF, structured wine data and unstructured wine data, respectively. The predicted ratings in each $\hat{\boldsymbol{R}}$ are those of the validation set and thus they each contain both training and validation ratings. The θ parameters are computed using Nelder-Mead minimisation (with the same process as the β hybrids) of the RMSE of the predicted validation ratings in $\hat{\boldsymbol{R}}^{\text{optim}}$.

The validation RMSE of $\hat{\boldsymbol{R}}^{\text{optim}}$ can be computed for the CDF and kNN versions of $\boldsymbol{R}$, $\eta(\boldsymbol{R})$ and $\zeta(\boldsymbol{R})$, but top-Q recommendations cannot be extracted. $\hat{\boldsymbol{R}}^{\text{optim}}$ contains predicted ratings for the validation set (in addition to the training ratings) and thus cannot be used to predict and rank unrated wines as it already contains the users' relevant wines. Hence, the ranking of the top-Q wines using $\hat{\boldsymbol{R}}^{\text{optim}}$ cannot be fairly compared to other methods.

The next chapter evaluates the strengths and weaknesses of these methods applied to the wine data set (described in Chapter 3) by examining the accuracy of rating predictions and the quality of recommendations made.

Chapter 5

Results and Discussion

This chapter discusses the results yielded by the various recommender systems built using the methods detailed in Chapter 4 when applied to the training and evaluation wine datasets (refer to Section 3.2). Based on the RMSE of rating predictions made for the validation set and the quality of top-Q recommendation lists produced (as determined by the quality metrics outlined in Section 4.2.2), this investigation of system performance explores:

- the efficacy of the CDF and kNN protocols implemented to manage the cold-start problem outlined in Section 3.2,
- the capabilities and limitations of the various recommender system methodologies implemented in Chapter 4,
- the optimal hyperparameter configurations that yield the lowest validation RMSE per system,
- the influence of normalisation and standardisation on rating predictions,
- the effects on recommendation quality when using rating data and wine attributes to compute wine similarity, and
- the optimal approach to building a South African wine recommendation system.

The aim of this investigation is to develop a final recommender system such that it maximises the accuracy of rating predictions and the quality of recommendations made (refer to Section 4.2). Consequently, the results obtained from the various systems built using the IBCF, CB, matrix factorisation, NN, and hybrid methods are presented; then, by selecting and combining the strongest methods and most appropriate cold-start management protocol, the final wine recommender system is produced.

As the cold-start management protocols largely influence the data used by the systems, they are evaluated first.

5.1 Cold-Start Management Protocols

To address the complications presented by the cold-start problem in which users do not exist in both the training and evaluation sets, two protocols based on empirical CDF sampling and kNN methods are used to impute the rating data in Section 3.2. Throughout all methods applied, the best results observed were for the kNN-imputed dataset. This may be in part due to the application of user similarity to generate more realistic rating imputations with the kNN protocol, but is perhaps more likely a result of the lack of randomisation (which is used in the CDF protocol).

Recall that the CDF protocol made use of random inverse sampling of a wine's rating distribution to impute the ratings matrix. Though this is thought to capture the randomness in rating patterns between users, it is perhaps not the best approach. With a mean wine rating of 3.79 in the dataset, it is understandable that the CDF protocol would more often predict values close to that mean. Thus, from Figure 3.3, the CDF protocol imputed ratings of 2.5, 3 and 4 in quantities that did not capture the shape of the original data as well as the kNN protocol.

Hence, the less-realistic CDF imputation leads to larger errors as, though the models can make accurate predictions, they are not close to the imputed data which contains ratings that are not representative of the original data. Consequently, it is assumed that the differences in RMSE for all methods between the CDF and kNN imputed datasets are a result of the ratings imputed by the CDF protocol which do not adequately model the real-life rating patterns of the users in the dataset.

As a result, the kNN rating imputation protocol is selected as the most appropriate way to impute a secondary rating in an attempt to mitigate the cold-start problem. Henceforth, only the results of the models built using the kNN protocol are presented and results for the CDF protocol may be found in Appendix A.10.

Table 5.1 summarises the optimal hyperparameter configurations for each method applied to the kNN-imputed dataset. Table 5.2 then contains the evaluation results for those optimal systems developed in each method and is used to select the best performing systems overall. Where applicable, this includes the training and validation RMSE[1] of each system along with the quality metric results (see Section 4.2.2) measured for recommendation lists of $Q = 20$ wines[2]. As systems are built using either the unscaled, normalised or standardised ratings, to ensure the correct comparison between systems, the RMSE results presented in Table 5.2 are computed using the original wine rating scale; the minimum RMSE across all methods, and the maximum results of each quality metric are highlighted in grey[3].

[1]The training RMSE for IBCF and CB methods cannot be recorded as the training data is explicitly used in the rating prediction (Algorithm 4) - this would result in a data leak whereby an observed rating is used to predict itself. The θ hybrids are determined using validation ratings and cannot be used to produce top-Q recommendations.

[2]For all quality metrics, stronger results are expected for higher values of Q, thus the theoretically best quality results for each system are reported for a maximum of 20 recommendations.

[3]Referring to equation (4.2.5), the similar FPR values for all systems in Table 5.2 are a result of the roughly equal true negative recommendations produced. Per user at $Q = 20$, the minimum number of irrelevant wines not recommended is 1,620 (20 irrelevant wines recommended) and the maximum is 1,640 (no irrelevant wines recommended). So, as the number of false positive recommendations can range between 0 and 20 only, the FPR values computed across all relevant users will always be some multiple of the fraction: $\{0, \ldots, 20\}/\{1620, \ldots, 1640\}$, no matter the method used.

Table 5.1: Optimal hyperparameter configurations for methods applied to the kNN-imputed dataset.

Method	Input Data	Hyperparameter Configuration
IBCF	$\boldsymbol{R}^{kNN}$ and $\boldsymbol{S}^{v}$	$k = 10$, NHSM similarity measure
Structured CB	$\boldsymbol{R}^{kNN}$ and $\boldsymbol{S}^{v:s}$	$k = 20$, HAR distance measure
Unstructured CB	$\boldsymbol{R}^{kNN}$ and $\boldsymbol{S}^{v:\not{s}}$	$k = 20$, PCC similarity measure
β Hybrid	$\boldsymbol{S}^{v}, \boldsymbol{S}^{v:s}, \boldsymbol{S}^{v:\not{s}}$	$\beta_1 = 0.3651, \beta_2 = 0.4873, \beta_3 = 0.2278$
θ Hybrid	$\hat{\boldsymbol{R}}^{\text{IBCF}}, \hat{\boldsymbol{R}}^{s}, \hat{\boldsymbol{R}}^{\not{s}}$	$\theta_1 = 0.8834, \theta_2 = 0.0656, \theta_3 = 0.0425$
SGD Factorisation	$\boldsymbol{R}^{kNN}$	Dimension = 30, $L1_U = 0.1$, $L2_U = 0.1$, $\alpha = 0.1$
	$\eta(\boldsymbol{R}^{kNN})$	Dimension = 15, $L1_U = 0.05$, $\alpha = 0.1$
	$\zeta(\boldsymbol{R}^{kNN})$	Dimension = 30, $L2_U = 0.05$, $L2_V = 0.05$, $\alpha = 0.1$
ALS Factorisation	$\boldsymbol{R}^{kNN}$	Dimension = 15, $\lambda = 0.2$
	$\eta(\boldsymbol{R}^{kNN})$	Dimension = 15, $\lambda = 0.05$
	$\zeta(\boldsymbol{R}^{kNN})$	Dimension = 15, $\lambda = 0.1$
Neural Network	$\eta(\boldsymbol{R}^{kNN})$	Final activation function = tanh, User Rating, Wine Rating and Structured Wine hidden layer size = small, $\alpha = 0.002$, Dropout = 0.1, $h = 3$

Table 5.2: Evaluation metric results for the optimal systems built using various methods applied to the kNN-imputed dataset.

Method	RMSE		Measured at $Q = 20$				
	Train	Valid	COV	PER	TPR	FPR	ARHR
IBCF	-	0.5938	0.9945	0.9156	0.6908	0.0118	0.6181
Structured CB	-	0.5926	0.9970	0.9440	0.0720	0.0122	0.0319
Unstructured CB	-	0.5909	1.0000	0.9522	0.0076	0.0122	0.0019
β Hybrid	-	0.5936	0.9976	0.9452	0.0675	0.0122	0.0314
θ Hybrid	-	0.5786	-	-	-	-	-
SGD $\boldsymbol{R}^{kNN}$	0.1517	1.2371	0.8433	0.9227	0.3224	0.0120	0.0942
SGD $\eta(\boldsymbol{R}^{kNN})$	0.2012	1.0145	0.4262	0.8476	0.2572	0.0120	0.0850
SGD $\zeta(\boldsymbol{R}^{kNN})$	0.1308	0.5731	0.9311	0.8673	0.0079	0.0122	0.0029
ALS $\boldsymbol{R}^{kNN}$	0.3411	0.6026	0.5713	0.5732	0.0530	0.0122	0.0117
ALS $\eta(\boldsymbol{R}^{kNN})$	0.3302	0.5840	0.5927	0.4045	0.0320	0.0122	0.0138
ALS $\zeta(\boldsymbol{R}^{kNN})$	0.2149	0.5744	0.8756	0.5394	0.0234	0.0122	0.0056
Neural Network	0.5830	0.5850	0.0128	0.0000	0.0133	0.0122	0.0032

5.2 Item-Based Collaborative Filtering

Building an optimal IBCF system involves selecting the similarity measure and number of similar items k used in Algorithm 4 that lead to the most accurate rating predictions and highest quality recommendations. Figure 5.1a plots the validation RMSE yielded by different similarity measures for increasing values of $k \in \{1, \dots, 20\}$ when Algorithm 4 is applied to the unscaled, normalised (η) and standardised (ζ) kNN-imputed datasets. Likewise, Figures 5.1b-e plot the coverage (COV), personalisation (PER), ROC curve and average reciprocal hit rate (ARHR) for each similarity measure used with Algorithm 5 for recommended lists of increasing length $Q \in \{1, \dots, 20\}$ computed with each version of scaling applied to $\boldsymbol{R}^{kNN}$.

In Figure 5.1a, the MSD and EUC measures yield consistently low RMSE values for the unscaled and scaled datasets as they are robust to scaling (refer to Section 3.5). Apart from COS, the performance of the measures is very similar between $\boldsymbol{R}$ and $\eta(\boldsymbol{R})$, likely due to the very minor differences between the most similar wines established in Algorithm 4.

The similarity measures performed slightly differently when applied to the standardised data, however. While the worst performing measure for the unscaled and normalised data was COS, it performed the best for the standardised data in Figure 5.1a. The standardised data, unlike the other rating matrices, contained negative ratings after scaling, changing the range of the cosine similarity from [0,1] to [-1,1].

This change can be seen in Figure 5.2 which contains density plots of the COS similarity matrices produced using $\boldsymbol{R}^{kNN}$, $\eta(\boldsymbol{R}^{kNN})$ and $\zeta(\boldsymbol{R}^{kNN})$. The standardised COS similarity matrix is able to capture a more expressive range of similarities as the peak around -1 reflects totally dissimilar wines, the peak around +1 reflects theoretically identical wines and values in between reflect a broader range of similarities. This is in contrast to the unscaled and normalised COS matrices which only measure similarity in the more restricted range [0,1]. The benefits of measuring IBCF similarity using standardised ratings are evident as, for $k = 10$, the lowest validation RMSE of 0.58 using the COS measure applied to the standardised kNN data is recorded.

The positive effects of standardisation are further observed in Figures 5.1b-e, whereby the performance of all measures is stronger in the standardised datasets. Based on RMSE alone, the COS, MSD or EUC formulas seem most appropriate to measure similarity. This, however, is disproved by the generally poor quality of the top-Q recommendations generated with those measures.

The coverage of recommendations made with the COS and CPCC measures are lowest overall in Figures 5.1b. Instead, the JAC and NHSM achieve almost complete coverage of the catalogue of wines at $Q = 2$. Similarly, the COS measure produces the least personalised recommendations in Figure 5.1c and is surpassed by JAC, NHSM, PCC and WPC measures yielding the highest PER values.

The superior quality of the NHSM and JAC top-Q recommendations is reflected in Figures 5.1d-e. The TPR and ARHR of recommended lists made by all measures except NHSM and JAC are essentially 0[4]. The high TPR and ARHR values yielded by the NHSM and JAC measures can be similarly interpreted; the number of relevant items recommended is high proportional

[4] All measures except NHSM and JAC are omitted from Figure 5.1e to more clearly visualise the performance differences between them.

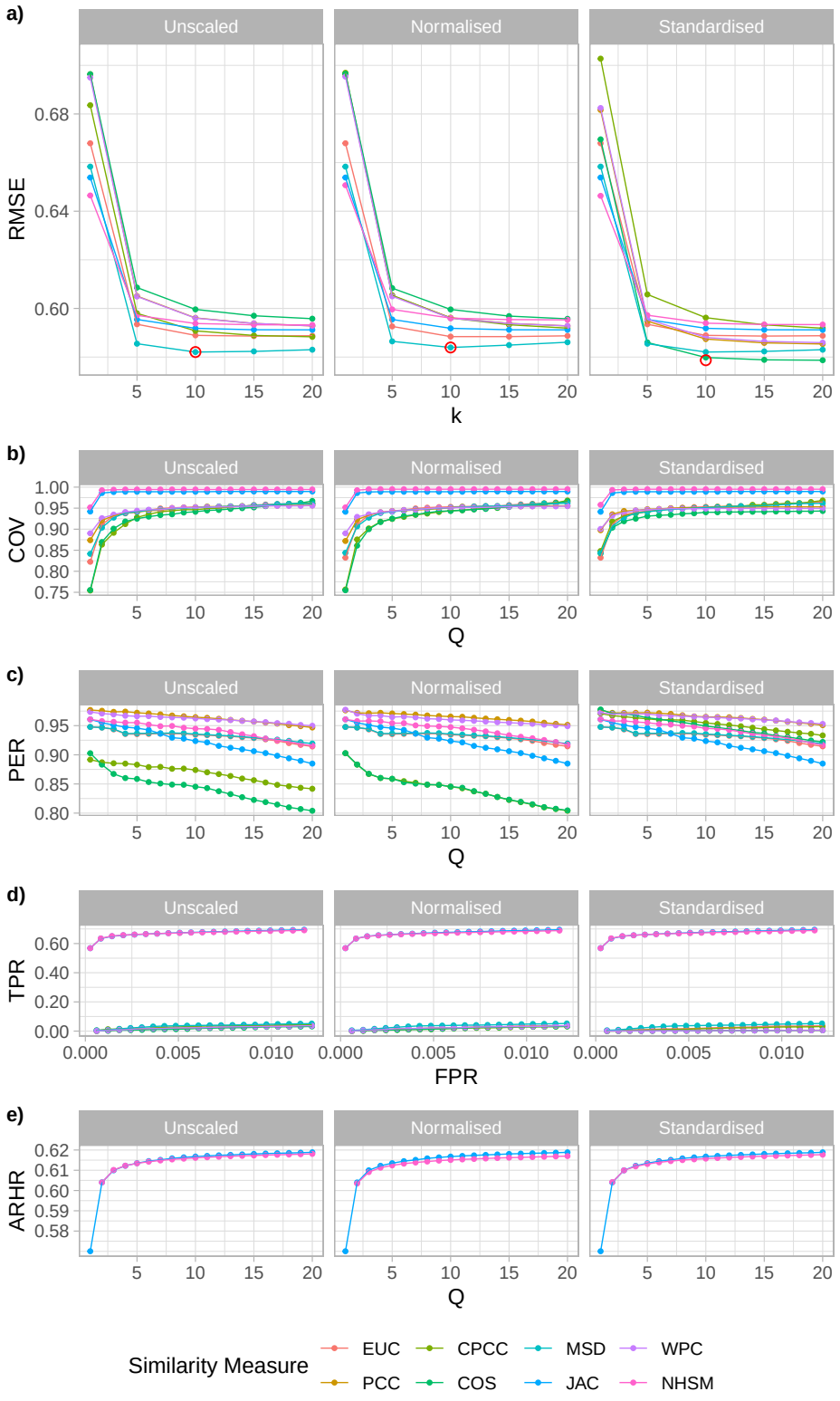

Figure 5.1: RMSE and Quality results of IBCF applied to the unscaled, normalised and standardised kNN validation sets for various similarity measures and values of the k most similar items.

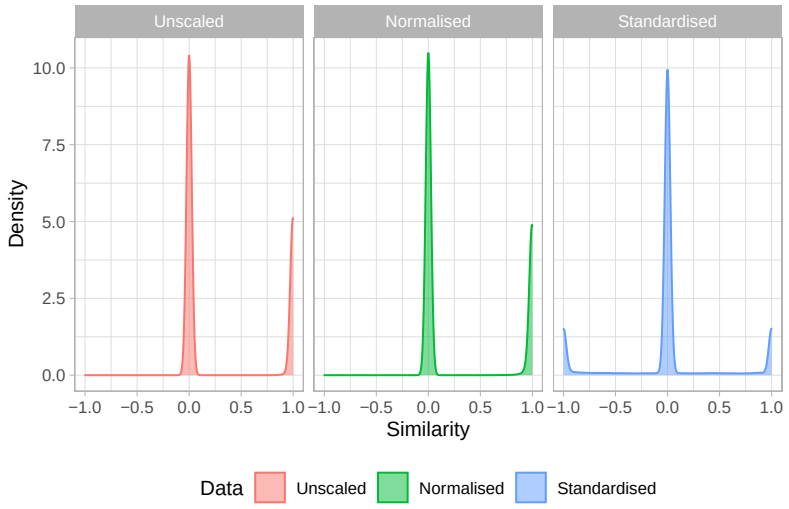

Figure 5.2: Density plots of the unscaled, normalised and standardised kNN COS similarity matrices.

to the relevant items not recommended. In other words, using the NHSM and JAC similarity matrices with Algorithm 5 leads to recommendations which appropriately recall the wines rated by users in the validation set. The higher ARHR of the NHSM measure in Figure 5.1e further reflects how the NHSM measure more appropriately ranked the recommended wines such that more relevant items are recommended at $Q = 1$ than the JAC measure for the kNN data.

The performance of NHSM and JAC are very similar (differences of only 0.01) in Figures 5.1a-e. But, as the NHSM produced more personalised recommendations in 5.1b it is chosen as the optimal measure at $k = 10$ with a validation RMSE of 0.5938. Apart from the coverage and personalisation of the standardised kNN top-Q recommendations, the unscaled data yielded marginally more accurate recommendations in Figure 5.1a and thus $\boldsymbol{R}^{kNN}$ is chosen as the optimal input for IBCF.

The high quality of the NHSM top-Q recommendations is evidence that the rigorous measure of similarity captured by Liu, Hu, et al. (2014) in equation (A.1.9) accurately modelled the relationship between wines, which in turn led to recommendations which matched the real-life rating patterns of users in the wine dataset.

These optimal kNN IBCF hyperparameter configurations are summarised in Table and 5.1 and the accompanying results produced by that system are contained in Table 5.2.

5.3 Content-Based Methods

Developing an optimal CB system requires the selection of the most appropriate content (structured or unstructured) and measures used to compare items. Figure 5.3a plots the validation RMSE for the structured distance measures (applied to the categorical and numeric wine data) and the similarity measures (applied to the unstructured wine descriptions) for increasing numbers of similar wines k in Algorithm 4 applied to the kNN dataset. Figures 5.3b-e plot the results of the quality metrics yielded by each distance measure and similarity measure applied

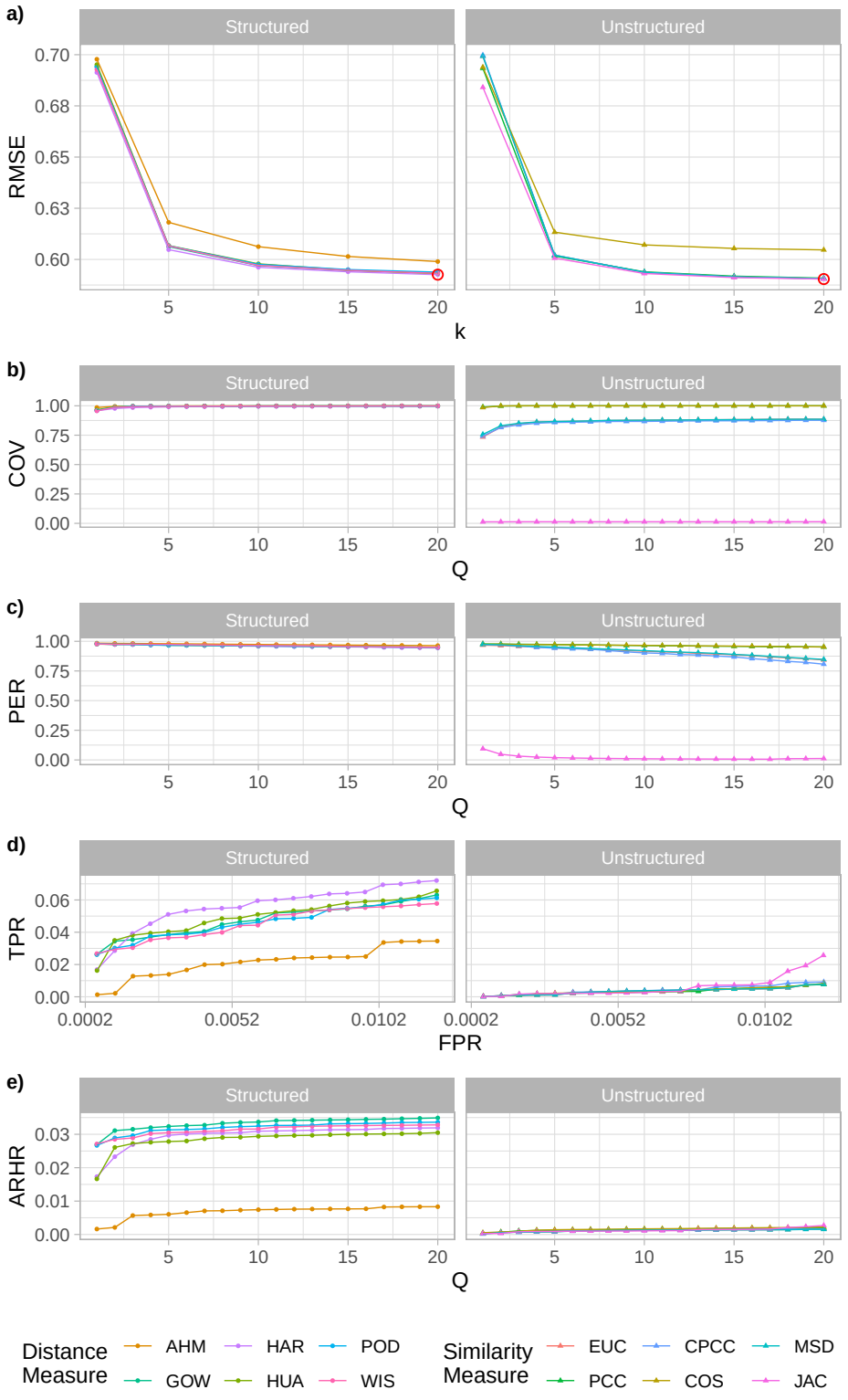

Figure 5.3: RMSE and Quality results of structured and unstructured CB methods applied to the kNN data for various similarity and distance measures and values of the k most similar items.

to the structured and unstructured wine data for increasing numbers of recommendations Q when applied to $\boldsymbol{R}^{kNN}$.

The validation RMSE produced using both structured and unstructured CB methods are similar to those of the IBCF measures with values around 0.59 in Figure 5.3a. All similarity formulas produced equally accurate validation rating predictions except for the AHM measure and COS measure.

While the COS measure was least accurate for the unstructured CB approach, it, along with the PCC measure, achieved the highest coverage in their top-Q recommendations in Figure 5.3b and the greatest personalisation in 5.3c.

The JAC measure, unlike in the IBCF approach produced the worst COV and PER results in Figures 5.3b and 5.3c[5].

Conversely, all distance measures covered almost all wines in the catalogue for lists of length $Q \geq 2$ in Figure 5.3b and produced equally personalised recommendations in Figure 5.3c. The consistently unique structured CB recommendations for all values of Q are in contrast to the IBCF approach in which the personalisation of recommendations decreases as Q increases in Figure 5.1b.

Next, the ROC curves and ARHR results for both the structured and unstructured CB approaches are far lower than the IBCF methods (Figure 5.1) for all similarity formulas in Figure 5.3e. Regardless, these figures can be used to select the optimal distance measure. As the HAR measure generates the highest TPR (with values more than 0.02 above other measures) and performs equally to the other measures in Figures 5.3a-c it is chosen as the optimal structured CB distance measure

Then, for the unstructured CB similarity measures, although the JAC measure yielded the better TPR and ARHR results in Figures 5.3d and 5.3e, this is likely just due to chance as the same recommendations are repeated for all users; hence, the JAC measure is inappropriate. All other measures perform similarly in Figures 5.3d and 5.3e but the PCC measure predicts recommendations with a lower RMSE in Figure 5.3a. Therefore, the PCC formula which measures the correlation between the TF-IDF vectors describing two wines is chosen as the optimal unstructured CB similarity measure.

It is worth noting that this application of CB methodologies is limited by the lack of user data available. More traditionally, the CB approach pairs items with users through their attributes and though exploiting wine content in this system is feasible, with no more than user rating history data available, the full potential of CB systems cannot be reached. Though more explicit user data (such as gender, age or purchase history) would allow the CB systems to model user preferences with more detail, the use of CB methods has other useful applications in this context. Specifically, refer to Section 5.7.2 which outlines how wine attributes and CB similarities are used to explain recommendations.

[5]As the JAC formula applied to the TF-IDF vectors $\boldsymbol{d}$ describing each wine only considers the proportions of intersecting non-negative weights and not the magnitudes of each ω_{ϱ}, it is apparent that the information captured using TF-IDF is lost. Because each TF-IDF vector consists of mainly non-negative weights, each wine would be considered equally similar to one another. Hence, despite user preferences, they are each recommended the same set of wines (evident in the low personalisation of Figure 5.3c) and thus a small subset of wines in the entire catalogue are recommended as seen in the low coverage in Figure 5.3b.

Furthermore, from Table 5.2, the respective minimum RMSE values of 0.5926 and 0.5909 are observed for $k = 20$ the structured and unstructured approaches. Although only slightly, the unstructured approach led to RMSE values less than both the structured CB and IBCF approaches, as well as greater coverage and personalisation. It may appear then, that the unstructured CB approach would be the more favourable system that makes use of wine similarity. However, when examining the ARHR values, it is clear that the unstructured approach produced a far smaller proportion of well-ranked wine recommendations than the structured and IBCF approaches.

Comparing Wine Similarity

The IBCF, structured and unstructured CB similarity matrices ($\boldsymbol{S}^{v}$, $\boldsymbol{S}^{v:s}$ and $\boldsymbol{S}^{v:\not s}$) are all developed using individual sets of data which capture different elements of the same set of wines. On the one hand, the three $\boldsymbol{S}$ matrices should be expected to be completely different as they are built independently of one another and should yield completely different recommendations. On the other hand, each $\boldsymbol{S}$ matrix measures the similarity between the same set of wines and if two wines are almost indistinguishable, then the similarity between them should be roughly equivalent in $\boldsymbol{S}^{v}$, $\boldsymbol{S}^{v:s}$ and $\boldsymbol{S}^{v:\not s}$.

When examining the RMSE values produced by each method that makes use of wine similarity in the third column of Table 5.2, it seems evident that no matter how similarity is measured, the intrinsic relationships between the wines are captured well with either the IBCF or CB systems as the results are essentially equal. Yet, when comparing the quality of top-Q recommendations made in the final three columns of Table 5.2, the performance of IBCF far surpasses that of CB systems.

This conflict between the apparent equal accuracy of IBCF rating predictions and their varying recommendation quality can be explained by the intricacies of Algorithms 4 and 5. In Algorithm 4 used by both IBCF and CB systems, the predicted rating is a weighted combination of the ratings made by a user and the similarity of those previously rated wines to a given other wine. Thus, the predicted ratings made for a user will always resemble an average of the users' previous ratings.

Figure 5.4 plots the density of the standard deviation of the users' ratings and illustrates that the majority of user rating standard deviations are under 0.5. In other words, most users do not award ratings more than 0.5 away from their average rating. This indicates that, for most users, the predicted rating need not stray far from the user's mean rating to achieve a low RMSE. This low rating standard deviation reveals how Algorithm 4 can be influenced to produce more or less accurate predictions.

Consider the set of ratings $\boldsymbol{r}_i = \{2.5, 3, 3.5\}$ made by user i with a mean rating of 3 and a standard deviation of 0.5 and the scaled IBCF similarity vectors $\dot{\boldsymbol{s}}_1 = \{0.05, 0.15, 0.8\}$ and $\dot{\boldsymbol{s}}_2 = \{0.35, 0.3, 0.35\}$. The vector $\dot{\boldsymbol{s}}_1$ can be considered more *expressive* than $\dot{\boldsymbol{s}}_2$ as it contains values which are largely dissimilar from each other - $\dot{\boldsymbol{s}}_1$ expresses that one wine is far more similar than the others. Whereas, $\dot{\boldsymbol{s}}_2$ provides less expressive detail as all wines are seemingly equally similar.

The predicted ratings $\hat{r}_{i1} = 3.375$ and $\hat{r}_{i2} = 3$ illustrate how similarity vectors with high

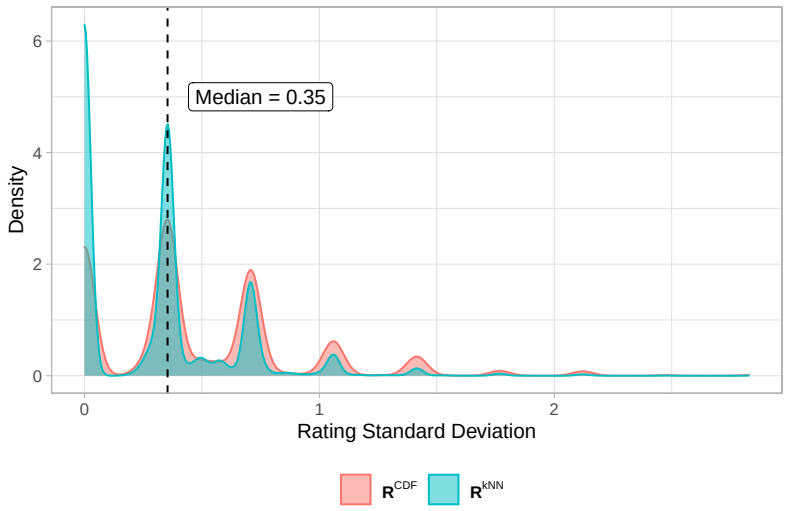

Figure 5.4: Density plot of the standard deviation of user ratings in both the CDF and kNN datasets.

expressiveness ($\dot{\boldsymbol{s}}_1$) bias the predicted rating more strongly towards the ratings awarded to the most similar wines. Similarity vectors with low expressiveness ($\dot{\boldsymbol{s}}_2$), however, produce predictions closer to the mean rating. This highlights a certain sensitivity of Algorithm 4 - more expressive similarity vectors (i.e. those with a diverse range of similarity values) produce predictions that stray from the user's mean whilst less expressive vectors (such as $\dot{\boldsymbol{s}}_2$) limit predictions to a general user average. This is advantageous when the similarity captured is accurate as expressive vectors will produce more accurate results, and disadvantageous when similarity is measured inaccurately as this negatively impacts predictions.

Though Algorithm 4 is sensitive to the similarity matrix it is supplied, the predicted rating is more dependent on the rating vector $\boldsymbol{r}_i$. As $\boldsymbol{r}_i$ consists of one or more values between 1 and 5 and $\dot{\boldsymbol{s}}_j$ consists of values ≤ 1, the magnitude of the similarity vector is always less than the rating vector: $||\dot{\boldsymbol{s}}_j|| \leq 1 \leq ||\boldsymbol{r}_i||$. Consequently, because $\boldsymbol{r}_i$ is constant in all systems, the ratings predicted with Algorithm 4 will always strongly resemble $\boldsymbol{r}_i$ no matter the type of similarity vector used ($\boldsymbol{s}^{v}$, $\boldsymbol{s}^{v:s}$ or $\boldsymbol{s}^{v:\not s}$).

This explains why the RMSE values for the IBCF and CB methods are almost equal in Table 5.2. When examining $\hat{r}_{i1}$ and $\hat{r}_{i2}$ above, though their computation involved completely different similarity vectors $\dot{\boldsymbol{s}}_1$ and $\dot{\boldsymbol{s}}_2$, the final predicted ratings are very similar. Hence, despite any inaccuracies within a given similarity matrix, the use of Algorithm 4 will lead to somewhat favourable RMSE results.

The same is not true for the quality of recommendations and the use of the top-Q item recommendation Algorithm 5; this process is dependent on similarity alone and thus the accuracy and expressiveness of the $\boldsymbol{S}$ matrix are paramount. Figure 5.5 plots the upper triangles of the optimal similarity matrices (with close-ups of the centres) determined in Sections 5.2 and 5.3 where darker tiles represent higher similarity values.

Figure 5.5a demonstrates the expressiveness of the IBCF similarity matrices; the sparse dark tiles amongst mainly light tiles indicate that the majority of wines are dissimilar to each other,

but certain wines are highly similar. The similarity vectors extracted from the matrices in Figure 5.5a will then be highly expressive as they will likely contain a few high similar values and many low values (as with $\dot{\boldsymbol{s}}_1$ above). This expressiveness will result in both rating predictions which differ from the user mean rating and, if the similarity measure is accurate, top-Q lists which contain relevant wines.

The IBCF similarity matrices are superior when compared to the structured CB similarity matrices in Figure 5.5b which suffers from numerous high similarity values throughout the matrix. The structured CB matrix lacks expressiveness as it considers many wines as similar; if all wines are strongly related, then the similarity values within the matrices are somewhat arbitrary. As with $\dot{\boldsymbol{s}}_2$ above, rating predictions made using similarity vectors consisting of roughly equal values leads to predictions close to the user mean rating. The more serious effect, however, is that the quality of top-Q recommendations decreases as recommendations may seem random. If many wines have tied high similarities, any true meaningful discrimination between wines is not possible and the top-Q are selected as a random subset, unlikely to capture many relevant items.

This explains why the HAR measure produced decent RMSE values but low-quality recommendations; the HAR matrix predicted ratings close to the user mean (which is often adequate), but the lack of expressiveness generated recommendation lists containing wines which are not of real interest to the users.

It is not for lack of expressiveness, however, that caused the unstructured PCC measure to produce such poor top-Q recommendations in Table 5.2. In Figure 5.5c, the PCC measure produces an expressive similarity matrix as with the IBCF matrices in Figure 5.5a, but it fails as its measure of similarity is inaccurate. This can be seen in Figures 5.5a-b where bands of colours reflect an underlying wine similarity structure absent from the uniform distribution of similarity values in Figure 5.5c.

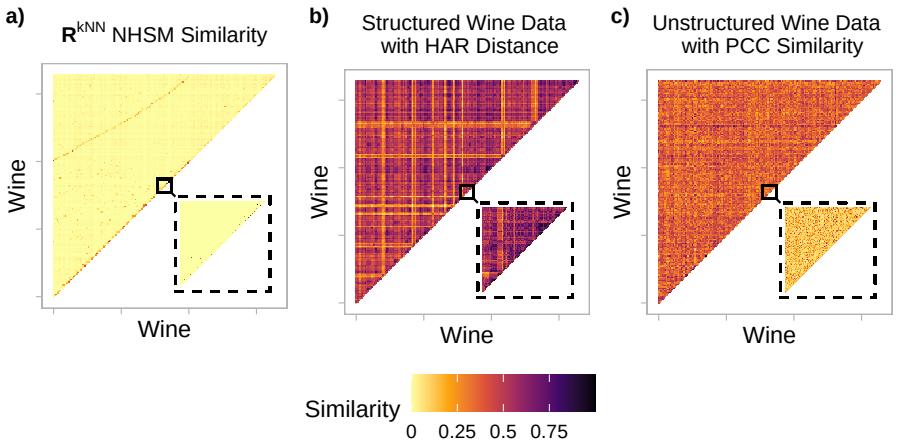

Figure 5.5: Heatmaps of the optimal IBCF, structured and unstructured CB similarity matrices.

Consider Table 5.3 which contains details pertaining to the top-3 most similar wines to an input white wine as determined by the optimal IBCF ($\boldsymbol{S}^{v}$), structured ($\boldsymbol{S}^{v:s}$) and unstructured ($\boldsymbol{S}^{v:\not s}$) CB similarity matrices. Theoretically, the most similar wine to the input wine should be a dry white wine priced around R100.

The unstructured CB approach ($\boldsymbol{S}^{v:\not s}$) determined the most similar wine to be a light-bodied red blend, followed by an expensive, off-dry Chardonnay and a pricey Sauvignon Blanc, the least truly similar top-3 overall. When examining portions of the wine descriptions, it is understandable how TF-IDF methods determined those wines to be most similar to the input - all the wines are described as fresh and fruity. Yet, supported by the low TPR and ARHR results in Table 5.2, TF-IDF applied to wine descriptions alone (perhaps as a result of a lack of detail), is insufficient in producing high-quality recommendations in practice.

The structured ($\boldsymbol{S}^{v:s}$) CB top-3 most similar wines in Table 5.3 seem far more logical when compared to the unstructured approach. All wines are full-bodied, 2020 Sauvignon Blancs in a similar price range with similar amounts of residual sugar to the input wine. And though the structured approach does not consider the wine descriptions, the top-3 descriptions match the input wine with terms such as "tropical", "green" and "fruit". This indicates the presence of an underlying correlation between the structured variables and their corresponding descriptions.

While the structured CB approach is an accurate approach to determining wine similarity, it is inappropriate for generating recommendations in a wine context. Just because a user enjoyed a dry, 2020 Sauvignon Blanc does not mean that that is all the user will be interested in. Wine is more nuanced and so other wines of real interest may be of different types (a user who drinks white wines may be interested in reds too), consisting of different grape varietals, vintages and regions. This highlights the lack of serendipity of CB systems discussed in Section 2.1.5; the use of wine content alone leads to obvious recommendations which are not meaningful for a user, reflected in the low TPR and ARHR values in Table 5.2.

Conversely, the top-3 wines recommended with the IBCF approach ($\boldsymbol{S}^{v}$) in Table 5.3 demonstrate how collaborative filtering is able to produce the highest quality recommendations in Table 5.2. At first glance, the top-3 recommendations of a Chardonnay, an expensive and a cheap Merlot are surprising and do not seem to be similar to the input wine. Though the wines may not appear to be similar in terms of their attributes, they are similar according to user preferences as a result of the same grouping of users rating them similarly. This is a benefit of the IBCF approach; the wines suggested are unexpected but are of actual interest to a user. The high TPR and ARHR values of the IBCF approaches in Table 5.2 show how determining wine similarity through user ratings produces recommendations which are truly relevant to a user.

Therefore, IBCF is the most accurate way to capture the nuanced relationships between users and wine in such a way that the most appropriate recommendations are made. Though structured CB provides more explicit wine similarities, it is not best suited to providing meaningful recommendations. Furthermore, while the unstructured approach to CB captured somewhat meaningful recommendations based on wine descriptions, due to either a lack of detailed descriptions or the limits of TF-IDF itself (see Table 2.1), the recommendations produced with $\boldsymbol{S}^{v:\not s}$ are far less relevant.

Table 5.3: Details of the top 3 wines most similar to an input white wine, determined by various approaches to wine similarity.

	Price	RS	Year	Grape	Type	Body	Description
Input Wine	107	3.5	2020	sauvignon blanc	White	Full	Tropical fruit followed by intense passion fruit aromas well supported by hints of green pepper...
$\boldsymbol{S}^{v}$ NHSM	77	4.87	2018	chardonnay	White	Med	Citrus and apricot aromas with hints of spice opening up into floral and tropical notes...
	292	4.11	2019	merlot	Red	Full	Flavours of cocoa and black cherry, with creamy texture and a hint of exotic masala spice and peppercorn...
	75	3.6	2019	merlot	Red	Full	Deep purple coloured wine, with intense flavours of dark berry fruit, spice and plum notes on the nose...
$\boldsymbol{S}^{v:s}$ HAR	98	2.9	2020	sauvignon blanc	White	Full	Tropical fruit with granadilla and white peach. Zesty and flinty on the palate...
	140	2.6	2020	sauvignon blanc	White	Full	Gooseberries, kiwi and a strong herbal interes with mineral structure and greener notes...
	118	2.88	2020	sauvignon blanc	White	Full	Aromas of grapefruit and guava on the nose followed by mouth watering acidity and upfront citrus fruit on the palate...
$\boldsymbol{S}^{v:\$}$ PCC	89	3.2	2017	blend	Red	Light	Fresh red berries with sweet strawberry aromas and fynbos notes, enhanced by subtle oak spice...
	567	5.4	2018	chardonnay	White	Full	Abundance of citrus flavours on the nose, a full palate and lingering aftertaste...
	275	1.8	2020	sauvignon blanc	White	Med	Vibrant yet complex with ripe layers of summer fruits...

5.4 Weighted Hybrids

Table 5.1 summarises the β and θ weights determined to optimally linearly combine the top similarity matrices and predicted ratings matrices. The optimal similarity matrices are chosen as those which performed the best overall in each IBCF, structured and unstructured CB method (NHSM, HAR, PCC) as discussed in Sections 5.2 and 5.3. The optimal predicted matrices $\hat{\boldsymbol{R}}$, however, are chosen as those which produced the lowest validation RMSE.

The COS measure applied to the standardised data resulted in the lowest IBCF RMSE, the GOW measure yielded the lowest structured CB error and the MSD measure yielded the lowest unstructured error. Thus, using those similarity matrices to compute the predicted matrices $\hat{\boldsymbol{R}}$, the θ hybrid is formed as:

$$k\text{NN: } \hat{\boldsymbol{R}}^{\text{optim}} = 0.8834\hat{\boldsymbol{R}}^{\text{IBCF}} + 0.0656\hat{\boldsymbol{R}}^{s} + 0.0425\hat{\boldsymbol{R}}^{\not s} \tag{5.4.1}$$

In the θ hybrid, the majority of the optimal rating prediction comes from the IBCF method, followed by much lower percentages of the structured and unstructured CB predictions. This indicates that the greater expressiveness and accuracy of the IBCF method leads to stronger recommendations overall as the optimal rating prediction is drawn mainly from the IBCF method.

The θ hybrid produced lower RMSE values than their constituent methods, but only very slightly. Consequently, the use of such hybrids is not appropriate as the improvement in RMSE is not worth the computation required to generate each IBCF and CB component.

From Table 5.2, the β hybrid also produces lower validation errors than their constituent methods. The NHSM measures produced optimal similarity for the kNN dataset (see Section 5.2) and the optimal structured and unstructured CB similarity matrices are derived using the HAR measure and PCC measure respectively (see Section 5.3). Using those matrices, the optimal similarity matrices are computed using the optimised weights as:

$$k\text{NN: } \boldsymbol{S}^{\text{optim}} = 0.3651\boldsymbol{S}^{v} + 0.4873\boldsymbol{S}^{v:s} + 0.2278\boldsymbol{S}^{v:\not s} \tag{5.4.2}$$

The optimisation of the kNN similarity matrices led to comparable proportions each of the similarity matrices contributing to $\boldsymbol{S}^{\text{optim}}$. Though this had a positive effect on the kNN RMSE in Table 5.2, the influence of the CB similarity matrices drastically diminished the quality of the recommendations produced by the β hybrid.

From Table 5.2, though the TPR and ARHR of the CB methods are far less than the IBCF method, they do achieve higher coverage and personalisation results overall. Hence, when incorporating the CB similarity matrices in equation (5.4.2), the coverage and personalisation of the β hybrid increases over the IBCF method.

Regardless, the performance increases of all hybrids are negligible. If the time and resources are available to linearly incorporate the CB with the IBCF similarity approaches, it should not be discouraged. But, the superior performance of the top-Q IBCF recommendations over its CB counterparts is evidence enough that IBCF is a sufficient standalone method.

5.5 Matrix Factorisation

Latent factor representations of $\boldsymbol{R}$, $\eta(\boldsymbol{R})$ and $\zeta(\boldsymbol{R})$ are computed using both SGD and ALS matrix factorisation techniques and are used to predict the ratings in the validation set. For each method, the optimal factor dimension, amount of regularisation and type of scaling should be determined. Figure 5.6 plots the cross-validation RMSE and recommendation quality metric results for SGD matrix factorisation (Algorithm 1) applied to the unscaled, normalised and standardised kNN ratings matrices. Figure 5.7 then plots the same set of results for ALS factorisation (Algorithm 2) applied to unscaled, normalised and standardised matrices $\boldsymbol{R}^{kNN}$.

Tables 5.1 and 5.2 contain the optimal hyperparameter configurations for the SGD and ALS algorithms and the results of those optimal models applied to the kNN data.

5.5.1 SGD factorisation

In Figure 5.6a, each point corresponds to the cross-validation RMSE of SGD matrix decomposition trained on the kNN-imputed (unscaled, normalised and standardised) matrices for increasing numbers of dimensions and amount of total regularisation. As the `recosystem` package allows for fine-grain controls over regularisation, L1 and L2 penalties can be added to both the user and wine factors. Instead of plotting the results of each type of regularisation separately, the total amount of regularisation is indicated by the colour of the points. Similarly, the default learning rate of 0.1 is found to be optimal and thus the grid search results of only $\alpha = 0.1$, number of dimensions and total regularisation are plot.

The RMSE between the plots in Figure 5.6a cannot be directly compared as each decomposition was conducted on data of different scales; the RMSE of each method after transforming the recomposed matrices back to the original rating scale are found in Table 5.2.

From Figure 5.6a, as the number of dimensions is increased, the variation in the RMSE of the predictions produced increases for the unscaled data. For the normalised and standardised decompositions, the RMSE reaches a maximum ceiling no matter the number of dimensions used. In general, larger amounts of regularisation lead to greater RMSE values as seen by the higher light-coloured points in 5.6a.

The lowest RMSE in each model (indicated by the red cross in each plot) is yielded by the combinations of grid search parameters summarised in Table 5.1 where L1_U, L2_U, L1_V, L2_V specifies the type of regularisation applied to either the user or wine latent matrix $\boldsymbol{U}$ or $\boldsymbol{V}$. Table 5.2 (that contains the RMSE of the recomposed matrices transformed to the original scale) reveals that the matrices recomposed with the optimal hyperparameter configurations in Table 5.1 all predict the training ratings extremely accurately with errors of less than 0.25. The lack of regularisation applied to the wine latent matrices in the unscaled and normalised models likely caused the overfitting of the training data as evident by the corresponding very high validation RMSE values > 1 observed in Table 5.2. Contrarily, the regularisation of the standardised user and wine latent matrices lead to a well-generalised model which predicted the validation set with the lowest error of 0.5731.

The excellent results of the standardised latent SGD matrices mirror the findings of Bell et al. (2009) who demonstrate the power of matrix factorisation. Additionally, the use of standardisation to both remove biases in each user's rating scales and ensure a stable gradient descent

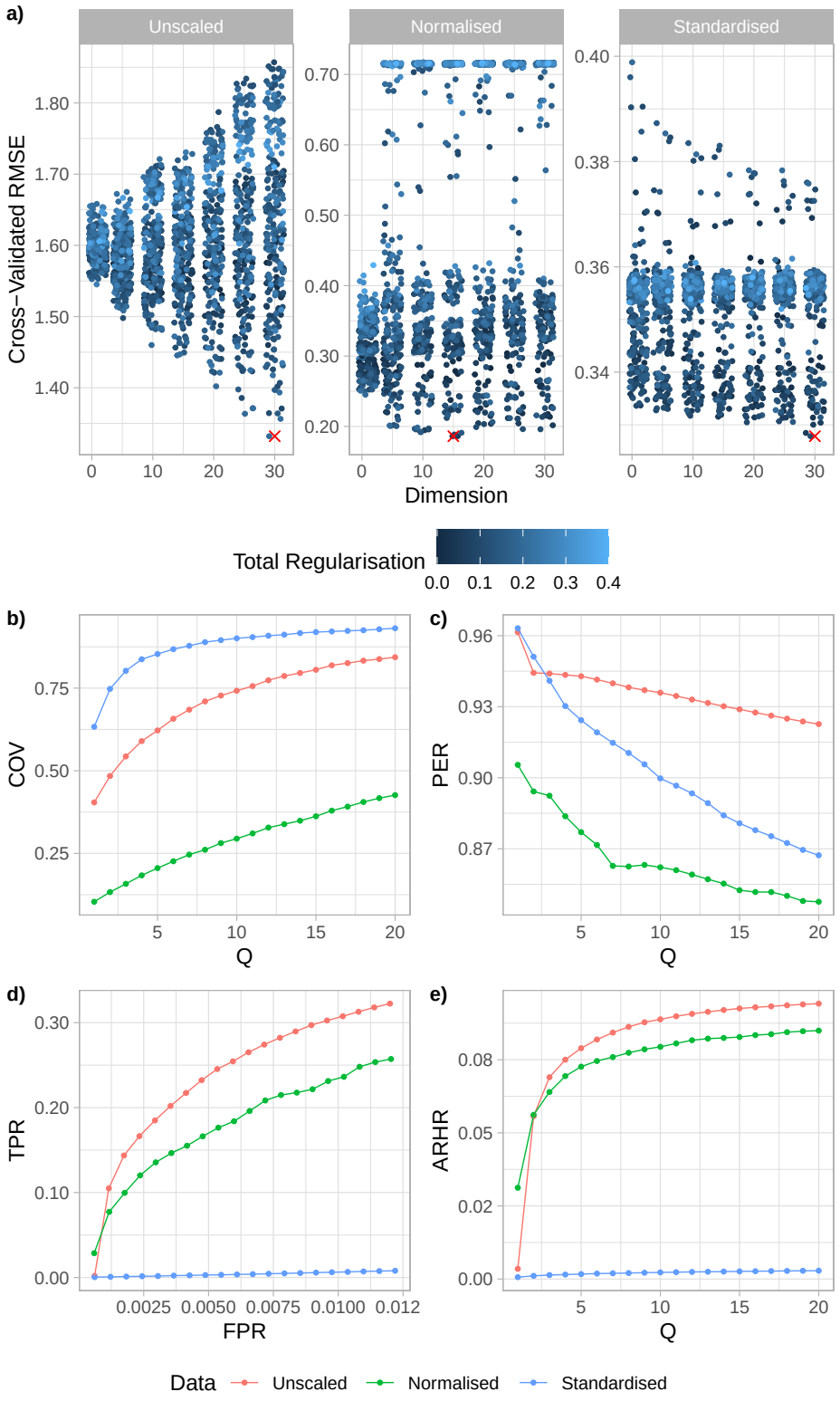

Figure 5.6: Cross-validation RMSE and Quality results of SGD matrix decomposition applied to the unscaled, normalised and standardised kNN training sets for various numbers of dimensions and amount of total regularisation.

(as discussed in Section 3.5) is clearly a critical pre-processing step as the standardised model outperformed the normalised and unscaled models for both the CDF and kNN data.

The top-Q recommendation quality metrics are more disappointing in general for the SGD approach. Compared to the IBCF and CB approaches, the SGD models produce recommendations which are less personal and cover less of the wine catalogue in Figures 5.6b and 5.6c. The results are worse still in Figures 5.6d and 5.6e as the number of relevant items recommended are far less than the IBCF method with ARHR values < 0.1.

Despite this, in all but Figure 5.6b the standardised SGD model performs sub-optimally and the unscaled model performs better than normalised model in all Figures 5.6b-e. Although the unscaled SGD latent matrices seem to produce the best top-Q recommended lists, it is not chosen as the optimal approach as the standardised SGD model yielded the lowest RMSE values overall.

The results of Figure 5.6 indicate that matrix factorisation is a superior method to predict user ratings but should not be used to generate lists of recommendations.

5.5.2 ALS factorisation

Figure 5.7a plots the RMSE results of the ALS matrix decomposition after transforming each recomposed matrix to the original rating scale for the kNN dataset. The RMSE of the predictions obtained when recomposing the training set are plotted as solid lines and the RMSE of the validation set predictions are plotted as dashed lines. As `sparklyr` applies a single value of regularisation to both the $\boldsymbol{U}$ and $\boldsymbol{V}$ matrices, the total amount of regularisation is plot as separate colours.

The results of the SGD decomposition are mirrored in Figure 5.7a as high values of regularisation produce the highest validation RMSE and low amounts of regularisation result in low RMSE for the training set. The results of the training and validation RMSE diverge for the unscaled and normalised data; as the results of the hyperparameter combinations improve for the training data, they worsen for the validation. This is not the case, however, for the standardised data, which yields an almost constant validation RMSE over all hyperparameter combinations.

The optimal latent factor dimension which yields the lowest RMSE for the training and validation data is indicated by the red circles in Figure 5.7a. Apart from the standardised data, the optimal dimension of the ALS latent models is 30 for the training data and 1 according to the validation data; this indicates that higher dimensions of ALS latent matrices leads to overfitting.

In the unscaled and normalised plots of Figure 5.7a, the low-dimension latent matrices that yield the smallest validation RMSE produce the highest training error and the high-dimension models which produce the lowest training errors perform poorly on the validation data. Thus, for all models, the optimal dimension is chosen as 15 as beyond that both validation and training error somewhat stabilise.

The optimal amount of regularisation RMSE applied to each ALS latent factor which yielded the lowest validation error is summarised in Table 5.1 and the results of the corresponding top-Q recommendations produced that model are found in Table 5.2. The lowest RMSE among ALS models is for the standardised data in the validation set with a value of 0.5744. The accuracy

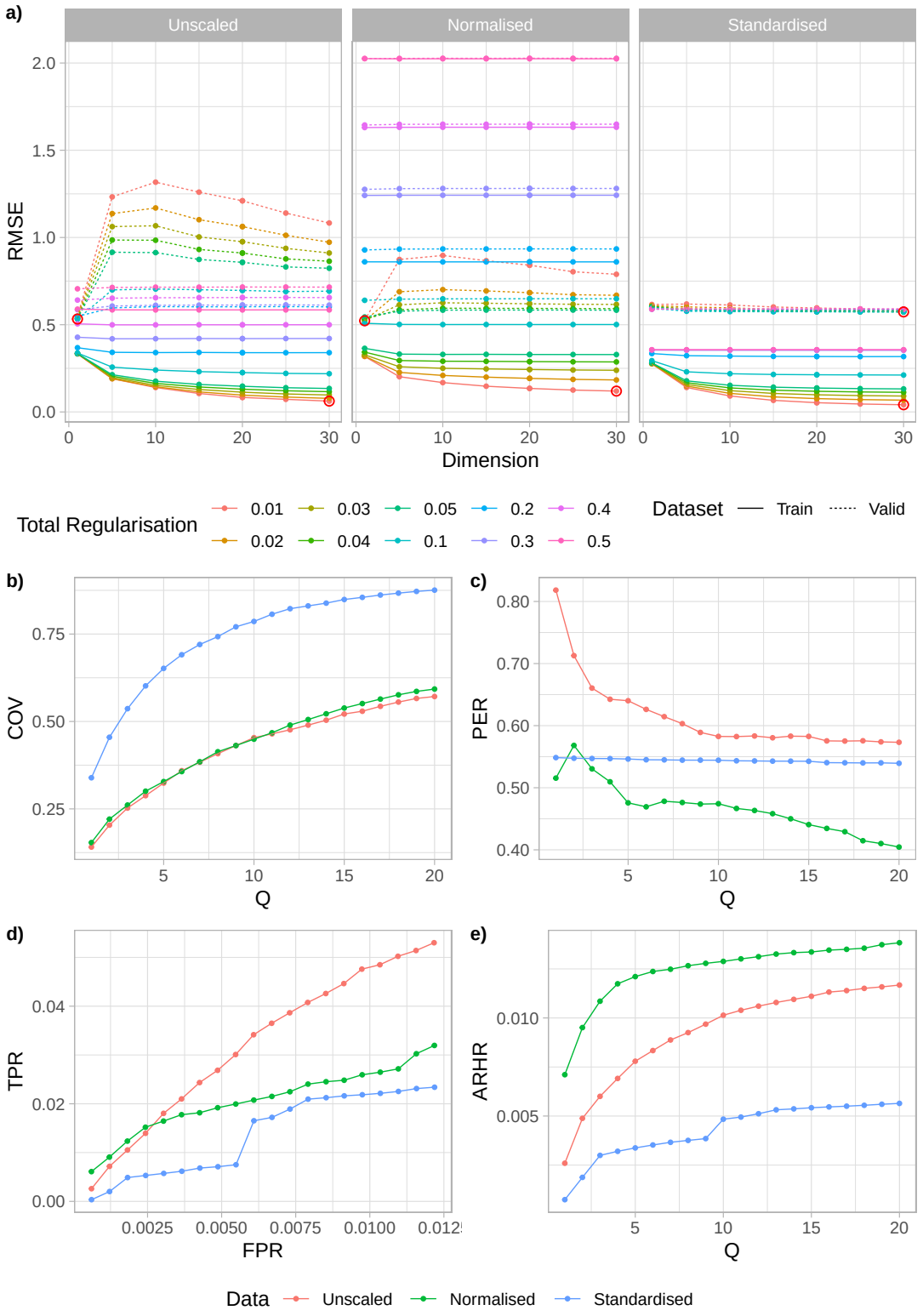

Figure 5.7: RMSE and Quality results of ALS matrix decomposition applied to the unscaled, normalised and standardised kNN training and validation sets for various numbers of dimensions and amount of total regularisation.

of these ALS rating predictions is less than the SGD decomposition results and more closely resembles those of the IBCF and CB results.

Coverage of the wine catalogue and personalisation of the top-Q lists in Figure 5.7b-c are not terrible, but they pale in comparison to the recommendations produced using wine similarity. And as with the SGD approach, worse results are observed for the ROC curve in Figure 5.7d and the ARHR plots in Figure 5.7e.

Apart from the coverage plot in Figure 5.7b, the unscaled model produces recommendations which yield the best results. It is worth nothing that the ALS approach is clearly more robust to scaling when compared to SGD as the training and validation RMSE values are very similar, despite the type of scaling applied. Despite this, the lowest validation RMSE is observed for the standardised kNN model and is chosen as optimal.

Both the ALS and SGD models produced recommendation lists containing wines of little real relevancy to the users. This may be a result of biases in the latent matrices whereby the most popular wines (which have higher ratings overall) have latent representations of greater magnitude; hence, when recomposing the matrix, the most popular wines would receive the highest predicted ratings. Consequently, users are simply recommended popular wines, not those which are truly relevant to them and thus the quality of the corresponding top-Q recommendations suffers.

This issue may be remedied by incorporating bias terms that further regularise the latent representations to prevent popular items from dominating the latent representations. Regardless, the high accuracy of the standardised SGD latent representations is enough evidence that matrix decomposition should be used for rating predictions and not recommendation lists.

5.6 Neural Networks

As discussed in Section 4.6.1, the user and wine rating input to the neural networks is chosen as the optimal latent matrices in Section 5.5 and corresponds to the standardised SGD latent matrices which produced the lowest validation RMSE (see Table 5.1 for hyperparameter details).

Figures 5.8 and 5.9 plot the RMSE results of the numerous neural networks built using the exhaustive grid search of hyperparameter combinations in Table 4.2 and the kNN rating data.

Figure 5.8a contains boxplots of the validation error produced by models of different sizes and final activation functions[6]. The network sizes are abbreviated as abc, where a and b refer to the hidden layer sizes of the user and wine rating component, respectively and c refers to the size of the structured wine hidden layers in Figure 4.2. For example, the abbreviation *ssl* indicates that both the user and wine rating hidden layer sizes are small and the structured wine hidden layer size is large (see Table 4.2 for details).

From Figure 5.8a, the RMSE values produced using the tanh final activation function are much more stable with far fewer validation RMSE outliers and a shorter spread of RMSE values. This supports the findings of Liu, Li, et al. (2021) who report that the non-linearity of the tanh function accurately captures the interaction between users and items. Hence, tanh is chosen as the optimal final activation function.

[6]The final activation function refers to the penultimate node in Figure 4.2.

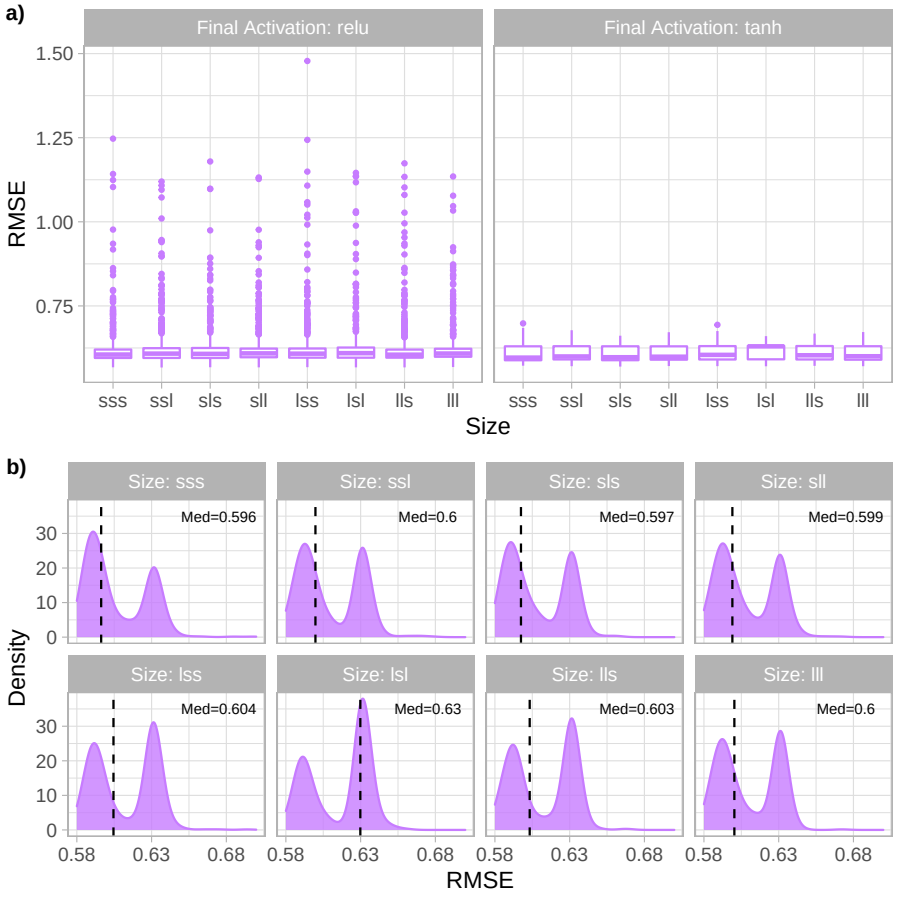

Figure 5.8: Boxplots and density plots of the validation RMSE values yielded by NN models of different final activation functions and sizes (where s and l refer to small and large hidden layer sizes respectively) applied to the kNN rating data.

Figure 5.8b plots the densities of the validation errors for models of different sizes and the tanh final activation function where the overall medians are indicated by the dashed lines. Though the shapes of the densities of each size configuration are rather similar, higher validation peaks are observed for lower RMSE values for the smaller models (such as *ssl* and *sls*) than for the larger models (such as *lll*). Further, as the *sss* model stood apart from the other size configurations with the greatest concentration of validation RMSE values below the median RMSE, it is chosen as optimal.

Next, Figure 5.9a plots the validation RMSE for *sss* models with the tanh final activation function for various learning rates, amount of dropout, epochs and filter widths. From Figure 5.9a, the highly regularised networks with dropout values of 0.3, 0.4, and 0.5 perform the least favourably, with high RMSE values on the first epoch that do not decrease steadily. While networks with dropout values of 0 may seem to produce slightly lower validation RMSE values, such networks with such little regularisation are expected to be poorly generalisable and likely overfit the training data. So, when examining the low RMSE values of the networks regularised with a dropout of 0.1 that are stable from the first epoch in Figure 5.9a, dropout of 0.1 is chosen optimal.

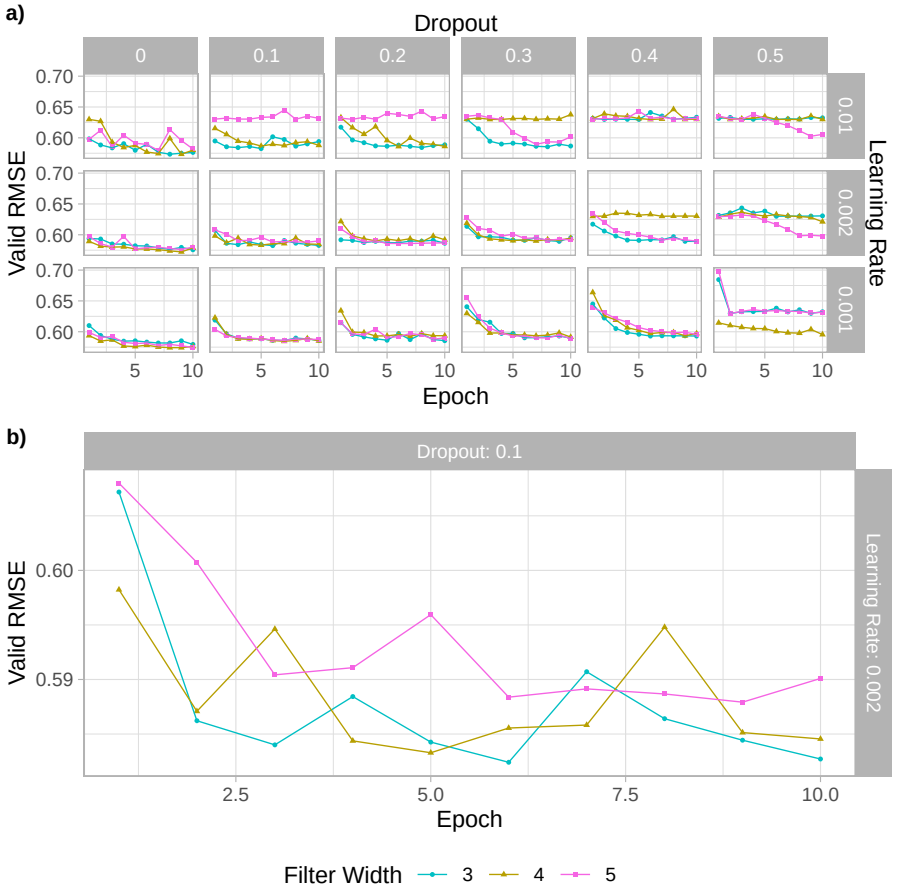

Figure 5.9: Validation RMSE produced by the kNN NN model of size *sss* and tanh final activation function for increasing numbers of epochs, amount of dropout and learning rate applied.

When examining the learning rates in Figure 5.9a, it is clear that overall, the learning rate α of 0.002 led to the most stable decrease in rating prediction errors for all network configurations and is chosen as optimal. Figure 5.9b then plots the optimal CDF NN model of size *sss*, the tanh final activation function, dropout of 0.1 and a learning rate of 0.002. Though the differences between the filter widths is very small, for both the sake of simplicity and as the model with the filter width $h = 3$ produced the lowest validation RMSE at the 10^{th} epoch, it is chosen as optimal.

Using these configurations, the optimal network is trained for a total of 20 epochs (to ensure the stabilisation of the validation errors) and the performance is recorded in Table 5.2. Though the NN validation error yielded is marginally better than the IBCF and CB approaches; the success of the NN models does not extend to the top-Q recommendations produced, however.

As with the other model-based methods (SGD and ALS), the NN approach falls short when producing lists of recommendations; the NN models produced recommendations which covered roughly 1% of all wines available in the catalogue. This result is explained by the personalisation of 0 produced by the NNs - each user is recommended the same set of items. Consequently, the TPR and ARHR results tend to 0 as each user is treated as equal and few users receive any

relevant recommendations.

The neural networks repeated the same set of recommendations likely due to the network being overwhelmed with wine data; when feeding the network, the user is represented by a latent factor of dimension $x = 30$ but the wine is represented by a latent factor in addition to a structured and unstructured numeric representation. This can be observed in Tables A.3 and 5.2 where the RMSE of the NN method is similar for both CDF and kNN datasets indicating that the NNs do not fairly consider the input ratings when producing predictions. Clearly, the user rating latent factors alone are insufficient in distinguishing between users as the network seems to be biased towards wines and not users.

When training, the network learns to predict a rating for a user-wine pair and given that the wine data overshadows the user data, the network likely learnt to predict more general ratings for a wine. In other words, the networks learnt to distinguish between popular and unpopular wines during training. This explains the high rating prediction accuracy of the network - as it is likely that more popular wines are rated higher, the network simply learnt to predict higher ratings for more popular wines. This too explains why the recommendation list quality is so low - the networks simply recommend the most popular wines overall, ignoring the user preferences as the user rating input was not discriminate enough.

This could be remedied by shrinking the wine components of the network structure in Figure 4.2 and expanding the user component. This may not work in practice, however, as the true solution to this problem involves capturing more descriptive user data and building a network supplied with a more equivalent quantity of user and wine data.

Though NNs provide a convenient way to incorporate all available data, the complexity of the network structure in combination with the lack of user data proves this approach to be unsatisfactory for producing recommendations. And though the rating predictive accuracy of the NNs is strong in Table 5.2, the computational effort of combing the SGD latent matrices with the wine data is not justified as the SGD method alone is more accurate still (with values as low as 0.5731 in Table 5.2).

5.7 The Final "soMLier" Recommender System

Using the results above, a final recommendation system "soMLier" is built such that it maximises the accuracy of rating predictions and the quality of recommendations made. The name "soMLier" combines the terms "sommelier" - wine professionals who specialise in recommendations and the Machine Learning abbreviation "ML" used to describe the techniques used by the system. Using a form of *switching* as discussed by Burke (2007), a selection of optimal models from Table 5.1 are combined to create "soMLier".

Importantly, the CDF and kNN imputation protocols are necessary for fairly evaluating and comparing systems. In practice, Algorithms 1, 4 and 5 can sufficiently generate recommendations for a user given that they have made at least one rating. If a user has made no ratings, in the context of this system, they would have made no measurable interactions and thus could not receive any recommendations.

Hence, the final hybrid wine recommender system does not apply the kNN protocol to new users but is built using models that make us of the kNN dataset and its performance is measured

Table 5.4: Evaluation metric results for the final system applied to the kNN-imputed test set.

	RMSE	**COV**	**PER**	**TPR**	**FPR**	**ARHR**
Method	SGD			IBCF		
Test Set Result	0.3346	0.9945	0.9156	0.6487	0.0118	0.5723

using the kNN test set.

5.7.1 System structure

From Table 5.2 and the discussions above, three findings are clear. Firstly, IBCF is by far the best approach to generating lists of quality recommendations to users but does not produce the most accurate rating predictions. Next, CB methods fail to produce useful recommendations as they are too unsurprising but provide sound means to compare how wines are intrinsically related. Lastly, the model-based methods produce accurate predictions using standardised ratings but are limited by only recommending popular items.

Thus, the final system is built such that wine recommendations are made using IBCF, rating predictions for those wine recommendations are computed using model-based methods, and explanations behind the recommendations are derived using CB methods.

Using the optimal hyperparameter configurations in Table 5.1, the recommendations supplied to a user given at least one rating are obtained by:

1. Using Algorithm 5 with the NHSM similarity matrix $\boldsymbol{S}^v$ to generate a list of top-Q wines that the user would be interested in.
2. Standardising and adding the user's ratings to $\zeta(\boldsymbol{R})$ and using SGD (Algorithm 1) to decompose the matrix.
3. Using the new user and wine latent factors to predict ratings for the Q wines recommended to the user.
4. Exploiting the structured CB HAR similarity between the rated and recommended wine attributes in addition to their corresponding reviews and descriptions to explain the recommendations.

When applying this final system to the test set, the final RMSE and quality metric results are recorded in Table 5.4. From this, it is clear that the final system leverages the benefits of each system as the testing RMSE is low and comparable to the validation error observed in Table 5.2. Likewise, the quality of the test recommendations are good with high coverage, personalisation and ARHR values similar to those observed in the validation set.

The main drawback of this system is the repeated decomposition of $\boldsymbol{R}$ each time a new user is added as this is not an instantaneous operation. Regardless, the implementation of Algorithm 1 in `recosystem` is efficient enough and the computation of rating predictions takes no more than a few seconds. The main advantage of the final system, however, is how the recommendations are explained to the user.

5.7.2 Explaining wine recommendations

Once the set of top-Q wines that will be suggested to the user has been established, insights into why the user may enjoy those wines are generated as follows.

1. Taking inspiration from Dillon et al. (2019), a user profile can be derived using the attributes of the wines rated by the user. Establishing a user profile involves the computation of: the number of regions of origin, the median price, the mode wine type and mode body of the wines rated, the average rating awarded as well as the mode grape varietal awarded the highest and lowest rated wines.
2. The same statistics are computed for the set of top-Q wines recommended.
3. The most common words used to describe both the rated and recommended wines are extracted from the wine descriptions and reviews.
4. The structured CB HAR similarity between the rated and recommended wines along with the numeric variables corresponding to each wine is extracted.

Using these components, the wine recommendations are firstly explained textually - consider the following example user profile:

> "You have rated 5 wines from 3 different regions. The average price of your rated wines is R200, and you most commonly rate white wines from the Western Cape. Viogniers are your most favoured as you award them an average rating of 4.5. Pinot Grigios are less enjoyed with an average rating of 2.5. Your top 20 wine recommendations come from 10 different regions. The average price of your recommended wines is R206.5, where mainly white wines from the Western Cape are suggested. Blends and Cabernet Francs are predicted to be the most enjoyed and some recommended Viogniers are also worth trying. The wines you rated are often described with the words 'crisp' and 'light'. Your recommended wines are too described using terms such as 'fresh' and 'dry'."

Then, word clouds containing the most commonly used words to describe both the rated and recommended wines are presented in Figure 5.10 to illustrate how the rated and recommended wines are similarly discussed.

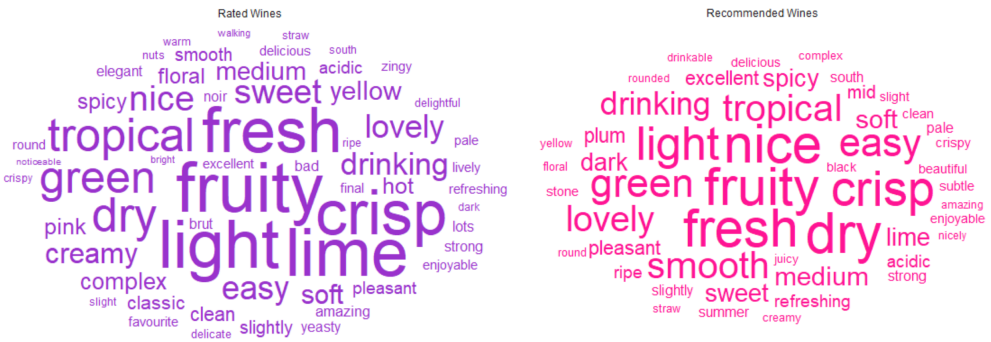

Figure 5.10: Word clouds of the most common words used to describe the rated and recommended wines.

This can be examined alongside a three-dimensional scatter plot of the numeric variables of the

rated and recommended wines such as Figure 5.11. This plot visualises how "close" the wines are when considering their chemistry and price.

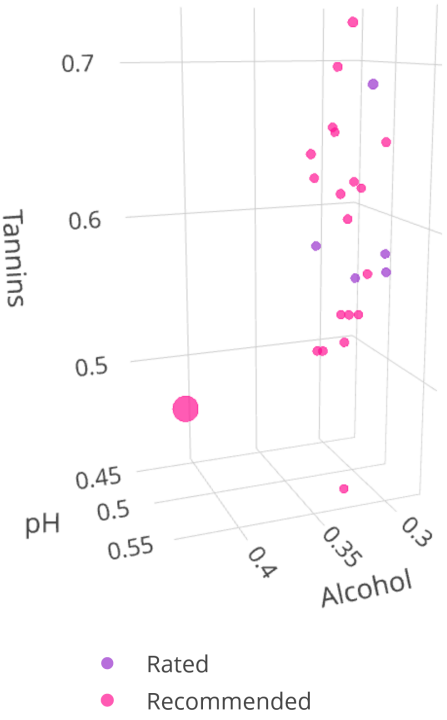

Figure 5.11: Three-dimensional scatter plot of the Alcohol, Tannins and pH of rated and recommended wines where price is indicated by the size of the points.

Lastly, a similarity network as in Figure 5.12 is provided and connects the rated wines (square nodes) and recommended wines (circle nodes) with edges, the thickness of which corresponds to their structured CB similarity. The colour of each node reflects the wine type (red, white, sparkling etc.) and this network provides another depiction of how similar the rated wines are to the recommended according to attributes such as their winery and grape type.

5.7.3 Application of the recommender system

An application of this system is built with Shiny (Chang et al., 2021) and the source code can be found at this link; this demonstrates the functionality of "soMLier" and can be run in `RStudio`[7]. Figure 5.13 displays the homepage of this application. The wine recommendations for a set of people who have previously used this system are available under the "Example Users" Tab. New users may use the system to generate their own wine recommendations under the "New User" Tab. The "Wine Catalogue" Tab may be used to familiarise oneself with the wines available in this system.

A brief guide to the system is as follows. Note that the recommendations may take a few seconds to load.

[7] A web-based version of this application can be found at: https://josh-red.shinyapps.io/soMLier/ and can be run on an internet browser.

Figure 5.12: Network connecting a rated wine (centre) to 20 recommended wines (circumference) where the CB similarity between the wines is indicated by edge thickness.

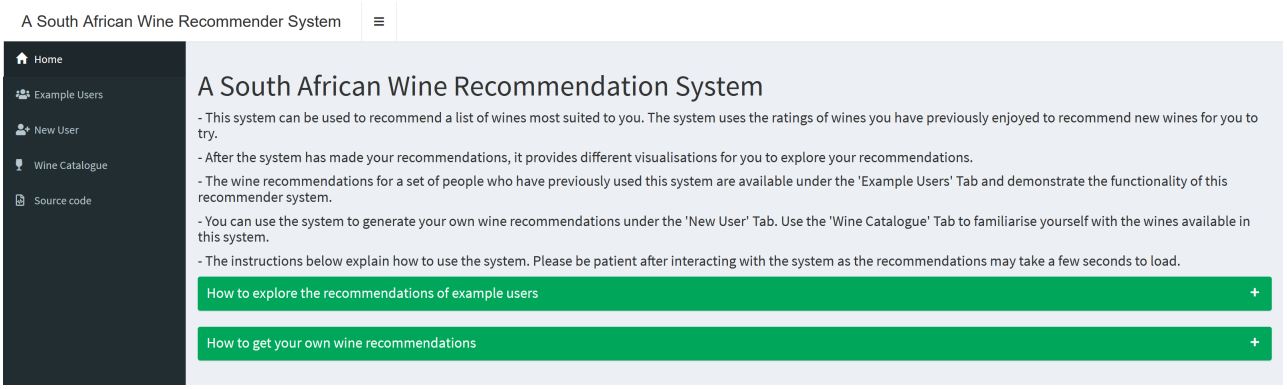

Figure 5.13: Homepage of the "soMLier" recommender system that allows exploration of example user recommendations and the generation of recommendations for new users.

5.7.3.1 Exploring example users

The components (a) to (f) of Figure 5.14 may be used to explore the results of a set of example users in the system, found under the "Example Users" Tab:

(a) Select an example user and the number of wine recommendations the system should generate in the "Inputs" Box. The system predicts the user's ratings for the recommended wines using latent matrices; the dimensions of which can also be changed in the "Inputs" Box.

(b) The "User Profile" describes the wines rated by and recommended to the user. The details of these wines can be explored in the "Rated Wines" and "Recommended Wines" Tabs.

(c) The difference between the actual ratings awarded by the user and the ratings predicted by the system can then be visualised. Notice how the error between the actual and predicted ratings decreases when more dimensions are selected in (a). Likewise, the error of the predicted ratings decreases for users who rated more wines illustrating that greater interaction with the system leads to improved accuracy.

(d) The rated and recommended wines that are most similar according to their attributes (vintage, region, grape, type, style, alcohol, residual sugar, tannins and price) are shown in a network. The thicker the connection between the rated wines (boxes) and recommended wines (circles) the greater their similarity.

(e) The similarity between the rated and recommended wines can also be viewed when considering only alcohol, residual sugar, tannins and price. Each bubble represents a wine, where its size corresponds to its price. The variables are scaled to [0,1] so bubbles closer to 1 reflect higher raw values.

(f) The most common adjectives and adverbs used in the descriptions and reviews of the rated and recommended wines can be used to illustrate how the wines are similar from a textual perspective.

5.7.3.2 Generating recommendations for new users

Under the "New Users" Tab, the components (a) to (f) of Figure 5.15 may be used to generate and explore wine recommendations for a new user:

(a) Use the "Search" Boxes to select between 1 and 5 wines and rate each of them on a scale between 1 and 5 using the sliders alongside.

(b) First click the "Get Recommendations" button to generate your recommendations based on the ratings you made in (a). "Your Profile" describes the wines rated and recommended to you. The details of your rated and recommended wines can be explored in the "Rated Wines" and "Recommended Wines" Tabs.

(c) Next, click the "Get Predicted Ratings" button to calculate your predicted ratings for your wine recommendations. This may take a few seconds. The difference between your ratings predicted by the system and the average ratings awarded to your recommended wines can be visualised in the "Predicted and Average Ratings' Tab.

(d) For each of the 5 wines you rated, the similarity of your recommended wines according to their attributes (vintage, region, grape, type, style, alcohol, residual sugar, tannins and price) are shown in a network. The thicker the connection between the rated wines (boxes) and your recommended wines (circles) the greater their similarity.

(e) The similarity between your rated and recommended wines can also be viewed when considering only alcohol, residual sugar, tannins and price. Each bubble represents a wine, where its size corresponds to its price. The variables are scaled to [0,1] so bubbles closer to 1 reflect higher raw values.

(f) The most common adjectives and adverbs used in the descriptions and reviews of your rated and recommended wines can be used to illustrate how the wines are similar from a textual perspective.

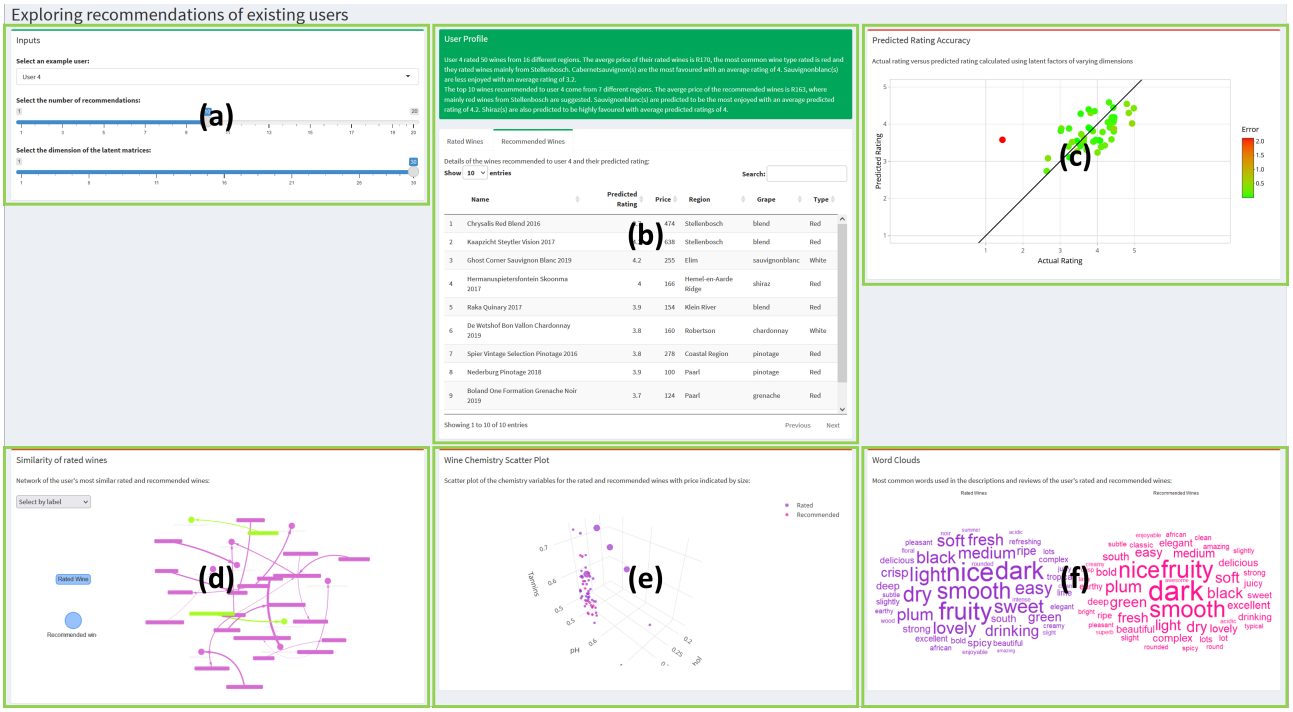

Figure 5.14: Components of the "soMLier" recommender system that allow exploration of the recommendations made to example users.

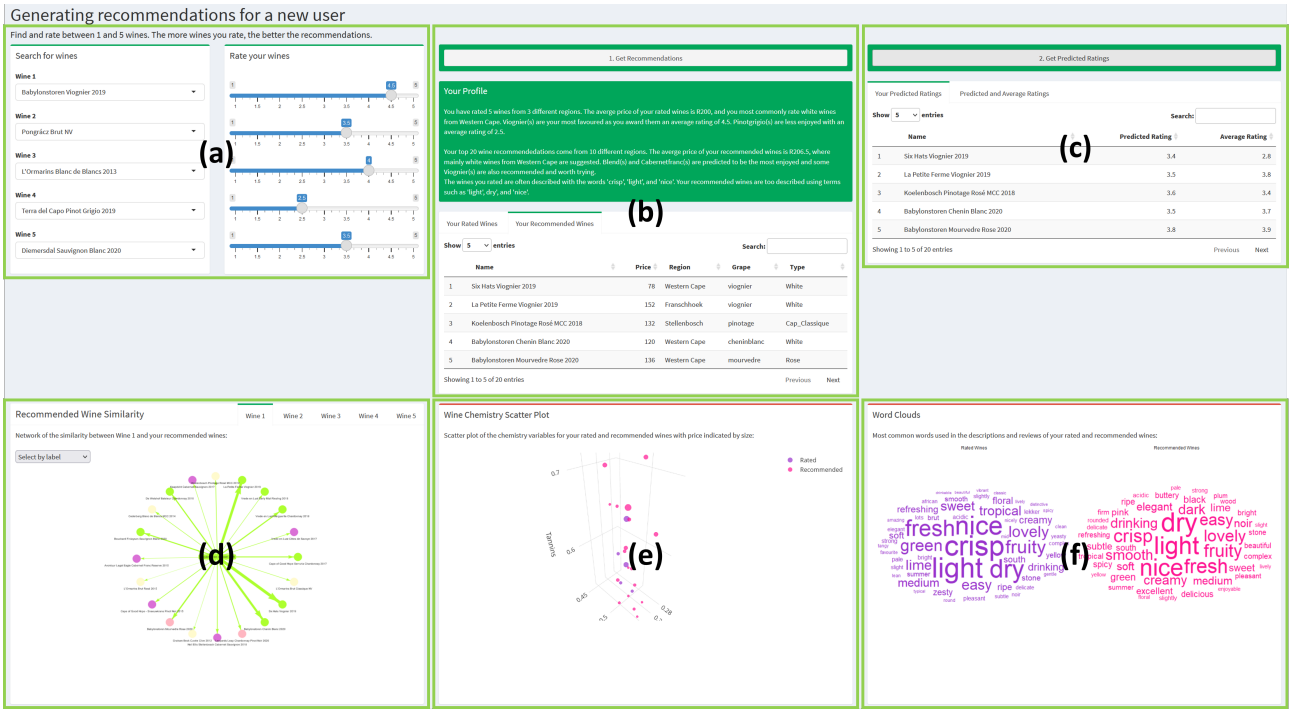

Figure 5.15: Components of the "soMLier" recommender system that allow exploration of the recommendations made to new users.

Chapter 6

Conclusions and Future Work

This thesis aimed to develop a South African wine recommender system that maximised both the accuracy of rating predictions and the quality of recommendations made. This system was built to address the limitations of popular wine recommenders; as current applications cater more specifically to wine consumers in other countries such as America and Europe, access to meaningful South African wine recommendations is restricted. Hence, through the investigation of multiple recommender system methodologies, the "soMLier" system was built using a hybrid approach that integrated several techniques to produce South African wine recommendations.

This hybrid system was developed using a combination of attribute data describing 1,640 wines supplied by wine.co.za and the corresponding ratings and reviews of 92,514 users scraped from Vinino.com. These datasets were used to explore how approaches to computing wine similarity and understanding user preferences could be exploited to generate wine recommendations. In particular, collaborative filtering, content-based, weighted hybrid, matrix factorisation, and neural network methods were examined and compared, the best of which were combined in the final system.

The rating (and corresponding review) data was split into training, validation and test sets and as each rating corresponded to a unique user-wine pair, users were too divided among the sets. This simulated the cold-start problem in which users (and their corresponding ratings) who rated few wines did not appear in the training data; as their preferences could not be measured, they could not be recommended any wines. To mitigate this, two protocols based on the inverse sampling of the empirical wine rating Cumulative Distribution Function (CDF) and k-Nearest Neighbour (kNN) methods, respectively, were implemented.

Both protocols were used to impute a secondary rating for users that had only rated one wine such that all users would exist in the training set. The CDF protocol included an element of randomisation to simulate the arbitrary differences between users' rating patterns, whereas the kNN protocol exploited user similarity to generate a second rating. Likely as a result of the lack of randomisation, however, the kNN protocol was found to more accurately model the rating patterns of users. Thus, when comparing the results obtained by the systems trained using the CDF and kNN imputed data, the kNN protocol led to models which produced more favourable results and more appropriately mitigated the effects of the cold-start problem.

Furthermore, the effects of scaling the rating data were investigated. Primarily, the influence

of user biases in their rating patterns were established by developing and comparing systems built using raw, normalised and standardised (user-mean centred) ratings. It was found that scaling did not effect the content-based methods, had a negligible influence on the item-based collaborative filtering systems and a much larger impact on the model-based methods. Normalisation and standardisation were required by the model-based methods to both remove biases and ensure stable learning.

The performance of each recommender system method implemented was established through their rating predictive accuracy and the quality of the lists of recommendations produced. The Root Mean Square Error (RMSE) between the observed ratings hidden in the validation set and the predicted ratings generated by each method was used to measure how correctly the systems could model and forecast user rating patterns.

Additionally, each system was used to generate lists of wines that each user in the validation set would likely enjoy; these lists were then compared to wines that were liked by users in the validation set to gauge the quality of the recommendations made. Quality was measured explicitly through coverage, personalisation and Average Reciprocal Hit Rate (ARHR) metrics that assessed the portion of the wine catalogue recommended, how unique a set of recommendations are to a user and if the most relevant wines are recommended first, respectively.

Using the accuracy and quality of the recommendations determined with the validation set, hyperparemeter tuning of each system was conducted to build a set of optimal models. When comparing the results yielded by each method's optimal system, the strengths and weaknesses of each were established. Then, performance of the final system (a combination of optimal standalone models) was evaluated using the accuracy and quality of recommendations made for the test set.

Firstly, Item-Based Collaborative Filtering (IBCF) was investigated instead of its user-based counterpart due to the greater stability of item similarity compared to user similarity. IBCF exploits a user's ratings and the similarity between wines (as determined by the groups of users who commonly rated them) to generate predicted ratings for wines not yet rated by a user; these are then used to produce a ranked list of wine suggestions. This approach is disadvantageous as it is both computationally expensive (due to the millions of calculations involved in determining wine similarity) and restricted to the user's past ratings.

Regardless, the optimal IBCF system made use of the New Heuristic Similarity Measure (NHSM) to establish the similarity between wines, leading to a validation RMSE of 0.5938. The quality of recommendations produced by the IBCF system was strongest overall as the coverage and personalisation results were very high with the greatest ARHR of 0.6181 observed. This indicated that, when compared to the results yielded by subsequent methods, IBCF is the best approach to generating lists of relevant wine recommendations.

Next, Content-Based (CB) methods explored the use of structured wine attributes (such as price, vintage and alcohol percentage) and unstructured textual descriptions to determine the similarity between all pairs of wines. Term frequency-inverse document frequency was used to produce numeric representations of the text used to describe each wine that could be compared arithmetically and mixed data distance measures were used to determine the similarity between the numeric and factor wine attributes. This approach to wine similarity could be used as with the IBCF method to predict ratings for unrated wines to generate ranked lists of

recommendations.

Though the content-based methods predicted ratings slightly more accurately than the IBCF method, they failed to produce high-quality wine recommendations. The CB recommendations were very personalised and covered the majority of the wine catalogue but lacked relevancy. When using wine content to determine similarity and generate suggestions, the recommendations tended to lack diversity, were unsurprising and therefore were of little significance to the user. For example, based on content alone, a user who only rated Sauvignon Blancs would only be recommended other dry white wines with no consideration for other types and styles which are of real interest.

As a result, the use of wine content alone was insufficient in generating meaningful suggestions. Even so, it was found that the attributes and text describing the wines could be used to explain the recommendations made by the IBCF method. Using content-based similarity and simple visualisations, the diverse suggestions made using IBCF could be clarified such that the recommendations would no longer seem arbitrary to novice users.

The sparsity of the rating data likely negatively impacted the predictive accuracy of the IBCF and CB methods. Hence, when linearly combining either the rating predictions or similarity values produced by the IBCF and CB methods in weighted hybrid approaches, very minimal performance improvements were observed. Thus, linearly weighted hybrids are not recommended in this context.

Contrarily, the rating predictive accuracy of the model-based methods was superior overall. Two types of matrix factorisation, Stochastic Gradient Descent (SGD) and Alternating Least Squares (ALS) were used to learn latent representations of the users and wines present in the rating data. These representations were then used to predict ratings for a user's unrated wines where those with the greatest predicted ratings would be recommended to the user. The lowest overall validation RMSE of 0.5731 was obtained by the SGD approach trained on rating data that had been standardised, where poorer accuracy was recorded for the raw and normalised ratings. Equivalent results were obtained by the ALS method, highlighting the fact that matrix factorisation can accurately model and predict user rating patterns.

The flexibility of the neural network approach allowed for the extension of the matrix factorisation models by combing the user and wine rating latent representations with numeric encodings of the wines' attributes, descriptions and written reviews. All available training data was fed to this neural network structure to predict ratings for a user's unrated wines and led to validation RMSE results equally as low as the matrix factorisation approach. However, as no substantial performance benefits were recorded when incorporating the user and wine latent factors with the wine attributes in this neural structure, the complexity involved in building such a network was not merited and is not recommended when the simpler matrix factorisation was more accurate.

Though both model-based approaches yielded strong predictive accuracy, they produced poor lists of recommendations with the lowest results for the quality metrics overall. In both the matrix factorisation and NN methods, the models built were biased towards the most popular wines. In effect, each user would be recommended the wines that were rated the highest, despite their personal preferences. In the case of the NN, this was likely a result of the network being supplied with substantially more wine data (such as the wine attributes, descriptions and

reviews) than user data as only the users' rating patterns were available.

Regardless, it was established that due to the high accuracy of the matrix factorisation technique, it would be best suited to predicting ratings but should not be used to generate recommendations.

Using these findings, the final "soMLier" system was designed as a *switching* hybrid that combined the strengths of each individual approach by using different techniques for different components of the system. Hence, the optimal IBCF system is used to generate a list of wines a user will likely enjoy. The CB similarity between wines, determined through their attributes and descriptions, is then used to rationalise these recommendations by explaining how the recommended wines are similar to those rated by a user. Lastly, SGD matrix decomposition is used to predict the ratings that the user would award their recommended wines to serve as an estimation of much they would enjoy their suggestions.

This final system produced superior test set results with a very low RMSE of 0.3346, high coverage and personalisation in addition to an average reciprocal hit rate of 0.5723 - a value greater than all other standalone system validation results (barring the original IBCF system).

The greater success of the model-based and IBCF methods (which are both considered collaborative filtering techniques) compared to the CB approach was largely a result of the data available. As only user rating data was available, developing inferences on their preferences was limited, thereby impeding the performance of the CB systems. This was not an issue for the IBCF and model-based methods as they only require rating data to perform well. Hence, as in this case, when the volume of rating data exceeds that of user data, the use of collaborative-filtering should be prioritised.

In future, this system can be further improved upon by:

1. gathering richer user data (such as age, location or purchase history) so that a more detailed understanding of user preferences can be exploited,
2. incorporating user attributes with wine data when producing content-based recommendations,
3. further regularising the matrix methods to account for the bias towards certain wines,
4. experimenting with other neural network structures that propagate user and wine data in a more balanced way such that greater discrimination between users is achieved,
5. exploring alternative approaches to collaborative filtering such as user clustering and user-based similarity, and
6. investigating other methods of mitigating the cold-start problem.

Although the final system was limited by the lack of detailed user data that could be used to understand user preferences, it addresses the aim of this thesis as the test results of the final system demonstrate how it can be used to generate accurate, high-quality recommendations. The "soMLier" system, as far as we know, uses a novel approach of integrating traditional methods (IBCF and CB) with more modern techniques (matrix factorisation) to generate and rationalise its recommendations. Consequently, this system addresses a gap in the current

offering of wine recommenders by not only supplying explanations behind the recommendations made but also providing suggestions for wines specifically produced and sold in South Africa.

Bibliography

Aditya, P.H., Indra Budi, and Qorib Munajat (2017). "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 2016*, pp. 303–308.

Aggarwal, Charu C. (2016). *Recommender Systems: The Textbook*.

Aggarwal, Charu C., Alexander Hinneburg, and Daniel A. Keim (2001). "On the surprising behavior of distance metrics in high dimensional space". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 1973, pp. 420–434.

Ahmad, Amir and Lipika Dey (2007). "A k-mean clustering algorithm for mixed numeric and categorical data". In: *Data and Knowledge Engineering* 63.2, pp. 503–527.

Allaire, J.J. and François Chollet (2022). *keras: R Interface to 'Keras'*.

Almasi, George S. and Albert J. Lee (1999). "A PDA-Based Personalised Recommender Agent". In: *IBM Thomas J. Watson Research Division*.

Barragáns-Martínez, Ana Belén, Enrique Costa-Montenegro, Juan C. Burguillo, Marta Rey-López, Fernando A. Mikic-Fonte, and Ana Peleteiro (2010). "A hybrid content-based and item-based collaborative filtering approach to recommend TV programs enhanced with singular value decomposition". In: *Information Sciences* 180.22, pp. 4290–4311.

Bell, Robert, Chris Volinsky, and Yehuda Koren (2009). "Matrix Factorization Techniques For Recommender Systems". In: *Computer* 42.8, pp. 30–37.

Bennett, James, Charles Elkan, Bing Liu, Padhraic Smyth, and Domonkos Tikk (2007). "KDD Cup and workshop 2007". In: *ACM SIGKDD Explorations Newsletter* 9.2, pp. 51–52.

Billsus, Daniel and Michael J. Pazzani (2000). "User modeling for adaptive news access". In: *User Modelling and User-Adapted Interaction* 10.2-3, pp. 147–180.

Bishnoi, Sudha and BK Hooda (2020). "A survey of distance measures for mixed variables". In: *International Journal of Chemical Studies* 8.4, pp. 338–343.

Bobadilla, Jesús, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez (2013). "Recommender systems survey". In: *Knowledge-Based Systems* 46, pp. 109–132.

Bookstein, Abraham, Vladimir A. Kulyukin, and Timo Raita (2002). "Generalized hamming distance". In: *Information Retrieval* 5.4, pp. 353–375.

Budiaji, Weksi (2021). *kmed: Distance-Based k-Medoids*.

Burke, Robin (1999a). "Integrating Knowledge-based and Collaborative-filtering Recommender Systems". In: *Proceedings of the Workshop on AI and Electronic Commerce*, pp. 69–72.

— (1999b). "Wasabi personal shopper: a case-based recommender system". In: *Proceedings of the National Conference on Artificial Intelligence*, pp. 844–849.

Burke, Robin (2000). "Knowledge-based recommender systems". In: *Encyclopedia of library and information systems* 69.Supplement 32, pp. 175–186.

— (2002). "Hybrid Recommender Systems: Survey and Experiments". In: *User Modeling and User-Adapted Interaction* 12.4, pp. 331–370.

— (2007). "Hybrid web recommender systems". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 4321 LNCS, pp. 377–408.

Catherine, Rose and William Cohen (2017). "TransNets: Learning to transform for recommendation". In: *RecSys 2017 - Proceedings of the 11th ACM Conference on Recommender Systems*, pp. 288–296.

Chang, Winston, Joe Cheng, J.J. Allaire, Carson Sievert, Barret Schloerke, Yihui Xie, Jeff Allen, Jonathan McPherson, Alan Dipert, and Barbara Borges (2021). *shiny: Web Application Framework for R*.

Cruz, Christophe, Cyril Van Nguyen, and Laurent Gautier (2018). "Word Embeddings for Wine Recommender Systems Using Vocabularies of Experts and Consumers". In: *Open Journal of Web Technologies, RonPub* 5.1, pp. 23–30.

Dillon, Stephen, Pamela Dillon, and Andrew Sussman (2019). *Personal Taste Assessment Method and System*.

Dziugaite, Gintare Karolina and Daniel M. Roy (2015). "Neural Network Matrix Factorization". In: *arXiv preprint arXiv:1511.06443*, pp. 1–7.

Goldberg, David, David Nichols, Brian M. Oki, and Douglas Terry (1992). "Using collaborative filtering to Weave an Information tapestry". In: *Communications of the ACM* 35.12, pp. 61–70.

Goode, Jamie (2020). *Wine Science: The Application of Science in Wine from Vine to Glass*. Octopus Books.

Gower, John C. (1971). "A general coefficient of similarity and some of its properties". In: *Biometrics*, pp. 857–871.

Gu, Jiuxiang, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, and Tshuan Chen (2018). "Recent advances in Convolutional Neural Networks". In: *Pattern Recognition* 77, pp. 354–377.

Guarino, Nicola, Daniel Oberle, and Steffen Staab (2009). "What is an Ontology". In: *Handbook on Ontologies*, pp. 1–17.

Harikumar, Sandhya and P.V. Surya (2015). "K-Medoid Clustering for Heterogeneous DataSets". In: *Procedia Computer Science* 70, pp. 226–237.

Hassanieh, Lamis Al, Chadi Abou Jaoudeh, Jacques Bou Abdo, and Jacques Demerjian (2018). "Similarity measures for collaborative filtering recommender systems". In: *2018 IEEE Middle East and North Africa Communications Conference, MENACOMM 2018*, pp. 1–5.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2009). *The Elements of Statistical Learning: data mining, inference, and prediction*. Vol. 2. Springer.

HelloVino (2022). *Hello Vino: Wine Assistant App Hello Vino: Wine Assistant App*. Available: http://www.hellovino.com/. [Accessed: 20-Jun-2022].

Hrnjica, Bahrudin, Denis Music, and Selver Softic (2020). "Model-based recommender systems". In: *Trends in Cloud-based IoT*, pp. 125–146.

Huang, Zhexue (1997). "Clustering large data sets with mixed numeric and categorical values". In: *Proceedings of the 1st pacific-asia conference on knowledge discovery and data mining, (PAKDD)*, pp. 21–34.

Jadhav, Anil, Dhanya Pramod, and Krishnan Ramanathan (2019). "Comparison of Performance of Data Imputation Methods for Numeric Dataset". In: *Applied Artificial Intelligence* 33.10, pp. 913–933.

Johnson, Hugh (1991). *Wine Companion: The Encyclopedia of Wines, Vineyards, & Winemakers*. Octopus Books.

Khan, Koffka and Ashok Sahai (2012). "A Comparison of BA, GA, PSO, BP and LM for Training Feed forward Neural Networks in e-Learning Context". In: *International Journal of Intelligent Systems and Applications* 4.7, pp. 23–29.

Kim, Donghyun, Chanyoung Park, Jinoh Oh, Sungyoung Lee, and Hwanjo Yu (2016). "Convolutional matrix factorization for document context-aware recommendation". In: *RecSys 2016 - Proceedings of the 10th ACM Conference on Recommender Systems*, pp. 233–240.

Kolodner, Janet L. (1992). "An Introduction to Case-Based Reasoning". In: *Artificial intelligence review* 6.1, pp. 3–34.

Kotonya, Neema, Paolo De Cristofaro, and Emiliano De Cristofaro (2018). "Of wines and reviews: Measuring and modeling the vivino wine social network". In: *Proceedings of the 2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2018*. Asonam, pp. 387–392.

Krötzsch, Markus, Denny Vrandeči, and Max Völkel (2006). "Semantic MediaWiki". In: *International semantic web conference*, pp. 935–942.

Kumar, P. Pavan, S. Vairachilai, Sirisha Potluri, and Sachi Nandan Mohanty (2021). *Recommender Systems: Algorithms and Applications*, pp. 2013–2015.

Lawrence, Richard D., George S. Almasi, and Holly E. Rushmeier (1999). "A scalable parallel algorithm for self-organizing maps with applications to sparse data mining problems". In: *Data Mining and Knowledge Discovery* 3.2, pp. 171–195.

Li, Piji, Zihao Wang, Zhaochun Ren, Lidong Bing, and Wai Lam (2017). "Neural rating regression with abstractive tips generation for recommendation". In: *SIGIR 2017 - Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 345–354.

Liu, Donghua, Jing Li, Bo Du, Jun Chang, Rong Gao, and Yujia Wu (2021). "A hybrid neural network approach to combine textual information and rating information for item recommendation". In: *Knowledge and Information Systems* 63.3, pp. 621–646.

Liu, Haifeng, Zheng Hu, Ahmad Mian, Hui Tian, and Xuzhen Zhu (2014). "A new user similarity model to improve the accuracy of collaborative filtering". In: *Knowledge-Based Systems* 56, pp. 156–166.

Liu, Hongtao, Yian Wang, Qiyao Peng, Fangzhao Wu, Lin Gan, Lin Pan, and Pengfei Jiao (2020). "Hybrid neural recommendation with joint deep representation learning of ratings and reviews". In: *Neurocomputing* 374, pp. 77–85.

Lops, Pasquale (2011). *Recommender Systems Handbook*, pp. 73–105.

Lu, Yichao, Ruihai Dong, and Barry Smyth (2018). "Coevolutionary recommendation model: Mutual learning between ratings and reviews". In: *The Web Conference 2018 - Proceedings of the World Wide Web Conference, WWW 2018*, pp. 773–782.

Lü, Linyuan, Matúš Medo, Chi Ho Yeung, Yi Cheng Zhang, Zi Ke Zhang, and Tao Zhou (2012). "Recommender systems". In: *Physics Reports* 519.1, pp. 1–49.

Luraschi, Javier, Kevin Kuo, Kevin Ushey, J.J. Allaire, Hossein Falaki, Lu Wang, Andy Zhang, Yitao Li, Edgar Ruiz, and The Apache Software Foundation (2022). *sparklyr: R Interface to Apache Spark*.

Lynch, John G. and Dan Ariely (2000). "Wine online: Search costs affect competition on price, quality, and distribution". In: *Marketing Science* 19.1, pp. 83–103.

Meira, Dânia, José Viterbo, and Flavia Bernardini (2018). "An experimental analysis on scalable implementations of the alternating least squares algorithm". In: *Proceedings of the 2018 Federated Conference on Computer Science and Information Systems, FedCSIS 2018* 15.i, pp. 351–359.

Melville, Prem, Raymond J. Mooney, and Ramadass Nagarajan (2002). "Content-Boosted Collaborative Filtering for Improved Recommendations". In: *Aaai/iaai* 23, pp. 187–192.

Melville, Prem and Vikas Sindhwani (2010). "Encyclopaedia of Machine Learning: Recommender Systems". In: *Encyclopaedia of Machine Learning*, pp. 829–838.

Michaelis, James R., Li Ding, and Deborah L. McGuinness (2008). "The TW wine agent: A social semantic web demo". In: *CEUR Workshop Proceedings*. Vol. 401, pp. 2–4.

Mobasher, Bamshad, Xin Jin, and Yanzan Zhou (2004). "Semantically enhanced collaborative filtering on the web". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 3209, pp. 57–76.

Mooney, Raymond J. and Loriene Roy (2000). "Content-based book recommending using learning for text categorization". In: *Proceedings of the ACM International Conference on Digital Libraries*, pp. 195–204.

Morrot, Gil, Frédéric Brochet, and Denis Dubourdieu (2001). "The color of odors". In: *Brain and Language* 79.2, pp. 309–320.

Mulligan, Patrick and Ben Gibson (2016). *A Visual Guide to Drink*. Penguin Books.

Noy, Natalya F. and Deborah L. Mcguinness (2001). "Ontology development 101: A guide to creating your first ontology". In: pp. 1–25.

Olsson, Donald M. and Lloyd S. Nelson (1975). "The nelder-mead simplex procedure for function minimization". In: *Technometrics* 17.1, pp. 45–51.

Paradarami, Tulasi K., Nathaniel D. Bastian, and Jennifer L. Wightman (2017). "A hybrid recommender system using artificial neural networks". In: *Expert Systems with Applications* 83, pp. 300–313.

Parker, Robert M. and Pierre-Antoine Rovani (2002). *Parker's Wine Buyer's Guide*. Simon and Schuster, p. 5.

Patra, Sukanya and Boudhayan Ganguly (2019). "Improvising Singular Value Decomposition by KNN for Use in Movie Recommender Systems". In: *Journal of Operations and Strategic Planning* 2.1, pp. 22–34.

Patton, Evan W. and Deborah L. McGuinness (2009). "The mobile wine agent: Pairing wine with the social semantic web". In: *CEUR Workshop Proceedings*. Vol. 520.

Pazzani, Michael and Daniel Billsus (1997). "Learning and Revising User Profiles: The Identification of Interesting Web Sites". In: *Machine Learning* 27.3, pp. 313–331.

Pazzani, Michael J. and Daniel Billsus (2007). "Content-based recommendation systems". In: *The adaptive web*. Springer, pp. 325–341.

Pennington, Jeffrey, Richard Socher, and Christopher D. Manning (2014). "GloVe: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543.

Podani, János (1999). "Extending Gower's general coefficient of similarity to ordinal characters". In: *Taxon* 48.2, pp. 331–340.

Puckette, Madeline and Justin Hammack (2018). *Wine Folly*. Penguin Random House.

Qiu, Yixuan, David Cortes, Chih-Jen Lin, Yu-Chin Juan, Wei-Sheng Chin, Yong Zhuang, Bo-Wen Yuan, Meng-Yuan Yang, and other contributors. See file AUTHORS for details. (2021). *recosystem: Recommender System using Matrix Factorization*.

R Core Team (2020). *R: A Language and Environment for Statistical Computing*. Vienna, Austria.

Ricci, Francesco, Lior Rokach, Bracha Shapira, Paul B Kantor, and Francesco Ricci (2011). *Recommender Systems Handbook*.

Sáenz-Navajas, María Pilar, Eva Campo, Angela Sutan, Jordi Ballester, and Dominique Valentin (2013). "Perception of wine quality according to extrinsic cues: The case of Burgundy wine consumers". In: *Food Quality and Preference* 27.1, pp. 44–53.

Sarwar, Badrul, George Karypis, Joseph Konstan, and John Riedl (2001). "Item-based collaborative filtering recommendation algorithms". In: *Proceedings of the 10th International Conference on World Wide Web, WWW 2001*, pp. 285–295.

Schröder, Gunnar, Maik Thiele, and Wolfgang Lehner (2011). "Setting goals and choosing metrics for recommender system evaluations". In: *CEUR Workshop Proceedings* 811, pp. 78–85.

Seo, Sungyong, Jing Huang, Hao Yang, and Yan Liu (2017). "Interpretable convolutional neural networks with dual local and global attention for review rating prediction". In: *RecSys 2017 - Proceedings of the 11th ACM Conference on Recommender Systems*, pp. 297–305.

Sharma, Siddharth, Simone Sharma, and Athaiya Anidhya (2020). "Understanding Activation Functions in Neural Networks". In: *International Journal of Engineering Applied Sciences and Technology* 4.12, pp. 310–316.

Shi, Yue, Martha Larson, Alexandros Karatzoglou, Nuria Oliver, Linas Baltrunas, and Alan Hanjalic (2012). "CLiMF: Learning to maximize reciprocal rank with collaborative less-is-more filtering". In: *RecSys'12 - Proceedings of the 6th ACM Conference on Recommender Systems*, pp. 139–146.

Sippd (2022). *Sippd - Your Personal Sommelier*. Available: https://sippd.com/. [Accessed: 20-Jun-2022].

Smith, Barry C. (2019). "Getting More Out of Wine: wine experts, wine apps and sensory science". In: *Current Opinion in Food Science* 27, pp. 123–129.

Smyth, Barry and Paul Cotter (2000). "Personalized TV listings service for the digital TV age". In: *Knowledge-Based Systems* 13.2-3, pp. 53–59.

Studer, Rudi, V. Richard Benjamins, and Dieter Fensel (1998). "Knowledge Engineering: Principles and methods". In: *Data and Knowledge Engineering* 25.1-2, pp. 161–197.

Tassiopoulos, Dimitri, Nancy Nuntsu, and Norbert Haydam (2004). "Wine tourists in South Africa: A demographic and psychographic study". In: *Journal of Wine Research* 15.1, pp. 51–63.

Vivino (2022). *Vivino - Buy the Right Wine*. Available: https://www.vivino.com/. [Accessed: 20-Jun-2022].

Walfish, Steven (2006). "A review of statistical outlier methods". In: *Pharmaceutical Technology* 30.11, pp. 82–86.

Wang, Hao, Naiyan Wang, and Dit Yan Yeung (2015). "Collaborative deep learning for recommender systems". In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* 2015-Augus, pp. 1235–1244.

Williams, David (2013). *A Little Course in Wine Tasting: Simply Everything You Need to Succeed*. Dorling Kindersley Ltd.

WineNet (2022). *wine.co.za - Virtual Home of South African Wine*. Available: https://wine.co.za/. [Accessed: 20-Jun-2022].

WineRing (2021). *WineRing*. Available: https://www.winering.com/. [Accessed: 20-Jun-2021].

Wishart, David (2003). "K-means clustering with outlier detection, mixed variables and missing values". In: *Exploratory Data Analysis in Empirical Research*. Springer, pp. 216–226.

Wu, Libing, Cong Quan, Chenliang Li, Qian Wang, Bolong Zheng, and Xiangyang Luo (2019). "A context-aware user-item representation learning for item recommendation". In: *ACM Transactions on Information Systems* 37.2.

Xi, Wu-dong, Ling Huang, Chang-dong Wang, Yin-yu Zheng, and Jian-huang Lai (2021). "Deep Rating and Review Neural Network for Item Recommendation". In: *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–11.

Xue, Gui Rong, Chenxi Lin, Qiang Yang, Wensi Xi, Hua Jun Zeng, Yong Yu, and Zheng Chen (2005). "Scalable collaborative filtering using cluster-based smoothing". In: *SIGIR 2005 - Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 114–121.

Zhang, Shuai, Lina Yao, Aixin Sun, and Yi Tay (2019). "Deep learning based recommender system: A survey and new perspectives". In: *ACM Computing Surveys* 52.1, pp. 1–35.

Zhang, Ye and Byron Wallace (2015). "A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification". In: *arXiv preprint arXiv:1510.03820*.

Zhang, Yongfeng, Qingyao Ai, Xu Chen, and W. Bruce Croft (2017). "Joint representation learning for top-N recommendation with heterogeneous information sources". In: *International Conference on Information and Knowledge Management, Proceedings* Part F1318, pp. 1449–1458.

Zhou, Yunhong, Dennis Wilkinson, Robert Schreiber, and Rong Pan (2008). "Large-Scale Parallel Collaborative Filtering for the Netflix Prize". In: *Algorithmic Aspects in Information and Management*, pp. 337–348.

Appendices

A.1 Similarity Measures

The following eight similarity measures investigated in this thesis may be modified and applied row-wise to $\boldsymbol{R}$ to produce the user similarity matrix $\boldsymbol{S}^u$, but focus will instead be placed on the column-wise formulas needed to produce the item similarity matrix $\boldsymbol{S}^v$. In all formulas, U_{ab} is the set of users who rated both items a and b, and r_{ua} and r_{ub} are the ratings made by user u on items a and b respectively. Unless stated otherwise, a similarity of 1 corresponds to two perfectly similar items and values of 0 indicate perfectly dissimilar items.

1. **Euclidean Similarity**

   The Euclidean distance formula $\sqrt{\sum_{u \in U_{ab}} (r_{ua} - r_{ub})^2}$ measures the distance between two vectors. To bind this distance to the range [0,1] such that maximally similar vectors yield values of 1 (i.e. the root of the sum of squared differences is 0), it is scaled as in equation (A.1.1) to produce the Euclidean *similarity*.

   This measure can be thought of as the straight-line distance between two coordinates in Euclidean space. Whilst this measure is simple, it is known that the principles of Euclidean space fail in higher dimensions. This measure may be flawed when used to compute the similarity between vectors of over 1,000 dimensions, as elements in high-dimensional space are commonly equidistant (Aggarwal et al., 2001).

   To avoid the erroneous similarity score of 1 when no common users exist between two items ($|U_{ab}| = 0$), the similarity is set to 0.

$$EUC(a,b) = \begin{cases} \frac{1}{1+\sqrt{\sum_{u \in U_{ab}} (r_{ua} - r_{ub})^2}} & \text{if } |U_{ab}| \geq 1 \\ 0 & \text{otherwise} \end{cases} \tag{A.1.1}$$

2. **Pearson Correlation Coefficient**

   Pearson correlation is a widely used similarity measure which measures how correlated two vectors are; perfectly similar and dissimilar vectors yield values of 1 and -1, respectively.

$$PCC(a,b) = \frac{\sum_{u \in U_{ab}} (r_{ua} - \bar{r}_a)(r_{ub} - \bar{r}_b)}{\sqrt{\sum_{u \in U_{ab}} (r_{ua} - \bar{r}_a)^2 \sum_{u \in U_{ab}} (r_{ub} - \bar{r}_b)^2}} \tag{A.1.2}$$

   where $\bar{r}_a$ is the mean rating of item a. This measure fails in situations where $r_{ua} = \bar{r}_a$ or $r_{ub} = \bar{r}_b$ for all $u \in U_{ab}$ as the denominator equates to 0. Likewise, when no users have rated an item pair in common ($|U_{ab}| = 0$), this similarity measure also fails (Hassanieh et al., 2018).

   Though the local mean item rating $\bar{r}_a$ can be used (i.e. the mean rating of the item a made by the subset of users U_{ab}), the global mean rating is used instead (Aggarwal, 2016).

3. **Constrained Pearson Correlation**

   A modified version of the Pearson correlation has been intruded such that the global median of the rating scale r_{med} (3 in the case of ratings in [1,5]) is used instead of the

median to improve the credibility of the measure (Liu, Hu, et al., 2014):

$$CPCC(a,b) = \frac{\sum_{u \in U_{ab}} (r_{ua} - r_{\text{med}})(r_{ub} - r_{\text{med}})}{\sqrt{\sum_{u \in U_{ab}} (r_{ua} - r_{\text{med}})^2 \sum_{u \in U_{ab}} (r_{ub} - r_{\text{med}})^2}} \tag{A.1.3}$$

Using the theoretical median rating to scale the ratings instead of the item means poses complications as rating values of 3 account for roughly 14% of all ratings and $|U_{ab}|$ is often 1. This increases the chances that $r_{ua} = r_{\text{med}}$, thereby reducing the number of similarity calculations which are feasible with the CPCC formula.

4. **Cosine Similarity**

   The cosine similarity measures the cosine of the angle between two vectors; the major drawback of which being it does not take into consideration the magnitude of the ratings made. As only the angle between two vectors is measured, the cosine similarity between $\{1, 1, 1\}$ and $\{10, 10, 10\}$, for example, would be 1 - a perfect similarity score; those two vectors, however, are clearly not perfectly similar. When the ratings are all positive, the minimum cosine similarity is 0 and is -1 for ratings with possible negative values (Hassanieh et al., 2018):

$$COS(a,b) = \frac{\sum_{u \in U_{uab}} r_{ua} r_{ub}}{\sum_{u \in U_{ab}} r_{ua}^2 \sum_{u \in U_{ab}} r_{ub}^2} \tag{A.1.4}$$

5. **Mean Square Difference**

   The mean square difference between two vectors measures the average distance between points in two vectors; because of the squared term, larger differences are penalised more greatly. After scaling such that perfectly dissimilar vectors yield 0 and perfectly similar vectors yield 1, the formula takes the form (Hassanieh et al., 2018):

$$MSD(a,b) = \frac{|U_{ab}|}{|U_{ab}| + \sum_{u \in U_{ab}} (r_{ua} - r_{ub})^2} \tag{A.1.5}$$

6. **Jaccard Distance**

   The Jaccard distance measures only the size of the sets of users who co-rated the items to calculate a similarity value - this measure is flawed as it does not consider the values of the ratings of those co-rated items at all (Liu, Hu, et al., 2014). Simply, it measures the number of users who rated two items together as a positive fraction of all of the users who rated either item:

$$JAC(a,b) = \frac{|U_a \cap U_b|}{|U_a \cup U_b|} \tag{A.1.6}$$

   Maximum values for the Jaccard distance near 1 indicate that most users who rated either item also rated both items together and thus the items should be considered similar. As this measure does not consider the magnitude of the ratings made, it does not account for situations in which a user may have liked item a and disliked item b - those cases erroneously contribute to a positive Jaccard similarity score just as much as situations in which users liked both items.

7. **Weighted Pearson Correlation**

The Weighted Pearson Correlation is the same as the Pearson correlation, barring the inclusion of an additional weight term (Hassanieh et al., 2018):

$$WPC(a,b) = \frac{\sum_{u \in U_{ab}} w_u \cdot (r_{ua} - \bar{r}_a) \cdot (r_{ub} - \bar{r}_b)}{\sqrt{\sum_{u \in U_{ab}} w_u \cdot (r_{ua} - \bar{r}_a)^2} \sqrt{\sum_{u \in U_{ab}} w_u \cdot (r_{ub} - \bar{r}_b)^2}} \tag{A.1.7}$$

where w_u is the weight of each user and $\bar{r}_a$ is the mean rating of item a. The weight for each user is calculated as:

$$w_u = \log\left(\frac{m}{m_u}\right) \tag{A.1.8}$$

where m is the total number of items and m_u is the number of ratings user u made.

In the item-based setting, the purpose of the weight term is to lower the impact of users who made many ratings and increase the impact of users who made few ratings. In altering the impact of certain users' ratings when calculating item-based similarity, a user who rated few items (and had a large weight) would contribute more greatly to the similarity between two items than a user who rated many items. That is to say that, a user who only rated two items and gave them similar scores is a greater indicator of item similarity than a user who rated many items and awarded two items a similar score.

8. **New Heuristic Similarity Measure**

After evaluating and comparing similarity measures, Liu, Hu, et al. (2014) posited an improved New Heuristic Similarity Model (NHSM $\in (0,1)$) defined as[1]:

$$NHSM(a,b) = PSS(a,b) \cdot JAC'(a,b) \cdot IRP(a,b) \tag{A.1.9}$$

$$PSS(a,b) = \text{Proximity}(u,j) \cdot \text{Significance}(u,j) \cdot \text{Singularity}(u,j) \tag{A.1.10}$$

$$\text{Proximity}(a,b) = 1 - \frac{1}{1 + \exp(-|r_{ua} - r_{ub}|)} \tag{A.1.11}$$

$$\text{Significance}(a,b) = \frac{1}{1 + \exp(-|r_{ua} - r_{\text{med}}| \cdot |r_{ub} - r_{\text{med}}|)} \tag{A.1.12}$$

$$\text{Singularity}(a,b) = 1 - \frac{1}{1 + \exp(-|\frac{r_{ua} - r_{ub}}{2} - \mu_u|)} \tag{A.1.13}$$

$$JAC'(a,b) = \frac{|U_a \cap U_b|}{|U_a| \cdot |U_b|} \tag{A.1.14}$$

$$IRP(a,b) = 1 - \frac{1}{1 + \exp(-|\bar{r}_a - \bar{r}_b| \cdot |s_a - s_b|)} \tag{A.1.15}$$

where the proximity is a measure of the distance between ratings, significance is a measure of the deviation of ratings from the median (higher and lower ratings are more significant than common ratings), and singularity measures how a pair of ratings deviates from the mean rating of an item. These measures are combined with a modified Jaccard

[1]Note that this measure was defined to calculate similarity between users, and has thus been adjusted to determine item similarity. For example, the IRP was originally a User Rating Preference (URP) measure, where user means and standard deviations were considered instead.

measure (JAC′) which analyses the proportion of common ratings made and an Item Rating Preference (IRP) measure, which incorporates the rating patterns of each item through the standard deviation (s_a) and mean ($\bar{r}_a$) of their ratings. Liu, Hu, et al. (2014) demonstrated how this measure provided the most accurate measure of similarity between two users when applied to user-based CF specifically.

A.2 Distance Measures

The six measures used to compute the similarity between items are summarised below and are implemented using the `kmed` function in `R` (Budiaji, 2021). Note, distance is measured differently to similarity; a distance of 0 between two items implies that they are identical and larger (potentially unbounded) distances imply dissimilarity. To ensure comparability between similarity and distance, the result of each distance measure below is transformed using:

$$d'_{ab} = 1 - \frac{d_{ab} - \min d}{\max d - \min d} \tag{A.2.1}$$

where $\min d$ and $\max d$ represent the minimum and maximum computed distance for a given distance measure d across all item pairs a, b. After transforming, $\dot{d}_{ab}$ is bound in [0,1] where values closer to 0 represent strong dissimilarity and values closer to 1 indicate perfect similarity, as with the similarity measures in Section A.1.

Additionally, as the numeric variables in a dataset are typically measured on different scales and are not always assumed to be normally distributed, the numeric variables should be normalised (see equation (3.5.1)) before applying the distance measures as variables of differing magnitudes may bias the results of the distance calculations.

1. **Gower Distance**

   Gower (1971) posited a distance measure for mixed data types which is widely used:

   $$GOW(a, b) = 1 - \tau_{ab} \tag{A.2.2}$$

   where τ_{ab} is computed over P variables:

   $$\tau_{ab} = \frac{\sum_{l=1}^{P} \omega_{abl}\tau_{abl}}{\sum_{l=1}^{p} \omega_{abl}} \tag{A.2.3}$$

   If variable l is numeric, then s_{abl} is measured as the absolute difference between values x_{al} and x_{bl}, normalised by the range of the variable R_l:

   $$\tau_{abl} = 1 - \frac{|x_{al} - x_{bl}|}{R_l} \tag{A.2.4}$$

   For factor variables, $\tau_{abl} = 0$ if the values are equal and 1 if different. The ω_{abl} term is 0 for variables l missing for a value, and 1 otherwise.

2. **Wishart**

Wishart (2003) noted that, despite the popularity of Gower's distance, the range standardisation may not be appropriate for continuous variables. Instead, the Wishart distance standardises numeric variables with the variance of that variable, σ_l^2 :

$$WIS(a,b) = \sqrt{\sum_{l=1}^{P} d_{abl}} \tag{A.2.5}$$

where

$$d_{abl} = \begin{cases} \left(\frac{x_{al}-x_{bl}}{\sigma_l}\right)^2 & \text{if } l \text{ is numeric} \\ 0 & \text{if } l \text{ is categorical and } x_{al} = x_{bl} \\ 1 & \text{if } l \text{ is categorical and } x_{al} \neq x_{bl} \end{cases} \tag{A.2.6}$$

3. **Podani**

Podani (1999) modified the Gower distance to account for ordinal variables and defined their measure as:

$$POD(a,b) = \sqrt{\sum_{l=1}^{P} \omega_{abl} \left(\frac{x_{al} - x_{bl}}{\delta_{abl}}\right)^2} \tag{A.2.7}$$

where $\omega_{abl} = 0$ if a value x_a or x_b is missing for variable l and 1 otherwise. For categorical variables, $x_{al} - x_{bl} = 0$ if the values are equal and 1 otherwise. The standardisation parameter takes on the values:

$$\delta_{abl} = \begin{cases} 1 & \text{if } l \text{ is categorical and } x_{al} = x_{bl} \\ \max x_{al} - \min x_{bl} & \text{if } l \text{ is continuous} \\ \max r_l - \min r_l & \text{if } l \text{ is ordinal} \end{cases} \tag{A.2.8}$$

where r_l are the numeric rankings of the ordinal groups.

4. **Huang**

In an attempt to extend the k-means algorithm to mixed data, Huang (1997) defined their similarity measure as:

$$HUA(a,b) = \sum_{r=1}^{P_n} (x_{ar} - x_{br})^2 + \gamma \sum_{s=1}^{P_c} (x_{as} - x_{bs}) \tag{A.2.9}$$

where P_n and P_c are the numbers of numeric and categorical variables, respectively. The weight term γ is used to control the importance of the categorical variables in determining similarity - if $\gamma = 0$, then the categorical variables would not be used at all. Instead, γ is chosen to be the mean variance of the numeric variables (Huang, 1997).

$$\gamma = \frac{\sum_{r=1}^{P_n} \sigma_r^2}{P_n} \tag{A.2.10}$$

As above, $(x_{as} - x_{bs}) = 0$ when $x_{as} = x_{bs}$ and 1 otherwise.

5. **Ahmad**

   Ahmad and Dey (2007) defined their distance measure too by separating numeric ($r \in P_n$) and categorical variables ($s \in P_c$):

$$AHM(a,b) = \sum_{r=1}^{P_n}(x_{ar} - x_{br})^2 + \sum_{s=1}^{P_c}\delta_c(x_{as} - x_{bs}) \tag{A.2.11}$$

   Ahmad and Dey (2007) define and illustrate $\delta_c(x_{as}, x_{bs})$ as the co-occurrence distance which measures how different values of categorical vary given the distributions of other variables.

6. **Harikumar**

   As with Huang, Harikumar and Surya (2015) treats categorical variables differently to numeric variables to account for variations in the distributions of the variable types (Bishnoi and Hooda, 2020). Harikumar and Surya (2015) applies the Manhattan distance to numeric variables ($r \in P_n$) which is more robust to outliers, the Hamming distance δ_b (Bookstein et al., 2002) to binary variables $t \in P_b$, and the co-occurrence distance to categorical variables ($s \in P_c$).

$$HAR(a,b) = \sum_{r=1}^{P_n}|x_{ar} - x_{br}| + \sum_{s=1}^{P_c}\delta_c(x_{as}, x_{bs}) + \sum_{t=1}^{P_b}\delta_b(x_{at}, x_{bt}) \tag{A.2.12}$$

   where $\delta_c(x_{as}, x_{bs})$ is the co-occurrence distance defined by Ahmad and Dey (2007).

A.3 Term Frequency - Inverse Document Frequency (TF-IDF)

In order to create $\boldsymbol{S}^{v:\not{s}}$, an item similarity matrix developed with unstructured data, specifically text, methods are required to represent the text-based elements of items in a structured manner.

Firstly, stop-words (commonly used words in a language, such as "a", "the", "is") can be removed and *stemming*, the reduction of words to their root forms (for example "computers" to "compute"), can be applied to simplify a body of text. Term Frequency - Inverse Document Frequency (TF-IDF) calculations can then be applied to the text to produce a structured vector space model of the document (Pazzani and Billsus, 2007).

With TF-IDF, a document (a body of text) is converted into a vector of weights, where each weight corresponds to the association between the document and a given term. To illustrate, if $D = \{d_1, d_2, \ldots, d_B\}$ is a set of documents under consideration and $T = \{t_1, t_2, \ldots, t_P\}$ is the set of all words (or terms) used in D, then each document d can be represented as a P-dimensional vector, $\boldsymbol{d} = \{\omega_1, \omega_2, \ldots, \omega_P\}$ (Lops, 2011). Each value ω_ϱ in this vector corresponds to the weight of word t_ϱ in document d which represents the importance and relevance of that term in the document (Pazzani and Billsus, 2007).

The weight of each term t_ϱ in a document d is a combination of the term frequency and inverse document frequency (Lops, 2011):

$$\text{TF-IDF}(t_\varrho, d) = \text{TF}(t_\varrho, d) \cdot \text{IDF}(t_\varrho) \tag{A.3.1}$$

The TF portion captures the frequency of that term in a given document:

$$\text{TF}(t_\varrho, d) = \frac{f_d(t_\varrho)}{\max f_d(t_z)} \tag{A.3.2}$$

where $f_d(t_\varrho)$ is the frequency of term t_ϱ in document d and $\max f_d(t_z)$ is the greatest frequency of all terms t_z appearing in document d. The IDF portion then captures how many documents contain a given term. The log is taken to act as a weight that decreases for frequently used words and increases for rarer words:

$$\text{IDF}(t_\varrho) = \log \frac{B}{n_{t_\varrho}} \tag{A.3.3}$$

where B is the number of documents in the set D, and n_{t_ϱ} is the number of documents that contain term t_ϱ. To ensure that longer documents are not preferred to shorter ones, the TF-IDF scores are cosine normalised to produce the final weights for each t_ϱ in d:

$$\omega_\varrho = \frac{\text{TF-IDF}(t_\varrho, d)}{\sqrt{\sum_{s=1}^{|T|} \text{TF-IDF}(t_s, d)^2}} \tag{A.3.4}$$

where ω_ϱ is the weight vector of all terms t_ϱ in document d and higher values correspond to a greater importance of that word in that document (Lops, 2011).

Finally, these weight values for each t_ϱ in all documents in D can be collected into their vector representations $\boldsymbol{d}$ and can be used to compare documents using any similarity function described in Appendix A.1. The similarity between all pairs of documents describing items can then be calculated to build the unstructured item similarity matrix $\boldsymbol{S}^{v:\not{s}}$.

Importantly, only the most significant words should be retained in a vector space model of free text. If the weights of all words in D are calculated for each document, not only will many values be 0 (as not all words occur in all documents), but the vectorised representations will be far too specific resulting in the over-fitting of models built using the weights (Aggarwal, 2016). Hence Pazzani and Billsus (1997) suggests that the number of terms per vector should be between 50 and 300. Selecting these most informative terms can be achieved through supervised feature selection methods which combine documents and associated ratings.

The Gini index measures how well a term can discriminate between rating values and is calculated as:

$$\text{Gini}(t_\varrho) = 1 - \sum_{h \in H} g_h(t_\varrho)^2 \tag{A.3.5}$$

where H is the set of possible rating values and $g_h(t_\varrho)$ is the fraction of documents awarded rating h that contain term t_ϱ. Small values for the Gini index indicate that the presence of that given word can discriminate strongly between ratings (Aggarwal, 2016). For example, if all documents containing term t_ϱ receive a rating of 5 ($g_5(t_\varrho) = 1$), then that word is maximally informative.

Although easily understood, the Gini index does not account for the ordering of the ratings h. An alternative option, the Normalised Deviation considers the rating distribution of documents containing a given word, calculated as:

$$\text{Dev}(t_\varrho) = \frac{|\mu^+(t_\varrho) - \mu^-(t_\varrho)|}{\sigma} \tag{A.3.6}$$

where $\mu^+(t_\varrho)$ is the mean rating of all documents containing the term t_ϱ, $\mu^-(t_\varrho)$ is the mean rating of all documents which do not contain t_ϱ, and σ is the standard deviation of the ratings of all documents. High $\text{Dev}(t_\varrho)$ values reflect a term's importance; if the difference between mean ratings of documents which do and do not contain a given word is large when compared to the standard deviation of all ratings, that word is said to discriminate well between ratings (Aggarwal, 2016).

A.4 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are a standard approach to training neural networks on non-numeric data such as images, audio and text. A *convolutional* layer is used to derive a feature map which represents the core features of the original input. That is, given an unstructured data format a, a numeric representation of the features of that input, $\boldsymbol{c}$, can be obtained by convolving the input with a set of *filters* F:

$$\boldsymbol{c} \sim F \cdot a \tag{A.4.1}$$

where the feature map $\boldsymbol{c}$ may then be propagated through a standard NN architecture as usual.

Although CNNs can be applied to an assortment of data types, focus will remain on text CNNs henceforth. A neural network may only accept numeric input data; in the case of images, pixel values are a direct numeric representation, but text is not as easily encoded.

Integer encoding, whereby each unique word is assigned a numeric value, can be used to convert text to a vector of numbers. Similarly, one-hot encoding may be used to convert text to a matrix of m rows representing each word used and n columns representing the entire vocabulary used. In one-hot encoded representations, a word is 1 (hot) if it is used in a given row and 0 otherwise. These methods are illustrated in Example A.4.1 below. One-hot encoding is problematic as in most settings with large vocabularies, the resulting matrix would be massive. Both methods, however, suffer from a lack of interpretability; the differences in magnitudes between integer representations and rows between binary representations are meaningless (Gu et al., 2018).

Example A.4.1.

$$\text{“you know who you are”} = \underbrace{\{1,2,3,1,4\}}_{\text{integer encoding}} = \underbrace{\begin{array}{c} \\ \text{you} \\ \text{know} \\ \text{who} \\ \text{you} \\ \text{are} \end{array} \begin{array}{c} \begin{array}{cccc} \text{you} & \text{know} & \text{who} & \text{are} \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array}}_{\text{one-hot encoding}}$$

Word Embeddings such as GloVe (Pennington et al., 2014) were developed to address this issue. Using machine learning techniques, Pennington et al. (2014) analysed massive volumes of text to develop numeric vectors that accurately represented each word in a vocabulary. The benefit of using word embeddings is that the numeric differences between word vectors reflect

meaningful semantic differences. For example, the difference between the word embeddings of "man" and "woman" should be similar to the difference between the embeddings for "brother" and "sister". Likewise, the embeddings for "bread" and "toast" should be similar (Pennington et al., 2014).

A document d can be converted to a word embedding matrix $\boldsymbol{D}_{\rho \times b}$ with each row consisting to the ρ words in d and q representing the dimension of the embedding vectors of the words. To extract the feature map from this numeric representation of text, filters are applied (convolved) to the rows of the matrix.

A filter $\boldsymbol{F}_{h \times b}$ is a matrix with a region size h and the same number of columns as the embedding vectors. Filters are used to extract the most prominent features of an input, by considering small subsections of the input iteratively. In the case of text, h specifies how many consecutive words are convolved with the filter. The output of the convolution of a filter and the input is calculated as:

$$\boldsymbol{o}_i = \boldsymbol{F} \cdot \boldsymbol{D}_{i:i+h-1} \tag{A.4.2}$$

where $\boldsymbol{D}_{i:i+h-1}$ specifies the h consecutive rows of $\boldsymbol{D}$ and $i \in \{1, \ldots, \rho - h + 1\}$.

Then, the feature map $\boldsymbol{c}$ is obtained by applying an activation function (such as ReLu) and a bias term:

$$\boldsymbol{c}_i = f(\boldsymbol{o}_i + bias) \tag{A.4.3}$$

Multiple filters may be applied to extract complementary features from the same regions of $\boldsymbol{D}$, resulting in numerous extracted features $\boldsymbol{c}_i^g$ for each filter g applied to consecutive rows of $\boldsymbol{D}_{i:i+h-1}$. This may lead to a large feature representation which should be reduced before proceeding to the next layers of the network (Zhang and Wallace, 2015).

The purpose of a *pooling* layer is to extract the most significant features from a feature map. Max pooling, selects the greatest features as the most representative, resulting in a reduced feature map:

$$\boldsymbol{c}_i = \max\left(\boldsymbol{c}_i^1, \ldots, \boldsymbol{c}_i^g\right) \tag{A.4.4}$$

The process of convolving and pooling may be repeated multiple times, with different filters of various sizes. Ultimately, the procedure computes a structured version of the original data. This introduces more complexity to the network as the values in each filter $\boldsymbol{F}$ are to be learnt as with the weights $\boldsymbol{w}$ of the remaining portion of the network (Liu, Li, et al., 2021).

A.5 Wine Terminology

Wine is a complex and highly variable drink consisting of multiple components (Goode, 2020), each of which is accompanied by plentiful amounts of wine-specific literature and scientific research. As a result, wine terminology has become less understandable to those who are inexperienced and can exclude those who do not drink from appreciating the intricacy of a glass of wine (Cruz et al., 2018). Hence, the following serves as an overview of wine to contextualise the terms frequently used throughout this research.

Wine is a drink produced from fermented fruit, most commonly grapes. Depending on the grapes used and production methods, four main *types* of wines may be produced: red, white,

rosé and sparkling. White wines are typically considered to be light (in flavour, not mass), delicate and refreshing, reds (red wines), however, are often richer and more intense than whites. Rosé wines may exhibit characteristics of red wines, but are more comparable to white wines and are identifiable by their pink colour. Sparkling wine is generally a carbonated form of white wine, although red wines may be carbonated too (Williams, 2013).

These wine types can be subdivided into wine *styles*; although many highly-specific styles exist, wines can be arranged into ten general styles: (1) white wines of a neutral 'wine' flavour, (2) fresh, floral and fruity whites, (3) aromatic whites with character, (4) sweet (dessert) white wines, (5) rosé, (6) plain reds of little flavour, (7) young reds not intended to mature, (8) mature reds of medium intensity, (9) concentrated, powerful and intense reds intended to mature, (10) wines fortified by the addition of alcohol (Johnson, 1991).

These wine styles are produced as a result of their *composition* and *vintage*. The composition of a wine describes the combination of grape *varieties* used in production. A grape's species (of which there are thousands), such as Chardonnay or Merlot, is known as its varietal. Wines composed of more than 85% of one grape variety are considered *single-varietal*, whereas wines that are composed of numerous grape types are known as *blends*. The vintage of the wine is the year in which the grapes for that bottle were harvested. Blends that contain wines from numerous vintages are described as *non-vintage* (Puckette and Hammack, 2018).

Beyond the variety of grapes used, the *terroir* of a wine has the largest impact on a wine's character. Terroir is a French term that describes the soil, topography and climate of the region in which the grapes were planted. Soil types dictate which types of grapes can be planted and impact the growth and health of the grapevines. The topography of a vineyard (rows of grapevines) affects how much sun, wind and water the grapes are exposed to. The climate of the region in which grapes are planted additionally influences how the grapes grow and ripen (Goode, 2020). Altogether, these factors impact the sugar levels and flavours within the grape (Mulligan and Gibson, 2016), which in turn influences the wines produced.

Wine has three components: its appearance, aroma and taste, all of which reveal unique elements of the drink. Firstly, the colour of the wine can indicate the age of a wine; deeper shades of yellow are found in older white wines, whereas the darker shades of red wine fade with age. Also, wines with red tints are typically more acidic than wines with purple hues. Furthermore, the viscosity (or thickness) of wine indicates the amount of alcohol and sugar in the wine; viscous wines which appear heavier are usually more alcoholic or sweet (Puckette and Hammack, 2018).

Next, the aromas of wine that are perceived due to chemical compounds that scent wine, can reveal much about the wine's style. Primary aromas are derived from the grapes used (Puckette and Hammack, 2018) and can be categorised into: citrus, orchard fruit (e.g. plum, peach), tropical fruit, greenery (e.g. grass, herbs), florals, red fruits (e.g. strawberry, cherry) and spices (e.g. cinnamon, ginger) (Williams, 2013). Secondary aromas, such as nuts, baked goods (e.g. butter, yeast, bread, biscuits) and minerals (e.g. salt, steel, wet stones) are derived from the fermentation process and are earthier by nature. Tertiary aromas are the result of the ageing process used and impart woody (e.g. oak, cedar, tobacco) scents (Williams, 2013).

The taste of a wine is its most complex component and consists of multiple traits: sweetness, acidity, tannins, alcohol, body, and finish. The *sweetness* of a wine is measured by its Residual

Sugar (RS) percentage - the amount of sugar that remains unfermented when bottling. Wines with less than 0.1% of residual sugar (such as French Malbec or Sauvignon Blanc) are considered 'very dry' or 'bone-dry', wines with less than 1% RS (such as Shiraz and Chardonnay) are considered 'dry', wines with an RS percentage between 1% and 3% (such as Chenin Blanc) are considered 'off-dry', wines with an RS percentage between 3% and 4% (such as Lambrusco or Moscato) are 'semi-sweet', 'sweet' wines (such as Port or Riesling) are those with an RS between 4% and 5%, and dessert wines with an RS percentage above 5% are 'very sweet' (Mulligan and Gibson, 2016).

The *acidity* of a wine is measured using the pH scale and is experienced as a tart, tingly, mouth-watering sensation on the tongue. Wines with a low pH and high acidity are described as being bright, zesty and sharp. If a wine is too acidic it may be described as sour or spicy. Conversely, wines with low acidity can appear dull and lifeless; hence, acidity is crucial in maintaining the balance and character of wine (Puckette and Hammack, 2018).

The *tannins* of wine cause a bitter, astringent, drying sensation in the mouth much like over-brewed tea. Polyphenols found in the skins and seeds of grapes produce a bitter sensation at the front of the mouth and those present in barrels used for fermentation cause an astringent sensation in the middle of the tongue. Tannins can be abrasive and unpleasant in younger red wines but 'soften' over time (Puckette and Hammack, 2018).

The *alcohol* of wine contributes a warm sensation and is measured as the percentage of ethanol (between 12% and 15% on average) in the wine. Alcohol is produced as grape sugars ferment (Mulligan and Gibson, 2016), hence, sweeter grapes can lead to more potentially alcoholic wines.

The *body* of a wine describes the extent to which a wine fills the mouth with flavour and acts as a measure of its boldness. 'Light' wines may feel thin like water, but 'full-bodied' wines are more intense, impactful and heavier on the tongue. Body is created through the combinations of the other taste traits; for example, higher levels of tannins, sweetness or alcohol all increase the body of the wine. Higher levels of acid or carbonation, however, decrease the body of the wine (Puckette and Hammack, 2018).

The *finish* of a wine describes the final flavours of a wine and the amount of time that the taste remains in the mouth. The finish of wine can indicate higher quality wines with many discernible layers of flavour (Puckette and Hammack, 2018).

All the components of wine must work harmoniously together to produce a well-*balanced* wine. This is in many ways subjective, however, as individual taste dictates preference overall. Some consumers may enjoy dry, tannic reds which may be considered unbalanced and unpleasant to others. Similarly, sweet, full-bodied whites may be sickly to some but preferred by others. Nevertheless, this highlights the joy in exploring wines - the infinite aroma and flavour possibilities presented by wines are what make them such beloved drinks worldwide.

A.6 Recommender System Case Studies

The following sections contain the details regarding the development of various wine recommender applications.

A.6.1 A knowledge engineering recommender system

Burke's (1999) application navigated a catalogue of wines iteratively to discover a wine most similar to the one specified by the user. Firstly, the user-specified a *source* wine - the wine for which the system would determine related items. The database of wines in the application would then be iteratively sorted to find the *target* wine which most closely matched the source (Burke, 1999b).

To discover the most similar wine, a *goal-based retrieval strategy* was applied. This strategy defined a list of attributes in the order of the priority that they should match the source wine. The goal ordering was: type, price, quality, flavour, body, finish, then drinkability (Burke, 1999b); this stipulated that the target wine should be most similar to the source according to type, then according to price, and so on. For each attribute, a proprietary similarity measure was defined to compare items based on that specific attribute. Wine attribute data was taken from Wine Spectator[2] and applied to the database of wines on VintageExchange.com (which has since closed).

The retrieval strategy was applied as follows: firstly, all N products were sorted into B buckets according to their similarity to the source type. For example, if the source was a Sauvignon Blanc, then the first bucket would contain all Sauvignon Blancs in the catalogue as they are perfectly similar according to type. The next bucket may contain all Chenin Blancs as they share a similarity of say 80% to Sauvignon Blanc as they are both direct descendants of the Traminer varietal (Mulligan and Gibson, 2016). If the system aimed to return n recommended wines to a user, where $n < N$, only the first b ($b < B$) buckets which contained n items, cumulatively, would be further processed. The buckets with the lowest similarities would be discarded as their items would never be considered for recommendation (Burke, 1999b).

The similarity measures for each of the remaining attributes were then applied until n buckets which each contained a single wine remained, or until all similarity measures had been applied. After initial tests, Burke (1999b) reported that the users of the system agreed that the target wines found in this manner were similar to the source wines. Further, the repeated sorting algorithm used had other benefits including the development of implicit user taste profiles which would describe their wine preferences and locating gaps in the catalogue where no wines could be recommended after sorting.

Through this implementation, Burke (1999b) found that the use of multiple, separate similarity measures for different attributes led to finer discrimination between wines. For example, the wine found to be most similar based on the measure used for price may be different to the most similar wine found using the quality measure. Hence, the use of different measures captured the non-uniformity of wine similarity.

However, this method proposed was limited too; applying the first three measures could produce a candidate list of thousands of wines, but after applying the fourth measure, no (or too few) items would be found. This highlighted an issue common in content-based recommendations namely, without sufficient engineering of item attributes, finding truly similar items based on a few attributes alone can be infeasible with items as nuanced as wine (Bobadilla et al., 2013). The converse scenario, where too many items are suitable for recommendation, also presented

[2] www.winespectator.com

a challenge - without additional data, the system was unable to select the most appropriate n items to recommend from a possible N candidates which all shared the same similarity.

Integration of user-based collaborative filtering

The original content-based approach could be initialised immediately as all that was required was a database of wine attributes and no user data was required. This limited the final predictions made since users' historical and individualistic preferences were never considered when making suggestions. Burke (1999a) later introduced a way to incorporate implicit collaborative filtering into the recommendations made.

To improve the accuracy and personalisation of recommendations using CF (Lü et al., 2012), user preference data needed to be derived as no data set on user wine preferences was available (Burke, 1999a). Burke determined that user ratings could be implied by their interaction with the system - the source wine could be considered positively rated (a 'like') as the user would likely search for wines similar to something they enjoyed previously. Then, if the user is presented with candidate items to choose from (after applying the similarity measures), items that were rejected would be considered negatively rated (a 'dislike') and the final item chosen, the target wine, would also be considered positively rated. These likes and dislikes could then be numerically encoded and used to calculate the correlations between users (Burke, 2000).

Using such implied ratings would only be feasible after long, widespread use of the system. As such data was unavailable, Burke (2000) tested his method on data collected from the "Entree" restaurant recommender system (Burke, 2000) which was developed using the same principles used in the Wasabi Personal Shopper. Data on user interaction with the Entree system was gathered over 3 years for a total of 20,000 sets of ratings. Each set of ratings contained the positively rated source and target restaurant and the ratings of any other candidate restaurants reviewed before the target was chosen. Half of these rating records were used for training and of the other half, 100 'active' users who had each rated at least 15 restaurants were used for testing. The system was tested by recording the number of times the system correctly recommended the true target restaurant when selecting a restaurant from the candidates (i) at random, (ii) based on the highest average rating and, (iii) based on the highest weighted average of the most correlated users determined by collaborative filtering.

The results showed a higher predictive accuracy when using collaborative filtering to discriminate between candidate items when compared to recommendations made at random or based on averages. The application of CF was limited by the data, but it was only applied after the sorting algorithm (i.e. after the content-based recommendations were generated). Thus, any potential inaccurate recommendations made by a collaborative filter supplied with insufficient data would have no worse impact than simply selecting the target item at random.

Burke (2000) concluded that adapting the recommendations made using wine content with collaborative filtering was appropriate in the context of finding wines similar to a source wine. As CF produces recommendations based on a user's past preferences, the user may become trapped by their past behaviour. If a user has only drank white wine in the past, they may never be recommended red wine when just using CF. Hence, Burke (2000) posited that any CF should be applied after content-based recommendations are generated to encourage diversity in the recommendations.

A.6.2 A clustering approach to content-based recommendations

Using supermarket transaction records gathered over 8 months, Almasi and Lee (1999) exploited user wine consumption to determine 4 clusters which each described a different wine preference profile. From these transaction records, 4000 users had bought more than 4 wines, and 3000 of those users had bought at least one of the top 10 most purchased wines. Those 3000 users were then clustered into 4 groups according to how frequently they purchased the top 10 most popular wines. Users were clustered according to the top 10 wines only to both increase cluster sharpness and simplify the process of assigning new users to a cluster (Almasi and Lee, 1999).

For a new user to obtain recommendations from Wine Guru, they would be requested to rate some or all of the top 10 wines (used to build the user clusters). A neural cluster scoring algorithm (Lawrence et al., 1999) was then applied to the ratings to establish which cluster the user would belong to.

The 3000 original users were split into clusters of sizes 630, 1110, 510, and 750. The first cluster captured users which purchased cheaper wines, the second cluster represented users with a strong preference for white over red, the third cluster was dominated by users who favoured Bulgarian to Chilean wine, and the fourth cluster was characterised by the users with an opposite preference to those in the third cluster and who favoured more expensive wines (Almasi and Lee, 1999).

Wine recommendations were informally presented through a personalised wine list. All 600 wines in the Wine Guru database were split into 10 clusters according to their type and intensity. Clusters 1 to 4 were for white wines, where cluster 1 had the lowest intensity (such as a Chenin Blanc (Puckette and Hammack, 2018)) and cluster 4 had the highest intensity (such as a wooded Chardonnay). Clusters 5 to 8 were for red wines ordered by increasing intensity and clusters 9 and 10 were for dessert wines.

A user's personalised wine list consisted of 40 wines constructed by selecting the top 4 most purchased wines in each of the 10 wine categories for that user's specific cluster. This can be considered a form of user-based collaborative filtering as a recommendation for a specific wine type was generated by selecting the most popular wines among a user's most similar 'peers' (i.e. their cluster).

The methods with which the system produced food-wine pairings can be considered a content-based approach, however. A database of meals was divided first into 15 categories such as fish, poultry, beef etc. These meal categories were further subdivided according to the main ingredient and preparation method. This resulted in the simplification of meals to 3 variable objects such as 'delicate fish broiled' or 'medium fish sauce'. Each combination of meal descriptors was assigned a weight m ($1 \leq m \leq 10$), where meals of weight 1 were considered the lightest types of meals.

To recommend a wine for a specific meal, the system would establish what weight, m, the proposed meal would correspond to. Then, the most popular wines in the user's cluster, u, for the wine cluster, w, which matched m (the weight of the meal) would be recommended as wine options. So, users in a cluster u would be recommended the most popular wines in wine cluster w_i, such that $i = m$, thereby matching the intensity of the wine with the meal type. For example, if a meal was assigned a weight of $m = 4$ and the user was a member of the third cluster, u_3, the wines recommended to the user would be the most purchased wines in the

fourth wine cluster, w_4, (heavier white wines) by users in u_3 (Almasi and Lee, 1999).

Although this system yielded good recommendations, it was restricted by its cluster design. Not only did the system require much data engineering to assign weights to meals, but meals would only be matched to the most popular wines in the corresponding cluster. This oversimplification prevented users from discovering more niche wines (which were purchased infrequently) or wines which did not fall within their taste profile; in this system, users in cluster u_i would rarely be recommended wines enjoyed by cluster u_j, $i \neq j$, as the collaborative filtering was conducted within clusters and not on the entire user base.

A.6.3 Networks used to produce food-wine recommendations

Ontologies are a formal way to represent knowledge on a specific domain in an explicit structure. An ontology presents a network of the various concepts and the links which connect them for a certain domain (Guarino et al., 2009). Both Michaelis et al. (2008) and Patton and McGuinness (2009) based their systems on the Stanford Knowledge Systems Laboratory (KSL) Wine Ontology (Noy and Mcguinness, 2001); with this ontology, the classes and attributes of a database of wines could be modelled into a formal knowledge base. Facets of that ontology captured the type, colour, grape, region, sugar level and flavour of the wines.

The first system, the "Tetherless World (TW) Wine Agent" (Michaelis et al., 2008), combined the KSL ontology with a 'wine wiki' which was built by collecting input from users. A wiki is a knowledge base that was created and continuously edited and managed by a set of users (Krötzsch et al., 2006). Presented with an input screen, users would contribute to the wiki by specifying their name, the food they ate, and the colour, body, flavour and sugar (sweet, semi-sweet, or dry) of the wine they drank.

The TW Wine agent then produced recommendations as follows: firstly, a hierarchy of foods (also modelled by an ontology) was presented to the user so that they could select the primary ingredient of their meal. Next, all n user recommendations which corresponded to the chosen food item were retrieved. Then, for each wine in the database, the properties of the wine (colour, body, flavour and sugar) would be compared to the corresponding wine wiki fields of the n recommendations. The wines were then ranked in order of the percentage of recommendations that matched their characteristics. For example, if wine x was represented by the characteristics {white, medium, moderate, dry} and 3 out of 4 recommendations for 'fish' matched those characteristics, then the wine would be recommended with a 75% endorsement for a fish meal.

The second system, "The Mobile Wine Agent" (Patton and McGuinness, 2009), was a further iteration of the TW agent with notable improvements to how recommendations were made. Users were able to access recommendations for meals based on a specific wine in addition to wine suggestions based on a meal. Next, the food ontology used to store the properties of various meal types was extended to capture flavours (such as bitter and sweet), thereby allowing users to filter and specify their food preferences more accurately.

Further, a user could indicate if they disagreed with a recommendation made, after which it would not be presented to the user again. Using the food ontology, meals at local restaurants could be extracted from their menus and modelled; this provided users with the ability to access wine recommendations by selecting meals at restaurants.

While these systems could use their knowledge bases to formulate recommendations, they did not exploit much other wine-related data, such as reviews or ratings, which could be used to create stronger suggestions.

A.6.4 WineRing

WineRing (Dillon et al., 2019) is able to make its suggestions as follows:

Firstly, the application records user ratings of some wines on a four-point scale: 'love', 'like', 'so-so', and 'dislike'. The properties of the rated wines are extracted and used to cluster the wines. Within the wine clusters, *rating dependency patterns* are established and used to build *user preference models*. These preference models represent the association between the ratings the user has awarded and the characteristics of the wines which drove those ratings. The user preference models can then be used to query the WineRing database to find and recommend wines that match the user's preferences (Dillon et al., 2019).

WineRing recognises that recommendations based on pure purchase and rating data will always be limited by the user's past behaviour. Hence, in addition to providing more diverse recommendations, the user preference models discover why a user-preferred a wine and can more richly assess if a user will truly enjoy a suggested wine.

A preference model is formally defined as a regression function which captures the dependency of wine ratings on wine traits (Dillon et al., 2019). Multiple separate preference models can be developed for a single user to represent their different tastes (for example, preferences in red wine will be modelled differently from white wine preferences). As more preference models are established for a user, the specificity, completeness and uniqueness of the recommendations can increase.

Recommendations made are the items predicted to receive a high rating according to the user preference model. A preference model is formed as follows:

Initially, the wines rated by the user are partitioned into hierarchical clusters. The clusters are formed according to the characteristics (comprised of traits and attributes) of the wine. Wine attributes are those characteristics that are not subject to user opinions such as origin, vintage and price, whereas wine traits are those which are observed by an expert. Taste and textural traits are represented in quantitative and ordinal scales (e.g. low body), flavour traits are represented by binary indicators (e.g. vanilla = 1, strawberry = 0), and wine styles are represented as percentages (e.g. California style Cabernet = 1, Napa-style Cabernet = 0.8).

The characteristics used to hierarchically cluster the wines are: composition, origin, fruit intensity, mineral intensity, body, acidity, oak intensity, tannin intensity, and flavours. Within each cluster, rating dependency patterns are then established. This involves identifying rating trends dependent on two or more wine characteristics within the wine subsets.

Analysis of the dependence patterns is conducted using ordered logit regression to accommodate the ordinal rating scale and the non-linearity of the rating dependency patterns. Multiple combinations of variables (wine characteristics) are explored to determine the most significant predictors of ratings. To reduce the search space of predictor variable combinations, multicollinear variables were not examined together; for example, body and alcohol intensity is not

used to locate dependency patterns as these variables are highly correlated. A typical dependency pattern may be that a user's ratings may increase with acidity levels and decrease with rising levels of fruit intensity.

Rating dependency patterns can be contrasting (ratings increase as values for one variable grow and the other decrease), focally contrasting (ratings are parabolic, indicating a preference for extreme values for variables, but not intermediate values), consistent (ratings are clustered around a single point) or polarising (consistent or contrasting ratings exhibited until a threshold, after which rating values are unpredictable).

Once the rating pattern and the corresponding governing wine characteristics have been identified, a (logit) model is fit to the data. In the case of regression models, a p-value threshold of 0.1 is set to ensure that there is at least a 90% chance that the model fit is significant. This p-value is then further interpreted as a measure of confidence in the predictions made using that specific model.

The preference model can then be considered to be the combination of parameters used in the rating dependency pattern, the characteristics used to cluster the subset of wines under consideration, and the optimal values for those characteristics which would lead to the greatest predicted rating (according to the rating dependency model).

Preference models are generalised and named but are unique to each user according to the model parameters used for a given user's rating patterns. A user may have many preference models to capture a diverse taste in wine; an example preference model which captures a user's preferences for fruit flavours, high acidity and wine style is:

> "***Marlborough Sauvignon Blanc***. *Intense aromas of passion fruit and lime with a herbal undertone. Unoaked, with high acidity and fresh flavours, moderate alcohol and a long finish.*" (Dillon et al., 2019)

Wine recommendations are then found by querying the database for wines that have characteristic values which match the optimal values within a preference model. Similarly, the characteristics of an unknown wine can be supplied to the preference model to calculate a predicted rating, which in turn would serve as a recommendation.

This method of producing recommendations is heavily dependent on a single user's rating history; as result, scarce, inconsistent and incomplete ratings negatively affect the results produced. To address this, a user's profile may be augmented by the preference models of other similar users. Unlike the methods used by other systems to compute the similarity between users, WineRing compares users based on their preference profiles.

For users which share preference models, the rating dependency patterns are compared. If two users share the same rating patterns for a subset of wines for a given preference model, the parameters of each user's rating model are compared to determine similarity. If two users have different rating patterns, similarity is computed on the predicted ratings produced by the models for a subset of wines.

Once similar users have been found, wines from different preference models (not part of the target user's profile) can then be recommended for the user to try and expand their experience of other wine types.

A.7 Visual Exploratory Data Analysis

For visual clarity, the following analysis does not include the outlier values, despite their presence in the data.

Each of the numeric wine variables are measured with different units and require scaling such that they can be appropriately compared to each other. The density plots in Figure A.1 provide an insight into the distributions of these variables; perhaps barring Tannins in Figure A.1e, each variable is quite skewed. Thus, when scaling the numeric variables, min-max normalisation (see Section 3.5 for more details) is chosen as the appropriate scaling method.

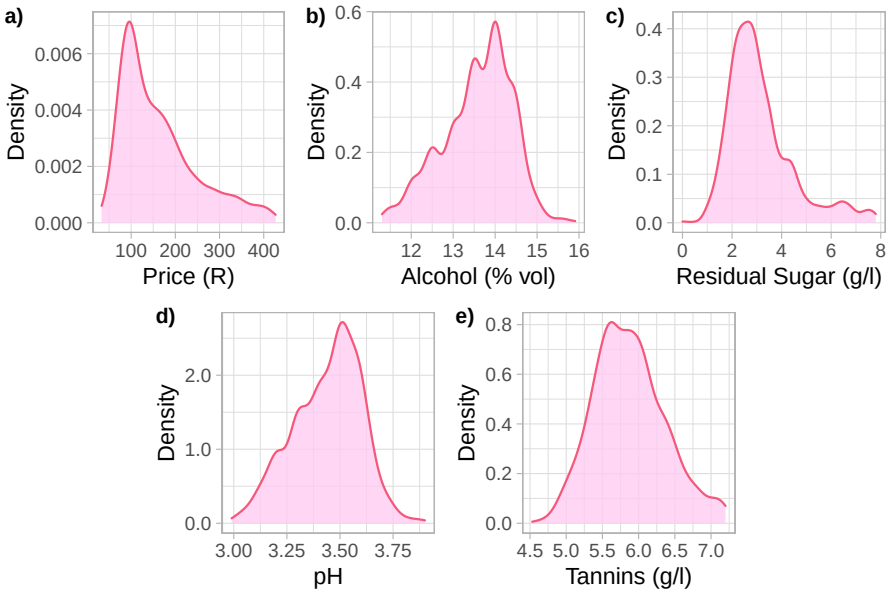

Figure A.1: Density plots of the numeric wine variables with outliers removed.

Figure A.2 contains violin plots of each numeric variable after scaling. Alcohol, pH and Tannins vary similarly with median values all close to 0.5. Residual sugar varies much less but, as in Table 3.3, contains many values on the higher end of the scale as a result of the dessert wines present in the data. Most wines have alcohol percentages on the lower end of the scale, with some wines (likely dessert and fortified wines) containing higher ABV values.

Next, Figure A.3 contains scatter plots of each pair of numeric variables with points coloured by wine type. These plots reveal that, except for Tannins and pH in Figure A.3j which are negatively correlated, and Alc and PH in Figure A.3f which are slightly positively correlated, the numeric variables do not share any strong linear relationships. This is important when differentiating between wines; for example, knowing that price bears no relationship to alcohol level in Figure A.3a indicates that if two wines share a similar price, it is not guaranteed that they will be as alcoholic. The lack of relationships between these variables highlights the importance of each of them individually - similarity determined through one numeric variable is not representative of similarity determined through another and thus all numeric variables should be considered together when calculating wine similarity.

Additionally, Figure A.3 indicates how the wines are often somewhat neatly clustered according to their type; price in combination with pH or TA in Figures A.3c) and A.3d) cleanly sepa-

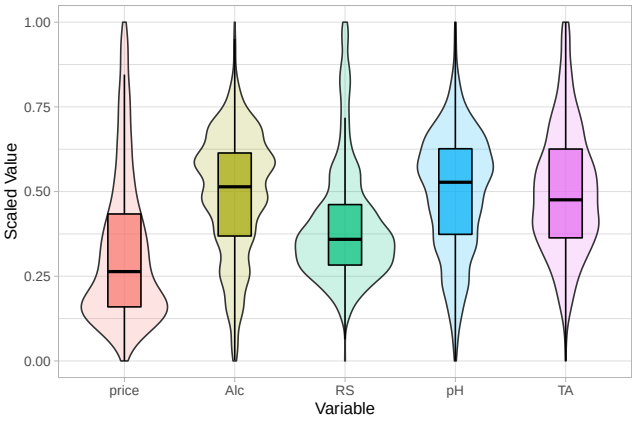

Figure A.2: Boxplots of the scaled numeric wine variables.

rate red from white wines as with RS and pH Figure A.3h). This highlights how the factor variables can be used along with numeric variables to further discriminate between wines when determining similarity.

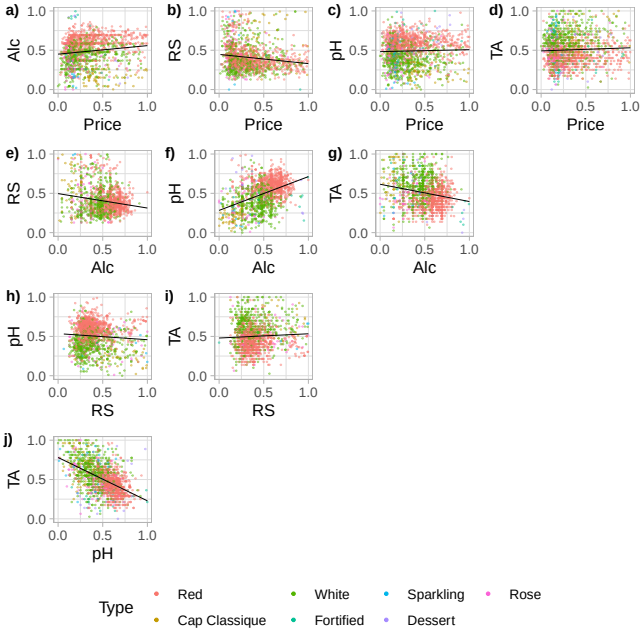

Figure A.3: Scatter plots of the scaled numeric wine variables with wine type indicated by point colour.

A.8 Variable Summary

Table A.1 summarises the variable notation used in the different RS methods explored throughout this thesis.

Table A.1: Summary of variables used throughout this thesis.

Variable	Description
$\boldsymbol{R}_{n\times m}$, r_{ij}	The sparse ratings matrix $\boldsymbol{R}$ consisting of n user rows, m item columns, populated with ratings r_{ij} made by user i on item j.
$\boldsymbol{R}^{+}$, $\boldsymbol{R}^{-}$	The set of observed and missing values in $\boldsymbol{R}$.
$\boldsymbol{R}_{i\cdot}$, $\boldsymbol{R}^{+}_{i\cdot}$, $\boldsymbol{R}^{-}_{i\cdot}$	The i^{th} user row of $\boldsymbol{R}$, the set of observed and the set of missing values in the i^{th} row of $\boldsymbol{R}$.
$\boldsymbol{R}_{\cdot j}$, $\boldsymbol{R}_{\cdot j}^{+}$, $\boldsymbol{R}_{\cdot j}^{-}$	The j^{th} item column of $\boldsymbol{R}$, the set of observed and the set of missing values in the j^{th} column of $\boldsymbol{R}$.
$\boldsymbol{R}^{CDF}$, $\boldsymbol{R}^{kNN}$	The ratings matrices imputed with the CDF and kNN protocols.
$\eta(\boldsymbol{R})$, $\zeta(\boldsymbol{R})$	The ratings matrix after normalisation and standardisation.
$\hat{\boldsymbol{R}}^{IBCF}$, $\hat{\boldsymbol{R}}^{s}$, $\hat{\boldsymbol{R}}^{\not s}$	The predicted ratings computed using IBCF, structured CB methods and unstructured CB methods.
U, V, U_{ab}, V_i	The set of all users, set of all items, set of users who co-rated items a and b and set of items rated by user i.
$\bar{r}_i$, $\bar{V}_i$	The mean rating made by user i and the set of all items rated by i with a rating $\geq \bar{r}_i$.
V_i^Q	The set of Q items recommended to user i
$\boldsymbol{A}_{m\times o}$, a_{ij}	The dense item attribute matrix consisting of m item rows, o distinct attributes columns, consisting of values a_{ij} that corresponds to the value of attribute j for item i.
$\boldsymbol{S}^{u}_{n\times n}$, $\boldsymbol{S}^{v}_{m\times m}$, $\boldsymbol{S}^{uv}_{n\times m}$	The user, item and user-item similarity matrices containing the similarity between all users, all items and all user-item combinations, respectively.
$\boldsymbol{S}^{v}$, $\boldsymbol{S}^{v:s}$, $\boldsymbol{S}^{v:\not s}$	An item similarity matrix developed with rating data (v), structured item data ($^{v:s}$) and unstructured item data ($^{v:\not s}$).
$\boldsymbol{S}^{v}_{\eta}$, $\boldsymbol{S}^{v}_{\zeta}$	The similarity matrices produced when using the normalised and standardised raring matrices $\boldsymbol{R}$.
$\boldsymbol{s}^{u}_{i} = \{s^{u}_{i1}, \ldots, s^{u}_{ik}\} \subseteq \boldsymbol{S}^{u}_{i\cdot}$	The user similarity vector that contains the user-based similarities between user i and the k most similar other users in the system.
$\boldsymbol{s}^{v}_{j} = \{s^{v}_{j1}, \ldots, s^{v}_{jk}\} \subseteq \boldsymbol{S}^{v}_{\cdot j}$	The item similarity vector that contains the item-based similarities between item j and the k most similar other items rated by i, where $\boldsymbol{S}^{v}$ may also be $\boldsymbol{S}^{v:s}$ or $\boldsymbol{S}^{v:\not s}$.

Variable	Description
$\dot{\boldsymbol{s}}_i^u$, $\dot{\boldsymbol{s}}_j^v$	The weighted versions of the vectors $\boldsymbol{s}_i^u$ and $\boldsymbol{s}_j^v$ that sum to 1.
$\boldsymbol{r}_j^u \subseteq \boldsymbol{R}_{\cdot j}^+$	The vector of ratings for item j awarded by each user in $\boldsymbol{s}_i^u$
$\boldsymbol{r}_i^v \subseteq \boldsymbol{R}_{i\cdot}^+$	The vector of ratings made by user i for each item in $\boldsymbol{s}_j^v$.
$\hat{r}_{ij}$, $\hat{r}_{ij}^u$, $\hat{r}_{ij}^v$	The model-based, user-based and item-based collaborative filtering predicted ratings for user i on item j.
$D = \{d_1, d_2, \ldots, d_B\}$, d	The set of B documents under consideration and a given document d.
$T = \{t_1, t_2, \ldots, t_P\}$, t_ϱ	The set of all P words (or terms) used in D and a given term t_ϱ.
$\boldsymbol{d} = \{\omega_1, \omega_2, \ldots, \omega_P\}$	An P-dimensional vector representing d where each value ω_ϱ corresponds to the weight of word t_ϱ in document d.
$\boldsymbol{x}_i$, $\boldsymbol{y}_j$	User and item profiles as determined through TF-IDF methods.
$\boldsymbol{U}_{n\times x}$, $\boldsymbol{V}_{m\times x}$	The user and item latent matrices of dimension x representing the underlying concepts $\boldsymbol{R}$.
$u_i\cdot \subseteq \boldsymbol{U}$, $v_j\cdot \subseteq \boldsymbol{V}$	The latent factors for user i and item j.
$\boldsymbol{w}$	The set of parameters to be optimised.
$\nabla \boldsymbol{j}$	The vector containing the gradient of the loss function for each parameter in $\boldsymbol{w}$.
α, λ	The learning rate used to control the descent of the loss function and regularisation parameter.
$e_{ij} = r_{ij} - \hat{r}_{ij}$	The error between the observed and predicted rating for user i and item j.
x_i, $z_j^{(l)}$, $z_j^{(l+1)}$	The input neurons x_i and the value at neuron z_j at layer (l) and $(l+1)$.
$w_{ij}^{(l)}$	The weights connecting an origin node i to destination node j at layer (k).
$\phi(z)$	The activation function applied to a neuron.
$\boldsymbol{c} \sim F \cdot a$	The feature map obtained after an input a produced after convolution with the set of filters F.
$\boldsymbol{D}_{\rho\times b}$	The embedded matrix representation of dimension b with each row corresponding to the ρ words in d.
$\boldsymbol{F}_{h\times b}$	A filter with a region size h and the same number of columns as the embedding vectors.
$\boldsymbol{o}_i$, $i \in \{1, \ldots, \rho - h + 1\}$	The output of the convolution of $\boldsymbol{F}$ and $\boldsymbol{D}$.

A.9 Results of Experimentation with a Baseline Neural Network

Figures A.4, A.5, A.6 show the training and validation RMSE produced by the baseline NN (detailed in Section 4.6.4.1) when all hyperparameters are fixed, barring the batch size, number of filters applied and embedding dimension, respectively. These plots are used to establish optimal values for the aforementioned hyperparameters as they are not fine-tuned using the grid search detailed in Section 4.6.4.2.

Note that the RMSE values in these plots are recorded when applying the baseline NN to the standardised CDF dataset. These results are presented before the predicted ratings produced by the NN are transformed back to the original rating scale and can thus not be compared to the results detailed in Section 5.

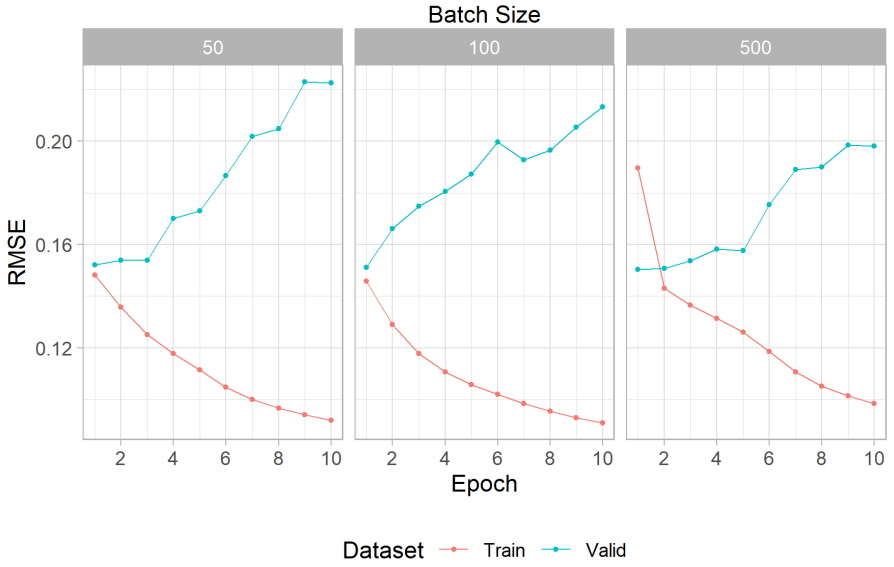

Figure A.4: Training and validation RMSE produced by the baseline NN for increasing batch sizes and epochs.

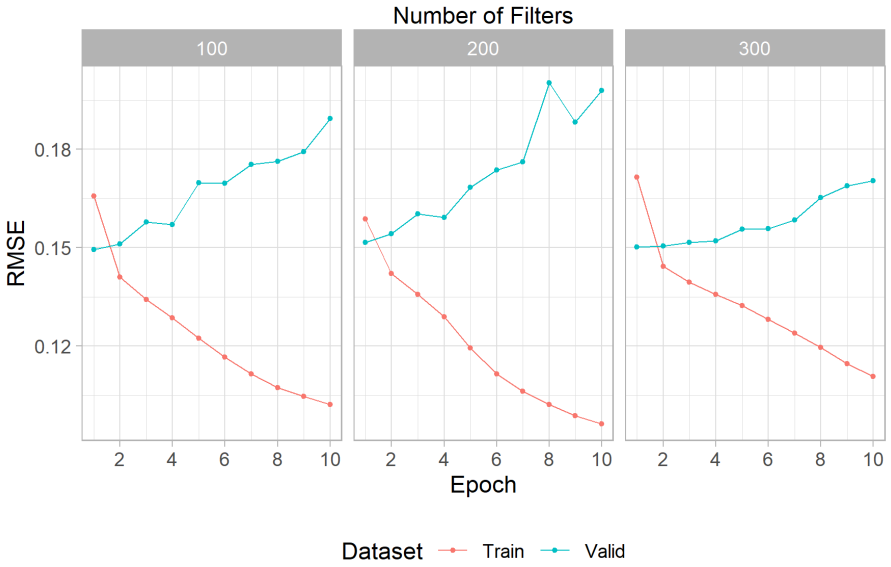

Figure A.5: Training and validation RMSE produced by the baseline NN for increasing number of filters and epochs.

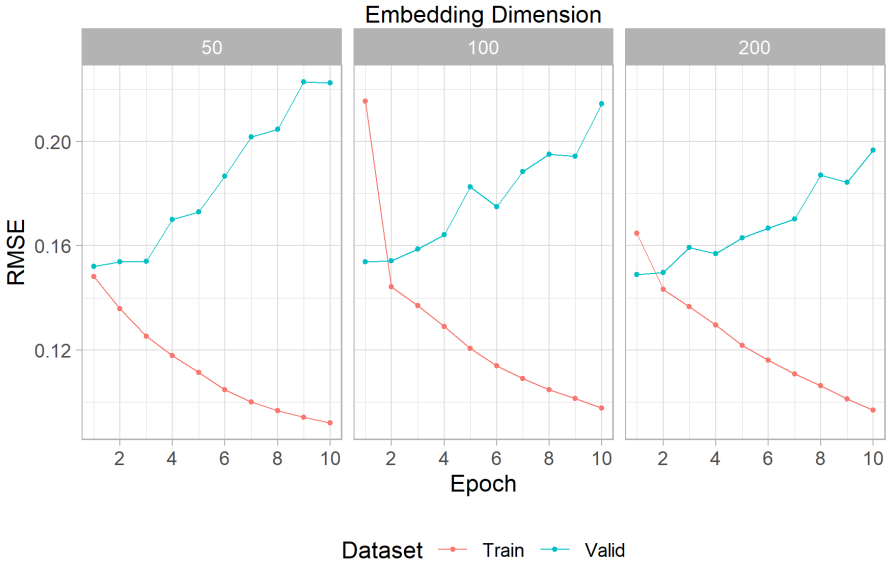

Figure A.6: Training and validation RMSE produced by the baseline NN for increasing embedding dimensions and epochs.

A.10 Results of Systems Using the CDF Imputation Protocol

Table A.2 summarises the optimal hyperparameter configurations for each method applied to the CDF dataset. Table A.3 then contains the evaluation results for those optimal systems developed in each method. Where applicable, this includes the training and validation RMSE of each system along with the quality metric results (see Section 4.2.2) measured for recommendation lists of $Q = 20$ wines. RMSE results presented in Table A.3 are computed using the original wine rating scale; the minimum RMSE across all methods, and the maximum results of each quality metric are highlighted in grey.

A.10.1 Item-Based Collaborative Filtering

Figure A.7a plots the validation RMSE yielded by different similarity measures for increasing values of k when used with Algorithm 4 applied to the unscaled, normalised (η) and standardised (ζ) CDF imputed dataset. Likewise, Figures A.7b-e plot the coverage (COV), personalisation (PER), ROC curve and average reciprocal hit rate (ARHR) for each similarity measure used with Algorithm 5 for recommended lists of increasing length Q computed with each version of scaling applied to $\boldsymbol{R}^{CDF}$.

A.10.2 Content-Based Methods

Figure A.8a plots the validation RMSE for the structured distance measures (applied to the categorical and numeric wine data) and the similarity measures (applied to the unstructured wine descriptions) for increasing numbers of similar wines k in Algorithm 4 applied to the CDF dataset. Figure A.8b-e plot the results of the quality metrics yielded by each distance measure and similarity measure applied to the structured and unstructured wine data for increasing numbers of recommendations Q when applied to $\boldsymbol{R}^{CDF}$.

A.10.3 Weighted Hybrids

Table A.2 summarises the β and θ weights determined to optimally linearly combine the top similarity matrices and predicted ratings matrices. The optimal similarity matrices are chosen as those which performed the best overall in each IBCF, structured and unstructured CB method (JAC, NHSM, HAR, PCC) as discussed in Sections 5.2 and 5.3. The optimal predicted matrices $\hat{\boldsymbol{R}}$, however, are chosen as those which produced the lowest validation RMSE.

Thus, using those similarity matrices to compute the predicted matrices $\hat{\boldsymbol{R}}$, the θ hybrids for the CDF and kNN datasets are:

$$\text{CDF: } \hat{\boldsymbol{R}}^{\text{optim}} = 0.8915\hat{\boldsymbol{R}}^{\text{IBCF}} + 0.0687\hat{\boldsymbol{R}}^{s} + 0.0235\hat{\boldsymbol{R}}^{\not s} \tag{A.10.1}$$

From Table A.3, the optimal CDF similarity matrices are computed using the optimised weights as:

$$\text{CDF: } \boldsymbol{S}^{\text{optim}} = 0.9542\boldsymbol{S}^{v} + 0.0162\boldsymbol{S}^{v:s} + 0.0162\boldsymbol{S}^{v:\not s} \tag{A.10.2}$$

The CDF β hybrid is formed much like the θ hybrid where the majority of the similarity data is taken from the IBCF approach and less than 2% is taken from each of the CB methods.

A.10.4 Matrix Factorisation

Figure A.9 plots the cross-validated RMSE and recommendation quality metric results for SGD matrix factorisation (Algorithm 1) applied to the unscaled, normalised and standardised CDF ratings matrix. Tables A.2 and A.3 contain the optimal hyperparameter configurations for the SGD and ALS algorithms and the results of those optimal models applied to the CDF data.

A.10.4.1 SGD factorisation

Figure A.9a plots the cross-validated RMSE of SGD matrix decomposition applied to the kNN-imputed (unscaled, normalised and standardised) training matrices for increasing numbers of dimensions and total regularisation. The default learning rate of 0.1 is found to be optimal and thus the grid search results of only $\alpha = 0.1$ are plotted. The top-Q recommendation quality metrics are more disappointing in general for the SGD approach. Compared to the IBCF and CB approaches, the SGD models produce recommendations which are less personal and cover less of the wine catalogue in Figures A.9b and A.9c. The results are worse still in Figures A.9d and A.9e as the number of relevant items recommended are far less than the IBCF method with ARHR values < 0.1.

A.10.4.2 ALS factorisation

Figure A.10a plots the RMSE results of the ALS matrix decomposition after transforming each recomposed matrix to the original rating scale for the CDF dataset. s. Coverage of the wine catalogue and personalisation of the top-Q lists in Figures A.10b-c are not terrible, but they pale in comparison to the recommendations produced using wine similarity. As with the SGD approach, worse results are observed for the ROC curve in A.10d and the ARHR plots in A.10e.

A.10.5 Neural Networks

As discussed in Section 4.6.1, the user and wine rating input to the neural networks is chosen as the optimal latent matrices in Section 5.5 and corresponds to the standardised SGD latent matrices which produced the lowest validation RMSE (see Tables A.2/ 5.1 for hyperparameter details).

Figures A.11 and A.12 plot the RMSE results of the numerous neural networks built using the exhaustive grid search of hyperparameter combinations in Table 4.2 and the CDF rating data.

From Figure A.11a, the RMSE values produced using the tanh final activation function are much more stable with far fewer validation RMSE outliers and a shorter spread of training RMSE values.

Figure A.11b plots the densities of the training and validation errors for models of different sizes and the tanh final activation function where the overall medians are indicated by the dashed lines. The model of size *sls* observed the lowest overall median RMSE of 0.616 and observed validation RMSE errors skewed largely towards lower values, it is chosen as the optimal model size configuration.

Next, Figure A.12a plots the validation RMSE for *sls* models with the tanh final activation function for various learning rates, amount of dropout, epochs and filter widths. From Figure A.12a, the highly regularised networks with dropout values of 0.3, 0.4, and 0.5 perform the least favourably, with high RMSE values on the first epoch that do not decrease steadily. While networks with dropout values of 0 may seem to produce slightly lower validation RMSE values, such networks with such little regularisation are expected to be poorly generalisable and likely overfit the training data. So, when examining the low RMSE values of the networks regularised with a dropout of 0.1 which are stable from the first epoch in A.12a, dropout of 0.1 is chosen optimal.

When examining the learning rates in Figure A.12a, it is clear that overall, the learning rate α of 0.002 led to the most stable decrease in rating prediction errors for all network configurations and is chosen as optimal. Figure A.12b then plots the optimal CDF NN model of size *sls*, the tanh final activation function, dropout of 0.1 and a learning rate of 0.002. As the model with the filter width $h = 4$ produced the lowest validation RMSE at the 10^{th} epoch, it is chosen as optimal.

This configuration is summarised in Table A.2. Using these optimal configurations, the network is trained for a total of 20 epochs (to ensure the stabilisation of the validation errors) and the performance of each is recorded in Table A.3.

Table A.2: Optimal hyperparameter configurations for methods applied to the CDF-imputed dataset.

Method	Input Data	Hyperparameter Configuration
IBCF	$\boldsymbol{R}^{CDF}$ and $\boldsymbol{S}^{v}$	$k = 10$, JAC similarity measure
Structured CB	$\boldsymbol{R}^{CDF}$ and $\boldsymbol{S}^{v:s}$	$k = 20$, HAR distance measure
Unstructured CB	$\boldsymbol{R}^{CDF}$ and $\boldsymbol{S}^{v:\not s}$	$k = 20$, PCC similarity measure
β Hybrid	$\boldsymbol{S}^{v}, \boldsymbol{S}^{v:s}, \boldsymbol{S}^{v:\not s}$	$\beta_1 = 0.9542, \beta_2 = 0.0162, \beta_3 = 0.0040$
θ Hybrid	$\hat{\boldsymbol{R}}^{\text{IBCF}}, \hat{\boldsymbol{R}}^{s}, \hat{\boldsymbol{R}}^{\not s}$	$\theta_1 = 0.8915, \theta_2 = 0.0687, \theta_3 = 0.0235$
SGD Factorisation	$\boldsymbol{R}^{CDF}$	Dimension = 30, $L1_U = 0.1$, $L2_U = 0.1$, $\alpha = 0.1$
	$\eta(\boldsymbol{R}^{CDF})$	Dimension = 15, $L1_U = 0.05$, $\alpha = 0.1$
	$\zeta(\boldsymbol{R}^{CDF})$	Dimension = 30, $L1_U = 0.01$, $L2_U = 0.01$, $L2_V = 0.01$, $\alpha = 0.1$
ALS Factorisation	$\boldsymbol{R}^{CDF}$	Dimension = 15, $\lambda = 0.3$
	$\eta(\boldsymbol{R}^{CDF})$	Dimension = 15, $\lambda = 0.05$
	$\zeta(\boldsymbol{R}^{CDF})$	Dimension = 15, $\lambda = 0.1$
Neural Network	$\eta(\boldsymbol{R}^{CDF})$	Final activation function = tanh, User Rating and Structured Wine hidden layer size = small, Wine Rating hidden layer size = large, $\alpha = 0.02$, Dropout = 0.1, $h = 4$

Table A.3: Evaluation metric results for the optimal systems built using various methods applied to the CDF-imputed dataset.

Method	RMSE		Measured at $Q = 20$				
	Train	Valid	COV	PER	TPR	FPR	ARHR
IBCF	-	0.7697	0.9896	0.8878	0.6948	0.0118	0.6188
Structured CB	-	0.7703	0.9970	0.9444	0.0720	0.0122	0.0319
Unstructured CB	-	0.7690	1.0000	0.9538	0.0076	0.0122	0.0019
β Hybrid	-	0.7646	0.9957	0.9032	0.6935	0.0188	0.6192
θ Hybrid	-	0.7578	-	-	-	-	-
SGD $\boldsymbol{R}^{CDF}$	0.1594	1.5200	0.8664	0.8797	0.1172	0.0121	0.0236
SGD $\eta(\boldsymbol{R}^{CDF})$	0.2015	1.2450	0.4799	0.8732	0.0904	0.0121	0.0212
SGD $\zeta(\boldsymbol{R}^{CDF})$	0.0794	0.7677	0.9665	0.6212	0.0213	0.0122	0.0039
ALS $\boldsymbol{R}^{CDF}$	0.4456	0.8036	0.6146	0.6301	0.0208	0.0122	0.0044
ALS $\eta(\boldsymbol{R}^{CDF})$	0.3381	0.7977	0.6726	0.597	0.0106	0.0122	0.0042
ALS $\zeta(\boldsymbol{R}^{CDF})$	0.2092	0.7583	0.9561	0.6525	0.0233	0.0122	0.0055
Neural Network	0.6020	0.5900	0.0122	0.0000	0.0120	0.0122	0.0098

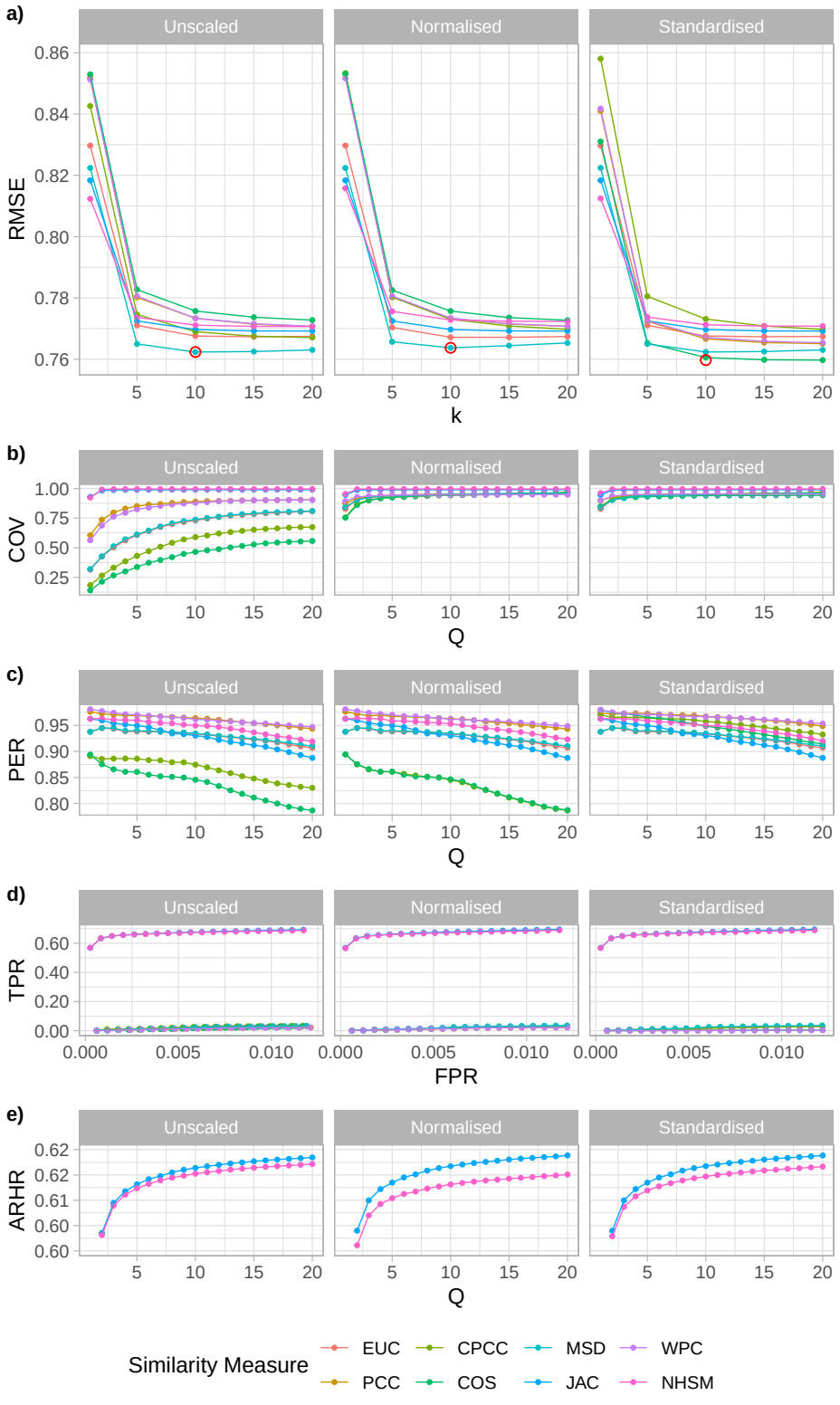

Figure A.7: RMSE and Quality results of IBCF applied to the unscaled, normalised and standardised CDF validation sets for various similarity measures and values of the k most similar items.

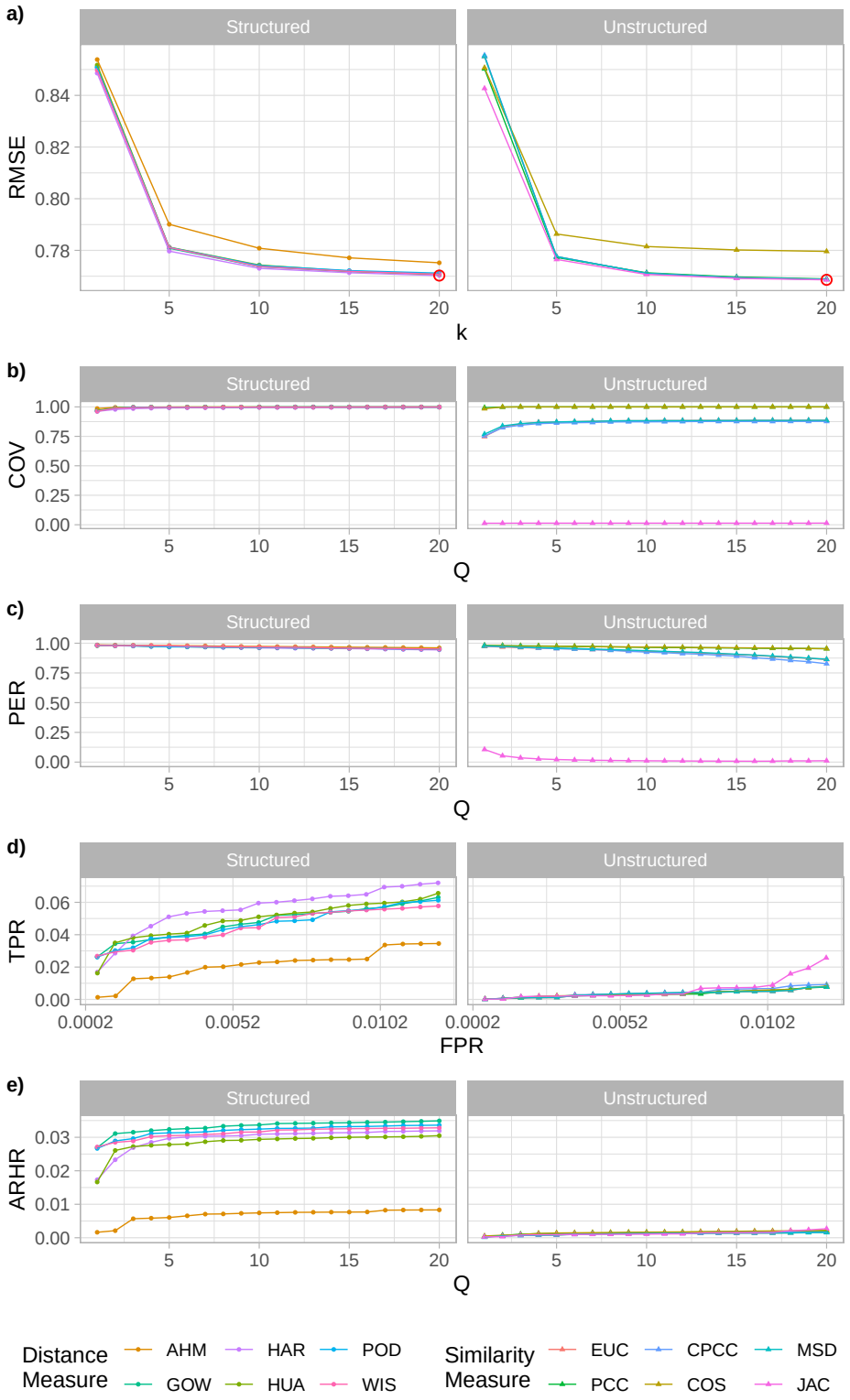

Figure A.8: RMSE and Quality results of structured and unstructured CB methods applied to the CDF data for various similarity and distance measures and values of the k most similar items.

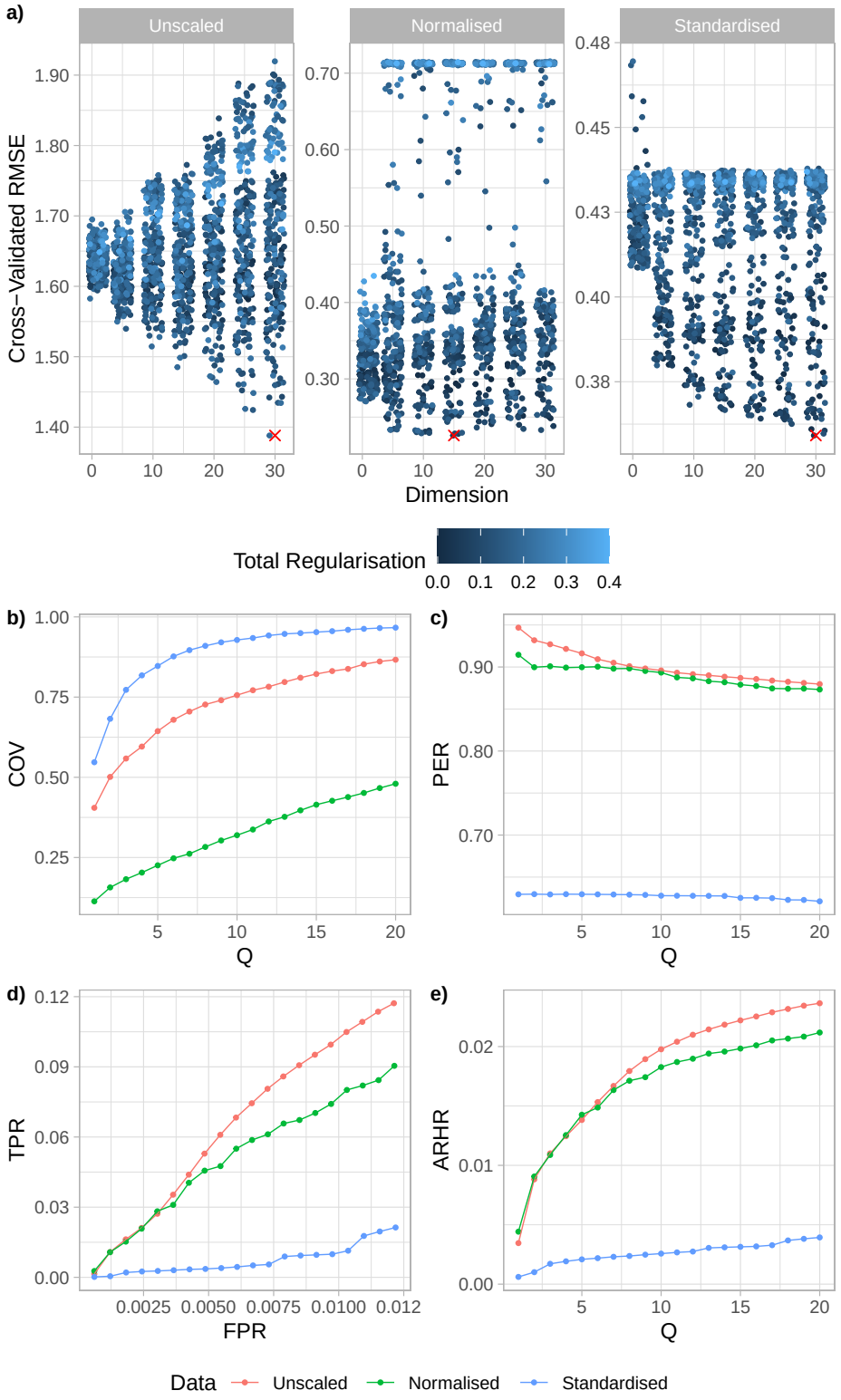

Figure A.9: Cross-validated RMSE and Quality results of SGD matrix decomposition applied to the unscaled, normalised and standardised CDF training sets for various numbers of dimensions x and amount of total regularisation.

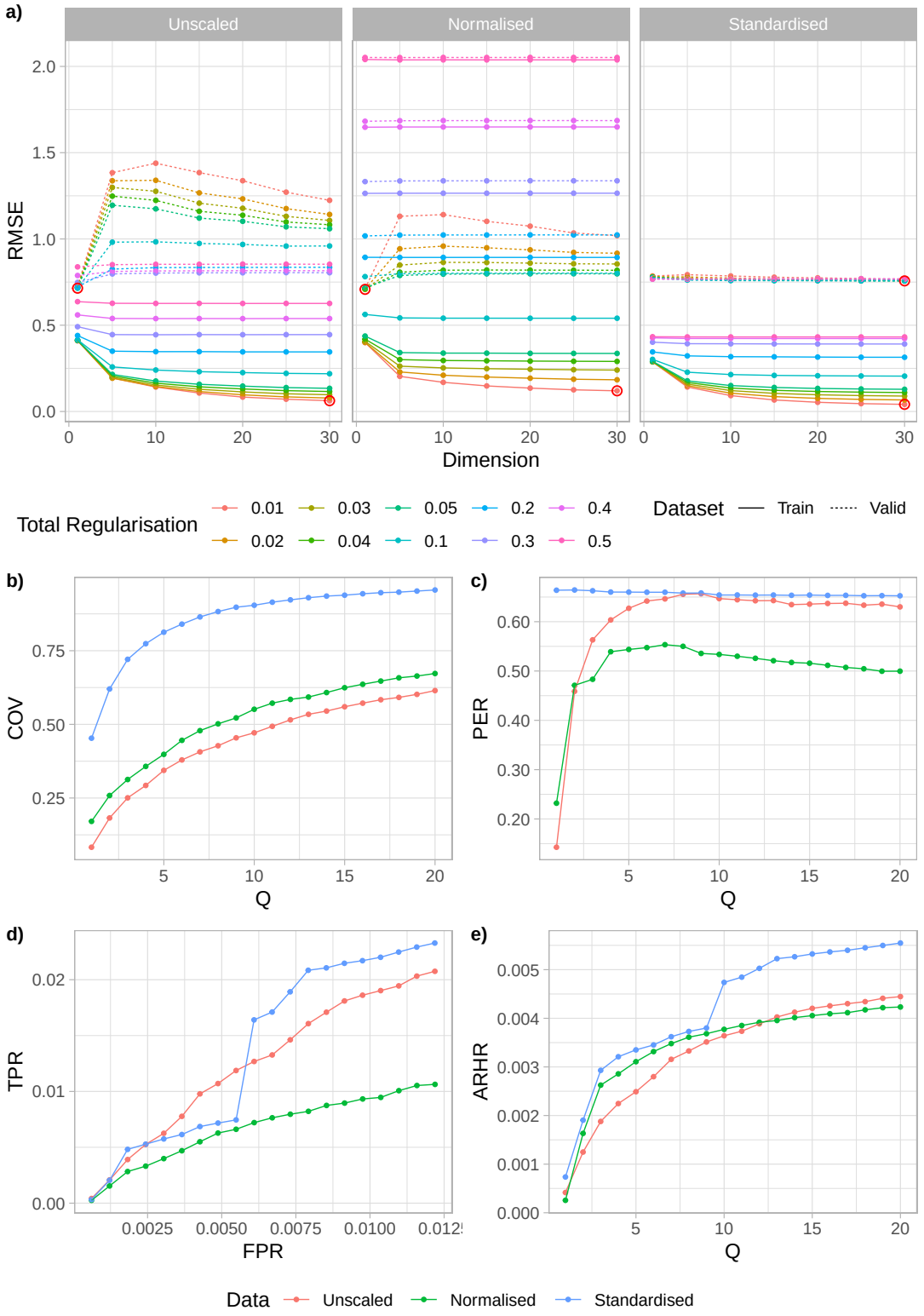

Figure A.10: RMSE and Quality results of ALS matrix decomposition applied to the unscaled, normalised and standardised CDF training and validation sets for various numbers of dimensions x and amount of total regularisation.

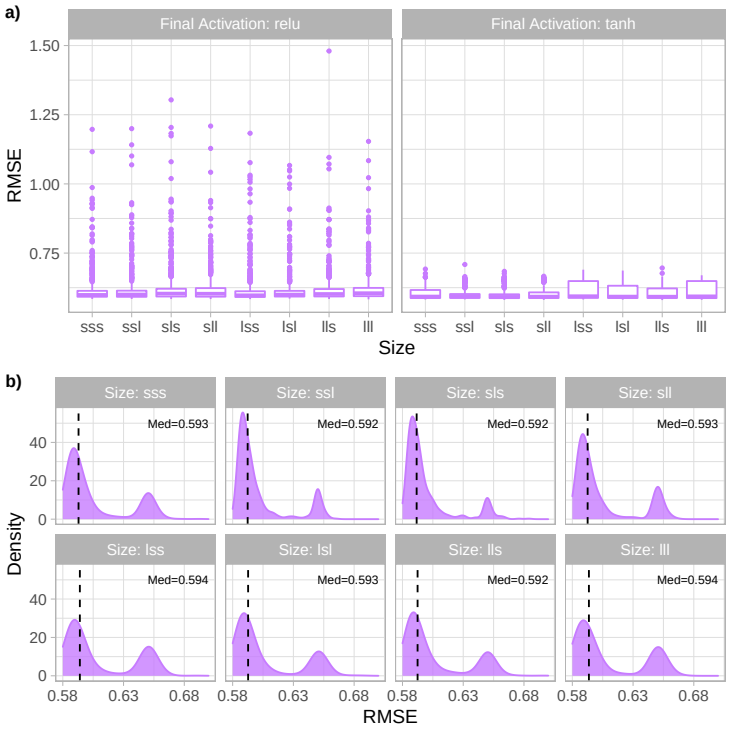

Figure A.11: Boxplots and density plots of the training and validation RMSE values yielded by NN models of different final activation functions and sizes applied to the CDF rating data.

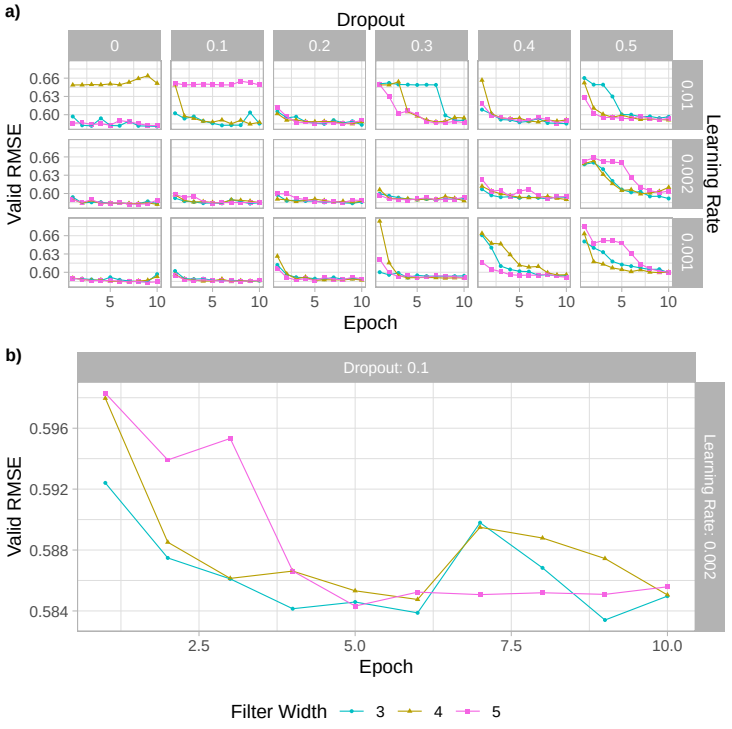

Figure A.12: Validation RMSE produced by the CDF NN model of size *sls* and tanh final activation function for increasing numbers of epochs, amount of dropout and learning rate applied.