BACHELOR'S DEGREE THESIS

Degree in Statistics
Degree in Economics

Title:

Allyn, A Recommender Assistant for Online Bookstores

Author: Laia Esquerrà Schaefer

Statistics Advisor: Esteban Vegas Lozano

Department: Genètica, Microbiologia i Estadística, secció d'Estadística

Economics Advisor: Dr. Salvador Torra Porras

Department: Econometria, Estadística i Economia Espanyola

Academic year: 2017-18

UNIVERSITAT DE BARCELONA

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

Universitat Politècnica de Catalunya

Universitat de Barcelona

Double Bachelor Degree in Statistics and Economics

Bachelor's Degree Thesis

Allyn: A Recommender Assistant for Online Bookstores

Laia Esquerrà Schaefer

June 2018

Statistics Advisor: Esteban Vegas Lozano

Economics Advisor: Dr. Salvador Torra Porras

Facultat de Matemàtiques i Estadística

Facultat d'Economia i Empresa

Abstract

Recommender Systems are information filtering engines used to estimate user preferences on items they have not seen: books, movies, restaurants or other things for which individuals have different tastes. Collaborative and Content-based Filtering have been the two popular memory-based methods to retrieve recommendations but these suffer from some limitations and might fail to provide effective recommendations. In this project we present several variations of Artificial Neural Networks, and in particular, of Autoencoders to generate model-based predictions for the users. We empirically show that a hybrid approach combining this model with other filtering engines provides a promising solution when compared to a standalone memory-based Collaborative Filtering Recommender. To wrap up the project, a chatbot connected to an e-commerce platform has been implemented so that, using Artificial Intelligence, it can retrieve recommendations to users.

Keywords: *Recommender systems*, *Collaborative filtering*, *Artificial neural networks*, *Denoising autoencoders*, *KerasR*, *Tensorflow*, *E-commerce*, *Information retrieval*, *Chatbots*

AMS Classification (MSC2010): 62M45 Neural nets and related approaches
68T05 Learning and adaptive systems

Resum

Els Sistemes de Recomanació són motors de filtratge de la informació que permeten estimar les preferències dels usuaris sobre ítems que no coneixen a priori. Aquests poden ser des de llibres o películes fins a restaurants o qualsevol altre element en el qual els usuaris puguin presentar gustos diferenciats. El present projecte es centra en la recomanació de llibres.

Es comença a parlar dels Sistemes de Recomanació al voltant de 1990 però és durant la darrera dècada amb el *boom* de la informació i les dades massives que comencen a tenir major repercussió. Tradicionalment, els mètodes utilitzats en aquests sistemes eren dos: el Filtratge Col · laboratiu i el Filtratge basat en Contingut. Tanmateix, ambdós són mètodes basats en memòria, fet que suposa diverses limitacions que poden arribar a portar a no propocionar recomanacions de manera eficient o precisa.

En aquest projecte es presenten diverses variacions de Xarxes Neuronals Artificials per a generar prediccions basades en models. En concret, es desenvolupen *Autoencoders*, una estructura particular d'aquestes que es caracteritza per tenir la mateixa entrada i sortida. D'aquesta manera, els *Autoencoders* aprenen a descobrir els patrons subjacents en dades molt esparses. Tots aquests models s'implementen utilitzant dos marcs de programació: Keras i Tensorflow per a `R`.

Es mostra empíricament que un enfocament híbrid que combina aquests models amb altres motors de filtratge proporciona una solució prometedora en comparació amb un recomanador que utilitza exclusivament Filtratge Col · laboratiu.

D'altra banda, s'analitzen els sistemes de recomanació des d'un punt de vista econòmic, emfatitzant especialment el seu impacte en empreses de comerç electrònic. S'analitzen els sistemes de recomanació desenvolupats per quatre empreses pioneres del sector així com les tecnologies *front-end* en què s'implementen. En concret, s'analitza el seu ús en *chatbots*, programes informàtics de missatgeria instantània que, a través de la Intel · ligència Artificial simulen la conversa humana.

Per tancar el projecte, es desenvolupa un chatbot propi implementat en una aplicació de missatgeria instantània i connectat a una empresa de comerç electrònic, capaç de donar recomanacions als usuaris fent ús del sistema de recomanació híbrid dut a terme.

Paraules clau: *Sistemes de recomanació*, *Filtratge Col · laboratiu*, *Xarxes Neuronals Artificials*, *Denoising autoencoders*, *KerasR*, *Tensorflow*, *Comerç electrònic*, *Recuperació d'informació*, *Chatbots*

Classificació AMS (MSC2010): 62M45 Neural nets and related approaches
68T05 Learning and adaptive systems

Acknowledgements

First, I want to thank my statistics advisor, Esteban Vegas, for all the time spent giving me useful advice, for encouraging me to overcome the difficulties I encountered during the realization of this work and for all the work he has put to work with me with the never ending changes there have been since we first started to talk about it last year.

Second, my economics advisor, Dr. Salvador Torra, for the help and ressources provided not only for the economic analysis but also for the statistics part.

About a year ago I was introduced to Recommender Systems in a short summer course. It was most definitely a fascinating topic for me and one thing became clear: I would do my Bachelor's Thesis dedicated to it. Therefore I also want to thank Bartek for introducing this topic in his course although he might not remember me and not even be aware of what has come out of it.

Furthermore, I want to thank Ernest Benedito, for introducing me to chatbots and for always being open to collaborate, offering his support throughout the project.

Finally, I would like to thank my family and close friends that have offered their support during the realization of this work and the completion of the Degree, with the special mention of Sandra and Marc, who have been following the progress of it and offered their help in many occasions.

Contents

List of Figures

List of Tables

List of Abbreviations

AE	Autoencoder
AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
BX	Book-Crossing
CB	Content-based
CF	Collaborative Filtering
CNN	Convolutional Neural Network
DNN	Deep Neural Network
DL	Deep Learning
GLM	Generalized Linear Model
KB	Knowledge-based
LP	Long Polling
MAE	Mean Absolute Error
ML	Machine Learning
MLP	Multilayer Perceptron
NCAE	Neural Collaborative Autoencoder
NL	Natural Language
NLP	Natural Language Processing
NLU	Natural Language Understanding
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
RS	Recommender System
UBCF	User-based Collaborative Filtering

Notation

General

v	scalar value
$\boldsymbol{v}$	row vector
$\boldsymbol{v}^T$	column vector
$\boldsymbol{M}$	matrix
$\sigma_{\boldsymbol{x}}$	standard deviation of $\boldsymbol{x}$
Σ	covariance matrix of $\boldsymbol{x}$ and $\boldsymbol{y}$
Φ	Cumulative Distribution Function (CDF) of the standard normal distribution

Sizes

d	total number of input nodes
L	number of layers in the network
m	number of examples in the dataset
$n^{[l]}$	number of hidden units (nodes) of the l^{th} layer
n_y	output size (or number of classes)

Objects

$\boldsymbol{X} \in \mathbb{R}^{d \times m}$	input matrix
$\boldsymbol{x}^{(i)} \in \mathbb{R}^d$	i^{th} example represented as a vector, $i = 1...m$
$x_j \in \mathbb{R}^1$	j^{th} input node, $j = 1...d$
$\boldsymbol{Y} \in \mathbb{R}^{n_y \times m}$	label (output) matrix
$y^{(i)} \in \mathbb{R}^{n_y}$	output label for the i^{th} example, $i = 1...m$. It can also be a vector, $\boldsymbol{y}^{(i)}$, when $n_y > 1$

$\hat{y}^{(i)} \in \mathbb{R}^{n_y}$	predicted output value for the i^{th} example, $i = 1...m$. It can also be a vector, $\boldsymbol{y}^{(i)}$, when $n_y > 1$
$y_j \in \mathbb{R}^1$	j^{th} output node, $j = 1...n^y$
$\boldsymbol{a}^{[l]}$	activated value of the hidden units of the l^{th} layer
$\boldsymbol{b}^{[l]}$	bias vector of the l^{th} layer
$\boldsymbol{W}^{[l]}$	weight matrix of the l^{th} layer
$\boldsymbol{z}^{[l]}$	input value of the hidden units of the l^{th} layer

Functions

$\sigma(\boldsymbol{z})$	activation function
$L(\hat{\boldsymbol{y}}, \boldsymbol{y})$	loss function
$J(\boldsymbol{W}, \boldsymbol{b})$	cost function

Introduction

> *"People read around ten MB worth of material a day, hear 400MB a day, and see one MB of information every second"* - The Economist (November 30, 2006)

That was in 2006. Almost 12 years after, this volume has increased by a factor of 2^8 according to Moore's Law: 5GB worth of material read and 100GB heard a day; 512MB of information seen every second. We are talking about massive volumes of data, which have become impossible to process by humans.

The Paradox of Choice defines how having too many options is actually counterproductive. Furthermore, this data overload not only affects users, but also companies who have faced difficulties to extract relevant information from it.

Through Machine Learning and Recommender Systems, we can overcome these. Both research fields have been developing high performance technologies and algorithms to process massive data which have given rise to automated information filtering engines.

Nevertheless, most of these algorithms face several problems; the most relevant being *data sparsity*. This means that when trying to provide suggestions to a user for example, only a few other users might have common elements in their past review history. Without common information, measuring similarities and providing suggestions for items of use becomes a hard task. Hybridization methods combining several techniques try to deal with these deficiencies.

This Thesis is developed with the objective of getting a better insight on Recommender Systems and provide an end-to-end solution that integrates an algorithm or an hybridized combination able to compute accurate rating predictions.

Aim of the Project

The main goal of this project is to get deeper insights on Recommender Systems using Artificial Neural Networks, two areas in Statistics which haven't been seen in depth during the Bachelor's Degree. As part of this project, we develop a detailed guide to these study areas.

Furthermore, we want to provide an end-to-end solution, where recommendations can be retrieved by any user at any time. These kind of solutions are quite infrequent in the field of Recommender Systems, as companies who have developed them focus on the integration of these algorithms in their websites. Therefore, we develop an open real-time solution which opens the door for possible future works, a *Chatbot* (also refered as Bot).

Through the development of this Thesis, we pretend to accomplish the following goals:

- Provide a general overview on the current state of **Recommender Systems** and their penetration in **e-commerce platforms**.
- Define the most common **data structure** used to develop Recommender Systems, and specially the **challenges** it represents for traditional Statistics.
- Present a general overview on **Artificial Neural Networks**, how they relate to Statistics and their **role** in Recommender Systems.
- Implement a **Neural Recommender System** able to provide accurate predictions for new users.
- Introduce the concept of **Chatbot** and its role as a real-time Recommender Assistant.
- Implement the Chatbot in a **front-end application** to make the system reachable for potential users.

Scope

In order to limit the scope of this project to the magnitude of the Bachelor's Thesis, the knowledge domain is contrained to online bookstores, and more specifically to the recommendation of Adult Books. The focus is set on retrieving accurate predictions without letting the front-end solution aside, which is presented as a simple Proof of Concept where we test its potential functionalities.

Report Outline

This report consists of nine chapters additionally to this Introduction where the project is introduced. Chapters are split into several parts:

- **Methodology**
  - **Chapter 1: Recommender Systems**. An introduction to Recommender Systems is provided, detailing data sources and recommendation techniques used.
  - **Chapter 2: Artificial Neural Networks**. We provide deep insights on this set of techniques, building the algorithm from the ground up.
  - **Chapter 3: Chatbots**. We expose a brief historical introduction to Chatbots and define their main building blocks. A state of the art analysis on the use of Chatbots for Recommender Systems is provided
- **Part I: An Economic Analysis**
  - **Chapter 4: State of the Art Recommender Systems**. We define present and potential applications of Recommender Systems and discuss their impact measures.
  - **Chapter 5: Actual Use Cases**. The Recommender Systems implemented by four specific companies are detailed.
- **Part II: Implementation**
  - **Chapter 6: The Dataset**. We introduce the set of data available to train the system.
  - **Chapter 7: Back-End Development**. The experiments developed in order to define the model are explained and the strucutre of the Chatbot back-end is detailed, as well as its functionalities.
  - **Chapter 8: Front-End Implementation**. We explain the development of the front-end for the Chatbot as well as the app channel selected to implement it.
- **Conclusions**
  - **Chapter 9: Conclusions and Future Work**. We conclude the project evaluating the final solution presented, stating the contributions and proposing future work to do.
- **Appendices**. Two appendices have been provided, the first with the complete auxiliary data tables and figures and, a second with the R code used.

Methodology

Chapter 1

Recommender Systems

Recommender Systems (RS) are information filtering engines used to estimate users' preferences on items they have not seen. They serve to guide users in a personalized way which allows them to discover these new products or services they might be interested in and possibly hadn't even notices. In order to make recommendations, individual interests and preferences are indirectly revealed. As software agents, RS present the potential to support and improve consumers' decisions in their online product selections, Xiao and Benbasat (2007).

Thus, RS are designed to help customers by introducing products or services as a friend or expert would have done in the past, generally recommending items to the users according to their purchase history or past ratings. Usually, a RS recommends items by either predicting ratings or providing a ranked list of items for each user.

In this sense, mainly 3 types of recommendation tasks exist based on their output: **rating prediction**, **ranking prediction** (top-n items) and **classification**. Each user gets different recommendations according to their profile. Therefore, we need a user model.

Lists of recommended products are usually done in one of two ways: **collaborative filtering** or **content-based filtering**. Hybrid methods exist which combine both approaches. This project will focus on these latter ones.

When it comes to designing RS there are three different paradigms that need to be taken into account: data used, information provided by the user and domain features.

In the following section we'll focuse on the data used, while new user's will provide information through the chatbot structure, where domain features will also need to be specified.

1.1 Data Sources

As information processing systems, implemented RS actively gather data (Ricci, Rokach, and Shapira 2011) from their website source. But experimental RS can also be built on a static database. Data is both about items and users. Which data is actually exploited can vary according to the recommendation technique to be applied.

In this section, the main data structure needed to implement the RS built will be presented. It is mainly composed of ratings and user and items descriptions as well as external constraints.

1.1.1 Data Structure

Information gathered by a RS can be split in three main datasets: items, users and ratings (also called transactions).

Items

Items are the recommended objects, which could also be people (as suggested connections in LinkedIn). Thus, items can be defined by many different characteristics sush as complexity, value or utility. According to the core technology of a RS, they can be using a range of properties and features of the items.

Users

Also users of the RS, can have different goals and/or characteristics. To personalize both recommendations and interactions with users, RS exploit a range of information about these users. Also in this case, information can be structured in various ways and the selection of what information is modeled depends on the recommendation technique. This model will profile user preferences and needs.

Ratings

In a more general approach we generically refer to a transaction as a recorded interaction between a user and the RS. Transactions store important information generated during this interaction, which is useful for the recommendation algorithm implemented.

Ratings are actually the most popular form of transaction data collected by RS. This collection can be done in two different ways, either implicitly or explicitly. **Explicit ratings** are asked to the user, i.e. an opinion on the item is provided on a rating scale.

On the other hand, **implicit ratings** reveal that the user has consumed that item but didn't give a rating to it.

If an interactive process is supported by the RS, this model can be more refined as user requests and system actions are alternated. That is, when a user requests a recommendation, the system has two options: produce a list of suggestions or ask the user to provide additional preferences in order to provide better results.

Formal Problem Definition

Given a list of M users $U = u_1, u_2, ..., u_M$, a list of N items $I = i_1, i_2, ..., i_N$ and a list of items, I_{u_i}, which have been rated by user u_i; the recommendation task consists in finding, for a particular user u, the new item $i \in I \setminus I_{u_i}$ for which u is most likely to be interested in, Ricci, Rokach, and Shapira (2011).

Ratings u_i can take a set S of possible values, which can be a numerical scale (e.g., $S = [1, 10]$) or dichotomic ($S = \{like, dislike\}$).

We will generally suppose that no more than one rating can be done by a $\{user, item\}$ pair. Furthermore, I_{uv} denotes a set of items which have been rated by two users u and v, i.e. $I_u \cap I_v$ and U_{ij} a set of users which have rated two items i and j.

1.1.2 Data Enrichment Techniques

Publicly avaliable datasets do not always offer all the information one is expecting to find. On one hand, avaliable data might not come in the most suitable form for analysis. On the other side, we might be missing relevant information for our approach.

Variable engineering can help us transform variables to overcome the first problem. The most simple example is the transformation from a factor variable with N levels, to $N - 1$ dummy variables. For our model, we'll need to reshape geospatial data, we'll see later the details on how to do so.

Missing information has nowadays also a solution: search the web for the desired additional details. Doing this process manually can be very time consuming or even impossible when working on large datasets. Therefore, we will make a short introduction to automated data scraping.

1.1.2.1 Geospatial Data

When defining preferences, cultural aspects can be highly relevant. In this sense, visualizing preferences on a map for example, can help us define groups that wouldn't be so obvious on a factorial plane.

Geospatial data can come in several different forms such as numerical coordinates, state codes or textual city names. Each type of information requires a different approach. Numerical coordinates for example, can be easily geolocated on a map using `R`'s `ggmap` package.

Other data formats might need some previous preprocessing to get them in a standard format. The `R` package `countrycode` aloows us to do so. `countrycode` translates long country names or coding schemes into another scheme like the official short English country name. It also creates new variables with the name of the continent or region to which each country belongs.

This package is not case-sensitive which allows it to understand multiple inputs which aren't in a specific format. It is always better to normalize names before though, using functions like `tolower` or regular expressions to supress special characters.

1.1.2.2 Data Scraping

Data scraping is a set of techniques used to get data in an unstructured format (HTML tags) from a website and transform it to a scructured format which can be easily used. The goal is to look for and extract the desired information, and aggregate it. What they all have in common is that the engine is looking for a certain kind of information which previously predetermined.

According to Vargiu and Urru (2012) extracted information can be about types of events, entities or relationships from textual data. This information has several different uses, from seach engines to news feeds or dictionaries. One of its potential applications is to scrape item's rating data to create recommendation engines.

Almost all programming languages used provide functions that perform web scraping, although approaches can be very different. There are several ways of scraping data from the web, Kaushik (2017), some of which are:

- **Human Copy-Paste**: It is a slow but efficient way of scraping data from the web. It refers to humans analyzing and copying the data to local storage themselves.
- **Text pattern matching**: Using regular expression matching facilities of program-

ming languages is another simple yet powerful approach to extract information from the web. Regular expressions are supported in 'R'.

- **API Interface**: Many websites like Facebook, Twitter or LinkedIn provide their own public and/or private API which can be called using standard code to retrieve data in the prescribed format.
- **DOM Parsing**: Using web browsers, programs can also retrieve dynamic content or parse web pages. From their DOM tree programs can retrieve parts of these pages.

Other scraping techniques include HTTP programming or HTML parsers. Text mining on the other hand, is used to extract patterns and relevant information in tasks which require discovering new and previously unkown data. When relying on text mining, relevant information such as keywords or document-term frequencies are extracted through linguistic and statistic algorithms. It is used for example to select relevant news articles whose existance is previously unkown.

1.1.3 Data Preprocessing

1.1.3.1 Similarity Measures

The most popular technique used in RS is Collaborative Filtering (CF) and particularly the use of a kNN classifier which will be described in the next section. What is now interesting about it is how the distance or similarity among users is defined.

Based on Ricci, Rokach, and Shapira (2011), we will present five main measures: **Euclidean distance** (E), **Minkowski distance** (M), **Mahalanobis distance** (H), **cosine similarity** (C) and **Pearson correlation** (P).

Euclidean distance

It is the simplest and most common distance which is associated to a straight line between two points.

$$d_E(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_{k=1}^{n} (x_k - y_k)^2} \tag{1.1}$$

where n is the number of rows or tuples of the data and x_k and y_k are the k^{th} attributes or components of data objects x and y, respectively.

Minkowski distance

This is a generalization of the previously presented, Euclidean distance.

$$d_M(\boldsymbol{x}, \boldsymbol{y}) = \left(\sum_{k=1}^{n} |x_k - y_k|^r \right)^{\frac{1}{r}} \tag{1.2}$$

where r is the degree of the distance. As we can see, it corresponds to the Euclidean distance when $r = 2$. Furthermore, this distance has specific names for several values of r, among others:

- **City block, Manhattan or L1 norm** when $r = 1$
- **Supremum, L_{max} norm or L_∞ norm** when $r \to \infty$

Mahalanobis distance

The Mahalanobis distance is defined as:

$$d_H(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{(\boldsymbol{x} - \boldsymbol{y})\Sigma^{-1}(\boldsymbol{x} - \boldsymbol{y})^T} \tag{1.3}$$

where Σ is the covariance matrix of the data.

Cosine similarity

This measure considers items as document vectors of an n-dimensional space and compute their similarity as the cosine of the angle that they form:

$$d_C(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{||\boldsymbol{x}|| \cdot ||\boldsymbol{y}||} \tag{1.4}$$

where $\cdot$ indicates vector dot product and $||\boldsymbol{x}||$ is the norm of vector $\boldsymbol{x}$. This similarity is known as the *cosine similarity* or the *L2 Norm* .

Pearson correlation

Lastly, similarity can also be given by the correlation among items which measures the linear relationship. Pearson correlation is the most common measure for that.

$$d_P(\boldsymbol{x}, \boldsymbol{y}) = \frac{\Sigma}{\sigma_{\boldsymbol{x}} \cdot \sigma_{\boldsymbol{y}}} \tag{1.5}$$

where Σ indicates the covariance matrix between $\boldsymbol{x}$ and $\boldsymbol{y}$, and σ their standard deviation.

Traditionally, in RS, either the cosine similarity or the Pearson correlation have been used. The latter one is the default setting for the `R recommenderlab` user-based collaborative filtering (UBCF) algorithm.

1.2 Recommendation Techniques

RS can be classified into different categories according to the technique used to make the recommendations. These can have their focus on users, items or ratings itselves, as well as in any combination of them. Here we present the 4 main recommendation techniques and particular cases that will be of our interest later.

1.2.1 Collaborative Filtering

CF is the most popular and well-known technique to build RS. It follows a very simple idea, which is that users tend to buy items preferred by users with similar tastes Adomavicius and Tuzhilin (2005). This similarity is calculated based on the past rating history of users, but it could also be implemented to define similarities over items.

The easiest way to apply it is using neighbour-based methods like kNN which are simple and efficient, while producing accurate and personalized recommendations.

Algorithm Types

According to Breese, Heckerman, and Kadie (1998), algorithms for collaborative recommendations can be grouped into two general classes: *memory-based* (or *heuristic-based*) and *model-based*

1. **Memory-based**: in this case, recommendations or predictions are made based on similarity values and past user-item ratings are directly used to predict ratings for new items. Predictions can be done both as a user-based or an item-based recommendation. Commonly used techniques include:
    - Neighbour-based CF
    - Cosine-based Similarity
    - Clustering

Advantages

These can be wrapped up in **simplicity**: these algorithms are simple to implement and easily understood; **justifiability**: computed predictions can be intuitively justified; **efficiency**: they require no costly training although recommendations can be more expensive to compute and; **stability**: new data can be easly handled without having to retrain the system and only similarities regarding the new item need to be computed.

Disadvantages

It main problem is that sparse data and common ratings give unreliable and not accurate recommendations.

2. **Model-based**: in contrast to memory-based algorithms, machine learning or data mining models are used in this case to find complex rating patterns in training rating data which is then used to predict ratings. Some commonly used techniques are:

   - Matrix Factorization
   - Bayesian Networks
   - Clustering
   - Artificial Neural Networks

Advantages

They can achieve valuable predictions even when working with small data about each user and they can deal with wide range of content, recommending all kinds of items, even the ones that are different to those seen in the past.

Disadvantages

As in the previous case, and in any RS, the number of ratings is generally small, which can make models suffer from **sparsity**. Additionally, model-based techniques present **scalability** limitations when dealing with **new users** or **new items** as models might need to be trained again and won't be able to recommend new products until there are enough ratings about it.

1.2.2 Content-based Filtering

Content-based (CB) filtering are based on a similar idea to CF, but in this case, similarity is defined by the intrinsic characteristics of a user or item. In this sense, the algorithm will recommend items that are similar to the ones the user has liked in the past. Typically, this content refers to items but some new approaches also define user profiles. CB RS use different resources, such as item information or user profiles, to learn latent factors that define associated features.

According to Felfernig and Burke (2008), the task is to learn a specific classification rule for each user on the basis of the user's rating information and the attributes of each item so that items can be classified as likely to be interesting or not.

When textual ratings are avaliable, exploration of ratings and its reviews allow more accurate rating predictions since they can more specifically define the sentiment or define outstanding/lacking product features.

Advantages

Three main advantages can be highlighted:

- **Independence**: these algorithms depend only on the ratings of the active user or item, thus, the volume of data loaded is smaller.
- **Justifiability**: recommendations can be easily explained by listing content features or descriptions that caused an item to be recommended.
- **New items**: because an item-content-based (user-content-based) recommender has access to item (user) features (e.g., keywords or categories/genere or age), it does not suffer from the new item (user) problem: new items (users) look just like old ones.

Disadvantages

Main disadvantages include:

- **Scalability**: the new user (item) problem remains in item-content-based (user-content-based) since users (items) must build up a sufficiently rich profile through the addition of multiple ratings
- **Limited content analysis**: most studies are based on lexical similarity (bag-of-words), thus, missing semantic meaning. Additionally, there is a natural limit in the number and type of features that are associated.

- **Over-specialization**: they are not built to find unexpected recommendations in the sense that they do not offer to the user substantially different products, limiting variability. This limitation is also called the *serendipity* problem.

Demographic Recommendation Technique

This is a particular case of a user-content-based algorithm which will focus on user's demographic features such as age, gender or country to make item recommendations. The assumption is that different recommendations should be generated for different demographic niches.

The reason it is pointed out is that avaliable user data includes age and location of the user making the recommendation, which is intented to be used in addition to a CF approach.

1.2.3 Knowledge-based Filtering

Knowledge-based (KB) RS recommend products based on specific domain knowledge on how certain item features satisfy users' needs and specifications. These algorithms rely on knowledge sources other than those previously discussed which can be divided on two main aspects: user requirements and domain knowledge.

According to Felfernig and Burke (2008), there are two well-known approaches to knowledge-based recommendation: case-based recommendation and constraint-based recommendation.

In the first, the system will try to discover what the user has in mind and find a suitable product for it. This requires domain-specific knowledge and considerations which will are not accessible in this case.

On the other hand, constraint-based recommendations take into account explicitly defined constraints, which are specially relevant for the present case and will be explained in more detail.

Constrained-based Recommendations

There are two main types of contraints that can be applied to a problem: filters or incompatibility. In the case of a Book RS, the user might be looking for a specific genere or author and items which do not correspond to this specification should not be considered at all.

Yet, if there is no item that really fits this wished or the calculated rating is negative (in the sense of dislike), other mechanisms to fulfill requirements as much as possible with a minimal set of changes are usually implemented.

The interaction with a KB RS is usually set up as a dialog (or conversational recommender) where users can specify their requirements in the form of answers to questions. This is particularly interesting for the present case, as the final model is implemented on a chatbot.

This process can be explicitly modeled through finite selection options (as have been implemented) or be enriched with natural language interaction.

The main **advantage** of KB systems is that they usually work better at the beginning but might be easily surpassed if they do not provided with learning components.

1.2.4 Hybrid Recommender Systems

A Hybrid RS combines two or more of the techniques listed above. This systems try to fix the disadvantages of one algorithm by takind advantage of another algorithm able to overcome them, improving overall performance. There are several ways to combine basic RS techniques in order to create a hybrid system.

According to Burke (2002), seven different hybridization techniques can be defined:

- **Weighted**: A linear combination of predictions from different recommendation techniques is computed to get the final recommendation.
- **Switching**: Using a switching criteria, the system switches between different recommendation techniques.
- **Mixed**: A list of results from all recommendations derived from applying various techniques are presented as a unified list without applying any computations to combine the results.
- **Feature Combination**: Results from the collaborative technique are used as another feature to build a content-based system over the augmented feature set.
- **Cascade**: Multistage technique that combines the results from different recommendation techniques in a prioritized manner.
- **Feature Augmentation**: Another technique that runs in multiple stages such that the rating or classification from an initial stage is used as an additional feature in

the subsequent stages.

- **Meta-level**: Model generated from a recommendation technique acts as an input to the next recommendation technique in the following stage

Many hybrid systems have been proposed in the literature, such as Ge et al. (2011), Schein et al. (2002) and Gunawardana and Meek (2009). Moreover, papers from Balabanovic and Shoham (1997), Melville, Mooney, and Nagarajan (2002) and Pazzani (1999), compare empirical performance of hybrid and pure collaborative and content-based methods and demonstrate that the hybrid methods can provide more accurate recommendations than pure approaches.

Chapter 2

Artificial Neural Networks

CF has been widely used in order to recommend new contents to users. However, it presents a relevant limitation because missing ratings difficult the computation of similarities between users or items. This lack of data can represent up to 99%, which brings us to look for a different approach which can potentially overcome the sparsity problem.

Machine Learning (ML) is a field of computer science which tries to build computer systems that automatically improve with experience (Mitchell 2006), combining statistical models and Artificial Intelligence (AI) to build them. These systems aren't always able to identify the whole process, but they are still able to build useful aproximations, (Alpaydın 2010). This approximation, accounts for part of the data, which in traditional statistics is called the explained variance.

The niche of ML is to find relevant patterns in the data without having to previously establish a formal equation to modelize the data. Therefore, ML offers higher flexibility when it comes to compute non-linear relationships. Additionally, there are several reasons why these algorithms are increasingly gaining popularity, among others:

- Parameter optimization through complex optimization algorithms huge amounts of parameters can be tuned.
- Countinuous improvement in systems that can learn over time and update themselves to the optimal setting in different conditions.
- Automation of tasks by supplying a machine with a learned algorithm, it can develop tasks on its own, which reduces human error problems but this can also have a drawback.

Nevertheless, ML algorithms also have some drawbacks which mainly concern time constraints and error correction. The large amount of data required is not always avaliable

and, when it is, it might not have the expected quality. Errors in data can make the algorithm learn a skewed pattern and when an error is made, diagnosing and correcting it can be highly difficult. When these immediately detected, operations could run off before human intervention allows the identification of the error and its source.

Along this chapter we will present a specific ML algorithm able to learn patterns and use this knowledge for missing imputation in very sparse data: **Artificial Neural Networks** (ANN).

Machine Learning and Statistics

ML is closely related to statistics, and more specifically with computational statistics. The main difference among the two lies on the learning approach. While computational statistics use the computational power of machines to solve large predefined problems, ML traditionally presents two different learning paradigms: **supervised learning** and **unsupervised learning**

In supervised learning, the correct values are provided during training, and the task map inputs to these values by minimizing errors in predictions. This corresponds to the traditional statistics approach. On the other hand, in unsupervised learning only the input data is known, and the task is to find patterns inside it. This, can also be done through Multivariate Analysis.

A third learning paradigm has stood out in the last few years: **reinforcement learning**. It stands in between the other two, although it is sometimes presented as part of supervised learning. Reinforcement learning is used for sequential decisions which lead to a final state, where a single action is not important, but the final output is. Therefore, the algorithm is not evaluated step by step (unsupervised), instead, it gets positive or negative reinforcements, which can be defined by a cost function, provided it finds the best solution to a problem with the least possible mistakes (supervised). This processes can be compared to Markov Decision Chains, offering a solution for finite horizon problems.

Thus, in supervised learning, ML and statistics overlap, but both approaches are rather complementary than contradictory, although terminology generally differs. Table 2.1 offers a small comparison on statistical terms and its denomination in the ML field.

According to Alpaydın (2010), the different applications of ML can be classified in three main tasks: **learning associations**, **classification** and **regression**. These have the same main structure as in statistic but present some differences regarding its approaches:

- **Learning Associations**. In many cases, we will be interested in finding *association rules* we didn't know existed. This is equivalent to learning a conditional probability, *P(Y|X)*, over the entire dataset instead of in variable pairs.

- **Classification**. We might be interested in assigning an individual observation in one of the classes. In ML though, observations are not as simple as in statistics. Inputs can be for example images, and the task could be *optical character recognition*, *face recognition* or *object detection*, but it could also be sound.

- **Regression**. In this sense, data might present very complex underlying patterns, which don't have a straightforward modelization. Using ML algorithms it can be approximated.

Statistics	**Machine Learning**
Classification, Regression,...	Supervised Learning
Classifier	Hypothesis
Clustering	Non-supervised Learning
Coefficients	Weights
Estimation	Learning
Explanatory Variable	Input
Goodness of Fit criterion	Cost Function
Individual	Instance
Model	Artificial Neural Network, Decision Tree,...
Response Variable	Output/Target
Variable	Attribute

Table 2.1: Statistics vs. Machine Learning Terms

ANN are used in many pattern classification and pattern recognition applications, Cheng and Titterington (1994), which can range from speech recognition, to object detection or process optimizations. These can be applied to almost every field, including routing and transportation (e.g. autonomous driving, radar localization...) and medicin (e.g. identification of cancerous cells). But we also have to consider that there are several structures of ANN, which serve very different purposes, also in regression. Therefore, in this section, a deeper introduction to ANN will be made, going over all of their elements and the different types of structures that can be built, in order to select the most suitable one for the present problem.

2.1 Precedents

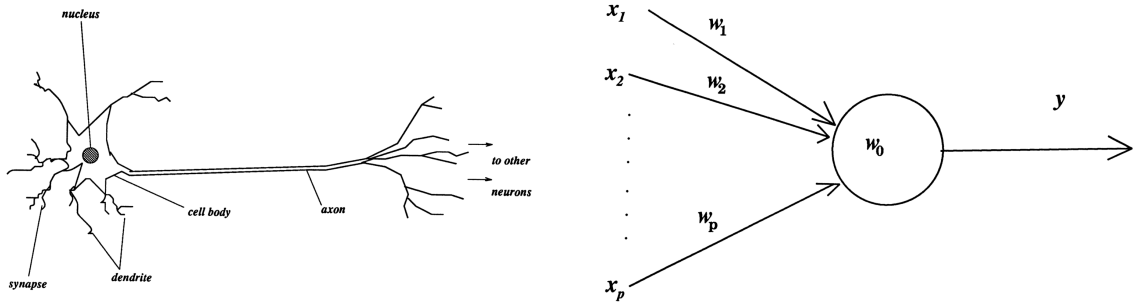
AI. A field that has rapidly grown in the last few years, is leading the creation of algorithms able to mimic human behaviour. ANN are computational models that take their inspiration from the brain's structure. They originated in mathematical neurobiology but have been broadly used in statistics as an alternative to traditional models.

ANN are structured in **perceptrons** or **nodes**, which imitate neurons, each of which is connected with some or all of its neigbouring nodes through a **propagation function**, which in its turn imitates a synapsis in the brain. When information flows from one node to the following, the reveived input is activated, through an **activation function**.

In this section, we will present the first ANN that were built and how they relate to traditional statistics.

2.1.1 The Perceptron

The Perceptron is the most elementary structure for an ANN, a single layer network with one hidden node and one output node. In some way it is just a new approach to multivariate regression which gives a graphical representation to statistical models, inspired on the structure of the brain.



(a) Schematich Diagram of a Real Neuron

(b) Artificial Neuron

Figure 2.1: Biological Neuron vs. Perceptron. *Source: Cheng and Titterington (1994)*

Figure 2.1 present a comparison of the two structures, but the similarity between these doesn't go any further. While the human brain contains millions of cells interconnected with each other to process, integrate and coordinate information received from the environment through a process which is still hardly understood, ANN can be seen as simple mathematical functions built on top of each other.

The artificial neuron, takes all input features and weights them into the hidden node.

Given a set of weights $w_j \in \mathbb{R}$, $j = 1, ..., d$, where d indicates the total number of input nodes, the value taken by a hidden node corresponds to a multivariate linear fit:

$$z = \sum_{j=1}^{d} w_j x_j + b_0 \tag{2.1}$$

The contribution of the Perceptron is the application of a transformation on this computed value to obtain the final output, the *activation function*. Thus, the output value is the result of applying a function, $g(.)$, to the previous fit.

$$y = g(z) \tag{2.2}$$

This said, we will now see how some statistical models had already done this.

2.1.2 The Perceptron and Statistical Models

The modelization of the hidden node we presented previously is a linear function. Thus, for the specific case of $g(z) = I_z$, the *identity function*, the Perceptron corresponds to the formulation of a linear regression.

$$y = \sum_{j=1}^{d} w_j x_j + b_0 \tag{2.3}$$

This is useful when we look for a continuous output, since the response variable to take values in the range $(-\infty, \infty)$. In a binary classification problem though, the output result is a discrete dichotomic value, generally 0 or 1. This variable appears when in a given sample we look if each indivuals holds or does not hold a target characteristic of the study and this is codified as $(Y = 1)$ or not $(Y = 0)$.

Logistic regression is a traditional model, used in a supervised learning problem of this type, where answers are tagged to 0 or 1. Relationships are modeled using the training data which has known labels and allows the estimation of the parameters. The goal is to accurately predict this tag and therefore, the error between predictions and known training data labels is minimized.

To contrain results between $[0, 1]$, in GLM, a transformation called the *link* function and notated as $f(.)$ is applied. Given the expected value μ and a linear predictor z, $z = f(\mu)$. Thus, the response function will be $\mu = f^{-1}(z) = g(z)$.

Some common *link* functions for binary data are:

- **Logit**

$$g(z) = \frac{e^{(z)}}{1 + e^{(z)}} = \frac{1}{1 - e^{-z}} \quad (2.4)$$

- **Probit**

$$g(z) = \Phi(z) \quad (2.5)$$

- **Log-log complementary**

$$g(z) = 1 - e^{e^{z}} \quad (2.6)$$

- **Log-log**

$$g(z) = 1 - e^{e^{-z}} \quad (2.7)$$

In ANN, these *link* functions correspond to the *activation functions*, and we will use the convention $\sigma(z) = g(z)$ for activation functions. The Logit link in particular is a very common transformation, called **softmax** in ANN. Thus, a Perceptron, which uses a softmax activation function will compute the same as a logistic regression with a logit link.

Now that we have proven that linear regression can be seen as a particular case of ANN, we will move forward to present more complex formulations.

2.2 Elements of an Artificial Neural Network

A single hidden layer ANN can be seen as represented in Figure 2.2, where the different elements are labeled. A neuron, n of the hidden layer i, receives signals $x_1, x_2, ..., x_d$ from all the nodes in the input layer j which are connected to it. Each of these connections has a computed weight $w_{i,j}$ that is optimized in the training process. The new node is composed by the weigthed sum of its inputs, $z_i = \sum_{j=1}^{d} w_{ij} x_j$, which is then passed to the chosen activation function, $g_i(z_i)$.

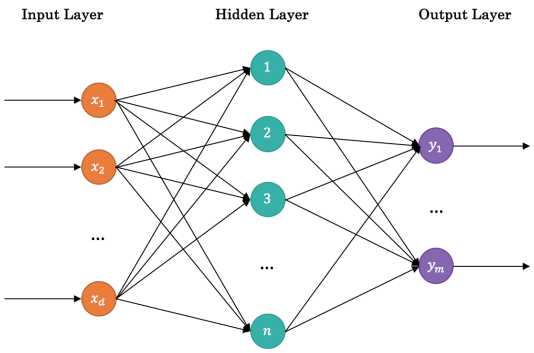


Figure 2.2: Single Hidden Layer ANN

In this section, we present all these elements in detail.

2.2.1 Nodes

The perceptron, or node, is the most elementary feature of an ANN. In some literature, *The Perceptron* is also used to call the full structure that we have seen before. Therefore, we will use the word **node** to refer to the elements of an ANN, in order to avoid confusions.

Each node receives inputs either from the environment or from other nodes. Those who receive the external information, form the input layer of the network.

As in the Perceptron, the associations among nodes are done through weights $w_{i,j} \in \mathbb{R}$, $j = 1, ..., d$, where d indicates the total number of connected nodes a the particular node $i = 1, ..., n$, where n is the number of nodes in a layer. These weights can be both called *synaptic* or *connection* weights.

We have seen the formulation for a single node. When there are several nodes in a layer, the formulation in generalized as follows:

$$z_i = \sum_{j=1}^{d} w_{ij} x_j + b_{i0} = \boldsymbol{w}_i^T \boldsymbol{x} + b_i \tag{2.8}$$

where $\boldsymbol{w_i} = [w_{i1}, ..., w_{id}]^T$, $\boldsymbol{x} = [x_1, ..., x_d]^T$ and b_i represents the intercept. In matricial notation, we can express the previous equation for the full layer as follows:

$$\boldsymbol{z} = \boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b} \tag{2.9}$$

Sometimes, the term $\boldsymbol{b}$ is also referred as **bias vector**.

Note, that in some ANN literature, the output value is also called x, instead of z. This is due to a simplification of the process. Following the reasonment we presented previously, we still need to apply the activation function on this node value, in order to have the *activated* value, which will be transferred to the following layers.

Usually, for hidden layers these activated values are also referred as $a_i \equiv x_i$, leaving the terminology x exclusively for the input values in order to avoid confusions. We will assume this notation.

Details on these activation functions can be found below.

2.2.1.1 Activation Function

When we talked about logistic regression, we presented several *link* functions used in GLM, and we highlighted that the *softmax* transformation was one of the most popular *activation* functions used in ANN. Yet, there are many different functions that have been used in practice. Here we present a list of the most common transformations and their graphical representation.

- **Identity**

$$\sigma(z) = z \tag{2.10}$$

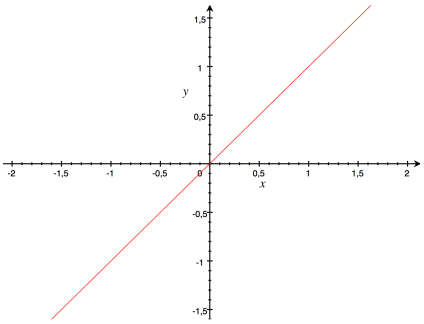


Figure 2.3: Identity Transformation

- **Threshold**

$$\sigma(z) = \begin{cases} 1 & \text{if } z > \theta \\ -1 & \text{if } z \leq \theta \end{cases} \tag{2.11}$$

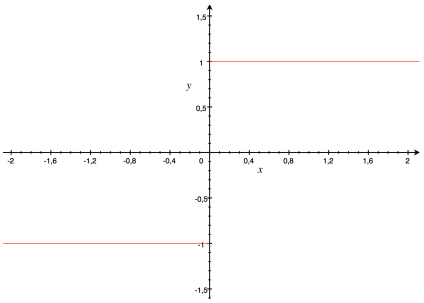


Figure 2.4: Threshold Transformation for $\theta = 0$

- **Softmax**

$$\sigma(z) = \frac{e^z}{1 + e^z} = \frac{1}{1 + e^{-z}} \tag{2.12}$$

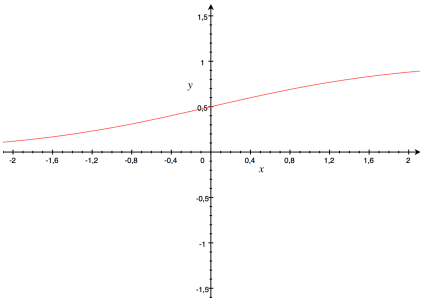


Figure 2.5: Softmax Transformation

- **Rectified Linear Unit (ReLU)**

$$\sigma(z) = z^+ = \max(0, z) \qquad (2.13)$$

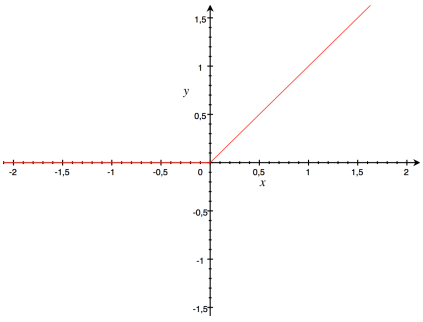


Figure 2.6: ReLU Transformation

- **Hyperbolic Tangent (tanh)**

$$\sigma(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{e^{2z} - 1}{e^{2z} + 1} \qquad (2.14)$$

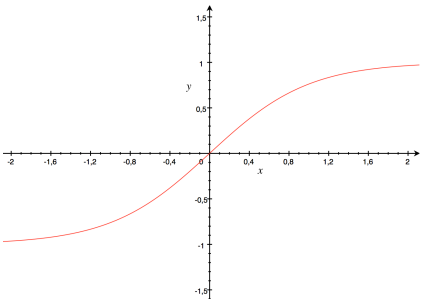


Figure 2.7: Tanh Transformation

Activation functions generally limit the range of values that a node will take, except the Identity transformation, which is rarely used as values can grow exponentially over the different layers. Softmax and Thresholds are used when the final output will be a discrete value, such as $[0, 1]$. The ReLU activation in particular, is very useful when values can't be negative or for large datasets, since a node with a negative value is automatically cancelled from the equation, by taking a 0 value. But, if we don't want to lose information in a regression, the tanh function is usually recommended.

2.2.2 Hidden Layers

Hidden layers have been mentioned a couple times until now. As one can deduce, hidden layers compose the internal structure of an ANN.

According to Alpaydın (2010), a Perceptron or an ANN that has a single layer of weights can only approximate linear functions of the input. But, several Perceptrons can be stacked to form a Multilayer Perceptron (MLP). These MLP can implement nonlinear discriminants in classification and can approximate nonlinear functions of the input, if used for regression.

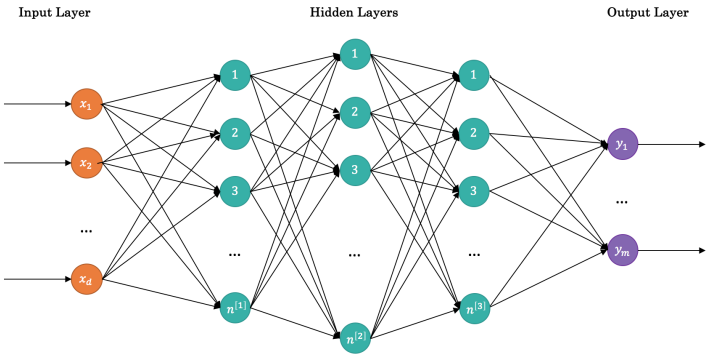


Figure 2.8: MLP Structure

In the structure of a MLP such as the one shown in figure Figure 2.8, the input and output layers are not counted as hidden layers. Thats why, Figure 2.2 was referred as a single layer ANN.

The standard notation for the hidden layers will be superscript [l], $l = 1, ..., L$, where L is the number of layers in the network. This notation is extended to all the parameters:

- $\boldsymbol{n}^{[l]}$: number of hidden units of the l^{th} layer.
- $\boldsymbol{a}^{[l]}$: activated value of the hidden units of the l^{th} layer.
- $\boldsymbol{W}^{[l]}$: weight matrix of the l^{th} layer.
- $\boldsymbol{b}^{[l]}$: bias vector of the l^{th} layer.

Each layer can have its own activation function.

2.2.3 Learning Rule

The learning rule is defined as the algorithm which optimizes the weights of an ANN in order to fit the desired output.

There are some different learning rules used in the literature, but before introducing them, we will need to talk about the building blocks for the optimization functions used to train the weights. Therefore, we will define both the *cost* and the *propagation* functions before moving on to the optimization algorithms.

2.2.3.1 Cost Function

Given some predictions $\hat{\boldsymbol{y}} \in \mathbb{R}^n$, and the desired output $\boldsymbol{y}$, a loss function, $L(\hat{\boldsymbol{y}}, \boldsymbol{y})$ measures the discrepancy between them. In other words, it computes the error for a single training example.

In ML there are six main loss functions which are commonly used. Different loss functions are more or less accurate for different data structures. In particular, one can distiguish **discrete** and **continuous** loss functions.

Discrete Functions

Two of the most common discrete loss functions, are **categorical crossentropy** and the **hinge loss**:

- **Categorical Cross Entropy**

$$L(\hat{\boldsymbol{y}}, \boldsymbol{y})_{CE} = -\sum_{i=1}^{n} y_i \cdot \log \hat{y}_i \tag{2.15}$$

- **Hinge Loss**

$$L(\hat{\boldsymbol{y}}, \boldsymbol{y})_{HL} = \max(0, \mathbf{1} - \hat{\boldsymbol{y}} \cdot \boldsymbol{y}) \tag{2.16}$$

The categorical crossentropy is also called the log loss function and its sum corresponds to the log-likelihood function for logistic regression which is why it is commonly used for this model. The hinge loss instead is usually applied to classification problems which use algorithms such as Support Vector Machines.

Continuous Functions

For this case, the most common loss functions, are the **Root Mean Square Error** and the **Mean Absolute Error**:

- **Root Mean Square Error**

$$L(\hat{\boldsymbol{y}}, \boldsymbol{y})_{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)} \tag{2.17}$$

- **Mean Absolute Error**

$$L(\hat{\boldsymbol{y}}, \boldsymbol{y})_{MAE} = \sum_{i=1}^{n} |\hat{y}_i - y_i| \tag{2.18}$$

While these work well for continuous outputs, they wouldn't have been optimal for dichomotic answers treated in logistic regression for example, where these metrics will lead to a non-convex optimization problem, i.e. it results in an optimization problem with multiple local optima. Thus, in order to find the global optima, we need to correctly define this measure.

The **cost** function is the average of the loss function of the entire training set. It optimizes, $\boldsymbol{W}$ and $\boldsymbol{b}$ to minimize the overall cost, $J(\boldsymbol{W}, \boldsymbol{b})$.

2.2.3.2 Propagation Function

Given the parameters $\boldsymbol{W}$, $\boldsymbol{b}$ and the corresponding cost function, $J(\boldsymbol{W}, \boldsymbol{b})$, we can find the local minimum using an optimization algorithm. If the defined cost function is convex, it will guarantee that this local optima is also the global optima.

In ANN, this optimization is implemented in two directions, what we call **forward** and **backward propagation**.

Forward Propagation

Forward propagation is just the sequential computation of the nodes. Therefore, the **propagation function** is defined as the computation of the input of a neuron, given the weights and the values of all the nodes in the previous layer that are connected to it, and the application of the corresponding activation function. The most general forward propagation equation is:

$$\boldsymbol{a}^{[l]} = g(\boldsymbol{W}^{[l-1]T}\boldsymbol{x}^{[l-1]} + \boldsymbol{b}^{[l-1]}) \tag{2.19}$$

Backward Propagation

Backward propagation is the reverse process of updating the weights from the end to the beggining. In order to do so, it computes the error of the values obtained through the forward propagation phase, which must have been done before. This computation will depend on the chosen optimization algorithm, which we present in the following section.

2.2.3.3 Optimization Algorithms

We present and build the backpropagation functions of three main optimization alogrithms: gradient descent, RMSprop and Adam optimization.

Gradient Descent

Gradient Descent, is an optimization method which iteratively optimizes differentiable cost function. It is based on the traditional optimization of functions: if we want to find

$\boldsymbol{W}$ and $\boldsymbol{b}$ that minimize a particular cost function, $J(\boldsymbol{W}, \boldsymbol{b})$, we can find this minimum value by taking the partial derivatives equal to 0, $J'(\boldsymbol{W}, \boldsymbol{b}) = 0$

Thus, in Gradient Descent, the weights and the bias are updated according to the following functions:

$$\boldsymbol{W} := \boldsymbol{W} - \alpha \cdot \frac{\partial J(\boldsymbol{W}, \boldsymbol{b})}{\partial \boldsymbol{W}} \tag{2.20}$$

$$\boldsymbol{b} := \boldsymbol{b} - \alpha \cdot \frac{\partial J(\boldsymbol{W}, \boldsymbol{b})}{\partial \boldsymbol{b}} \tag{2.21}$$

where α indicates the **learning rate**, which determines the magnitude of change to be made in the parameter. It is generally taken between 0.0 and 1.0, but mostly $\alpha \leq 0.2$. We can get an intuition of why we are interested in small learning rates from the follwing graph:

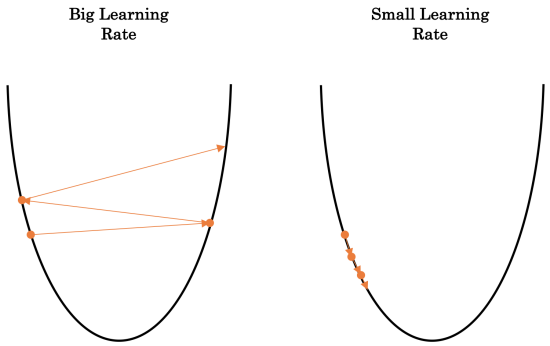


Figure 2.9: Gradient Descent Learning Rates

The problem with a fixed learning rate is that it can be very slow to converge if it isn't properly tuned. Therefore, an adaptative learning rate can be introduced either manually or through other optimization methods which speed up this convergence.

RMSprop

RMSprop, which stands for Root Mean Square Propagation, is an optimization algorithm which speeds up the convergence of the values, by denoising some unnecessary oscilations. This is very important in ML applications, since we are not talking about an $\mathbb{R}^2$ dimensional space, as shown in Figure 2.9 but a very high dimensional space instead. Thus, if steps are taken in the wrong direction it can be very hard to eventually reach the optimal value.

The term RMSprop, comes from the implementation of an exponentially weighted

average of the squares of the derivatives. At each iteration, i, $S_{\partial w^i}$ and $S_{\partial b^i}$ are computed as follows:

$$S_{\partial w^i} = \beta_2 S_{\partial w^{i-1}} + (1-\beta_2)\left(\frac{\partial J(\boldsymbol{W},\boldsymbol{b})}{\partial \boldsymbol{W}}\right)^2 \tag{2.22}$$

$$S_{\partial b^i} = \beta_2 S_{\partial b^{i-1}} + (1-\beta_2)\left(\frac{\partial J(\boldsymbol{W},\boldsymbol{b})}{\partial \boldsymbol{b}}\right)^2 \tag{2.23}$$

$S_{\partial w^0} = 0$ and $S_{\partial b^0} = 0$. The parameters W and b are updated according to the following equations:

$$\boldsymbol{W} := \boldsymbol{W} - \alpha \cdot \frac{1}{\sqrt{S_{\partial w^i} + \varepsilon}} \cdot \frac{\partial J(\boldsymbol{W},\boldsymbol{b})}{\partial \boldsymbol{W}} \tag{2.24}$$

$$\boldsymbol{b} := \boldsymbol{b} - \alpha \cdot \frac{1}{\sqrt{S_{\partial b^i} + \varepsilon}} \cdot \frac{\partial J(\boldsymbol{W},\boldsymbol{b})}{\partial \boldsymbol{b}} \tag{2.25}$$

The main advantage is that β_2 can be fixed to 0.999 and tuning α becomes less important, since the learning rate is indirectly decayed at each iteration. A very small value $\varepsilon = 10^{-8}$ is usually added in order to avoid zero divisions.

Adam Optimization

Finally, the Adam optimization algorithm is an extension to stochastic gradient descent, which combines momentum and RMSprop and puts them together. This algorithm has gained a lot of popularity in DL.

Additionally to the terms presented in RMSprop, the Adam optimizer introduces the momentum exponentially weighted averages, $V_{\partial w^i}$ and $V_{\partial b^i}$ at each iteration. The full formulation is the following:

Given, $V_{\partial w^0} = 0$, $V_{\partial b^0} = 0$, $S_{\partial w^0} = 0$ and $S_{\partial b^0} = 0$.

$$V_{\partial w^i} = \beta_1 S_{\partial w^{i-1}} + (1-\beta_1)\frac{\partial J(\boldsymbol{W},\boldsymbol{b})}{\partial \boldsymbol{W}} \tag{2.26}$$

$$V_{\partial b^i} = \beta_1 S_{\partial b^{i-1}} + (1-\beta_1)\frac{\partial J(\boldsymbol{W},\boldsymbol{b})}{\partial \boldsymbol{b}} \tag{2.27}$$

$$S_{\partial w^i} = \beta_2 S_{\partial w^{i-1}} + (1-\beta_2)\left(\frac{\partial J(\boldsymbol{W},\boldsymbol{b})}{\partial \boldsymbol{W}}\right)^2 \tag{2.28}$$

$$S_{\partial b^i} = \beta_2 S_{\partial b^{i-1}} + (1-\beta_2)\left(\frac{\partial J(\boldsymbol{W},\boldsymbol{b})}{\partial \boldsymbol{b}}\right)^2 \tag{2.29}$$

$$\boldsymbol{W} := \boldsymbol{W} - \alpha \cdot \frac{V_{\partial w^i}}{\sqrt{S_{\partial w^i} + \varepsilon}} \tag{2.30}$$

$$\boldsymbol{b} := \boldsymbol{b} - \alpha \cdot \frac{V_{\partial b^i}}{\sqrt{S_{\partial b^i} + \varepsilon}} \tag{2.31}$$

As in RMSprop, $\beta_2 = 0.999$ and β_1 can be fixed to 0.9. Again, the initial choice of α has a lower impact on the full learning process.

2.3 Variants of Artificial Neural Networks

Now that we have seen all the building blocks of ANN, we will see that several variants of ANN can be defined according to their different arquitechtures. In particular, the three main variants used for different RS will be presented: **Convolutional Neural Networks**, **Recurrent Neural Networks** and **Autoencoders**.

All of these architectures are built over several hidden layers. Hence, we are talking about **deep learning algorithms**.

2.3.1 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are deep feed-forward ANN which are generally used for image processing. Their basic structure is the same of a regular MLP: an input layer, multiple hidden layers and an output layer. The difference falls on the architecture of the hidden layers, which include:

- **Convolutional layers** take all the input nodes and compute a single output instead of computing an output for each node. Mathematically, the **convolution** corresponds to **cross-correlation**. The convolution emulates the activation function, but in this case it has the advantage that is reduces the number of free parameters. This is specially relevant for images, where every pixel correspond to three diferent input values in an RGB setting.

Convolutions can be **valid** or **same**. The first reduce the dimensionality while the latter, pad the resulting output in order to keep the original dimension. **Strides** can also be added to convolutions in order to reduce even more the dimensions, by "jumping" part of the combinable clusters.

- **Pooling layers** reduce the dimensions of the original image by clustering several pixels and combining them into one node. There are two main clustering techniques than can be applied: *max pooling*, which takes the highes value from each cluster; or *average pooling*, which computes the average of all the pixels in a cluster.

- **Fully-connected layers** correspond to the traditional layers where every node from one layer is connected to all the neurons in the following one.

One setting in particular, known as the **LeNet-5**, became very famous for optical character recognition. Its structure can be depicted as follows:

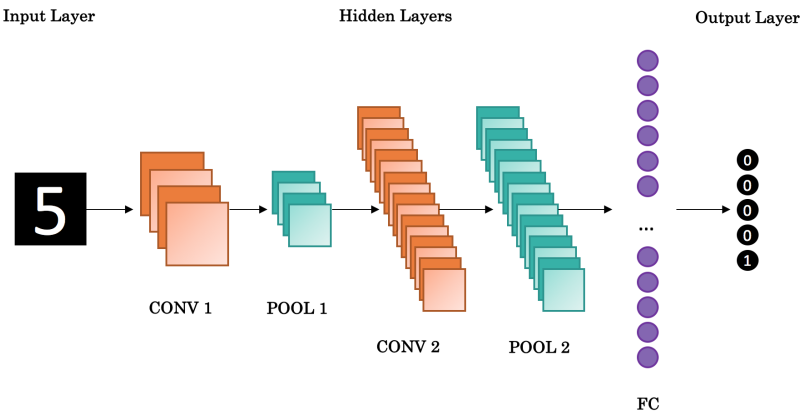


Figure 2.10: LeNet-5 5-layer CNN for Optical Character Recognition

CNN can also be used for image classification, and for object detection, where several objects can be detected inside a single image. Videos and movies can be seen as a sequence of images, thus, CNN are used in combination with RNN to make video and movie recommendations.

2.3.2 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are deep ANN, that, unlinke feed-forward networks, retain previous information to process sequences. In order to do so they can have a single output at the end or several sequential outputs in intermediate layers, which, in its turn, are feed to the inputs of the next layer.

A fully recurrent ANN as shown in Figure 2.11, loops on itself infinitely, being able to give real-time output at each period of time t.

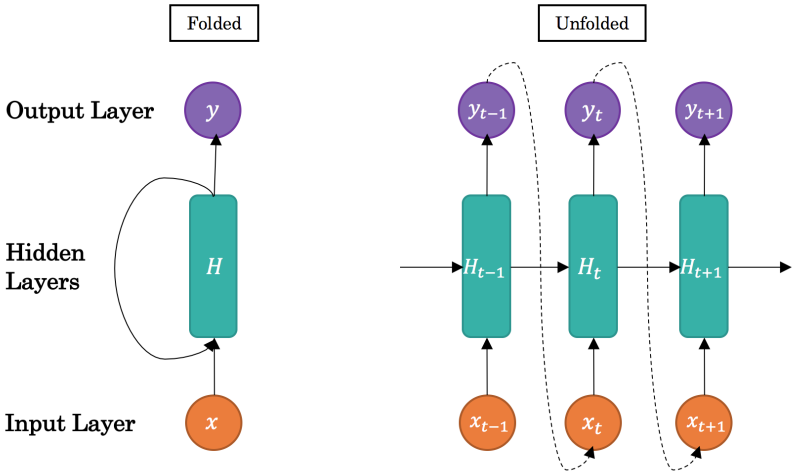


Figure 2.11: Folded and Unfolded Basic RNN

The main **problem** in RNN becomes computing gradients. In particular, the high number of connections can result in **vanishing gradients**, when over many iterations, derivatives become extremely small.

Some alternatives have been proposed to deal with this problem, such as **Long-Term Short Memory** (LTSM) networks and **Gated Reccurent Units** (GRU). Both of these algorithms present mechanisms to "forget" gates over time, putting a higher emphasis on most recent outputs but at the same time allowing temporal coherence.

We won't get into more detail on these, since our particular dataset doesn't have sequential data. Presenting them was relevant though, because not only video and movie recommenders use them, but also music RS and sentiment classification, which is very important for **written reviews**.

2.3.3 Autoencoders

Autoencoders are a particular case of ANN, born from the idea of mapping an input into a lower-dimensional representation, which can be later reconstructed to match its original form. Therefore, they are typically used for dimensionality reduction.

Figure 2.12 presents the basic structure of an autoencoder. Independent from the number of layers, autoencoders have two main parts:

- **Encoder**: set of layers which reduce input dimensionality, $\mathbb{R}^d \rightarrow \mathbb{R}^n$.

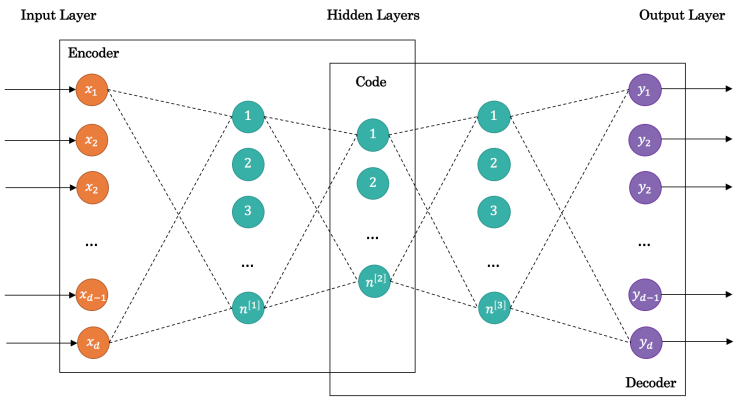


Figure 2.12: Structure of an Autoencoder with 3 Fully-Connected Hidden Layers

- **Decoder**: set of layers which reconstruct the input, $\mathbb{R}^n \rightarrow \mathbb{R}^d$.

Undoubdetly, autoencoders are forced to compress the input, learning how to reduce noise in the process and obtain a d dimensional representation of the data.

Architecturally, a simple autoencoder doesn't differ from a feed-forward simple MLP, which has an input layer, an output layer and one or more hidden layers connected. But, the output layer has the same number of nodes as the input layer and, in order to make it map them against each other, we just need to feed it with the same input and output sorted accordingly.

Some stacked autoencoders have been used for image recognition but we will stick to its most simple formulation.

In order for an autoencoder not to learn the identity functions, some variations have been proposed over the years: **denoising autoencoders**, **sparse autoencoders**, **variational autoencoders** and **contractive autoencoders**.

Autoencoders for Recommender Systems

Q. Li, Zheng, and Wu (2017) present a generic recommender framework called Neural Collaborative Autoencoder (NCAE) to perform collaborative filtering, which works well for both explicit and implicit ratings. NCAE can effectively capture the relationship between interactions performing a non-linear matrix factorization process.

Denoising AE have also gained popularity in RS. These take partially corrupted inputs and try to reconstruct them entirely. In RS, unavailable reviews do not forward any information; adding a small blank noise, the strongest trends are. Strub et al. (2016) built a Stacked Denoising Autoencoder (SDAE)

Other authors have also developed variants of AE to implement Collaborative DL. H. Wang, Wang, and Yeung (2014) in particular, integrate SDAE with Probabilistic Matrix Factorization to perform deep content feature learning and collaborative filtering jointly.

2.4 Parameter and Hyperparameter Tuning

Hyperparameters are defined as those parameters that control lower-level parameters. Thus, the optimization of the latter depends on these hyperparameters.

In an ANN, parameters are $\boldsymbol{W}$ and $\boldsymbol{b}$, which are optimized during training. Hyperparameters refers to the rest of parameters which will determine how fast or slow this optimization process goes and how good the algorithm will be able to predict.

Hyperparameters can be ranked, by how important it is to accurately tune them:

- **Learning rate**. It is the most relevant feature in an ANN, the one that will determine convergence speed. The higher this value is, the faster it will converge but it might also turn into high oscilations around the optimal value, without every reaching it. In order to reduce the pressure on this hyperparameter, we can introduce a learning rate decay or algorithms such as RMSprop and Adam, which we have seen weigth its value in every successive iteration.
- **Learning rate decay**. Determines how fast the learning rate is reduced over iterations.
- **Number of layers**. Determine the basic structure of the ANN, and by that, the number of parameters to be optimized.
- **Number of hidden units**. In a lower scale than the layers, these are also determinant for the number of parameters to be optimized.
- **Mini-batch size**. Instead of training with the complete dataset, smaller batches can be optimized sequentially in each iteration. The smaller its size is, the more loops we will need to do inside an iteration, but it will also consume less memory.
- $\boldsymbol{\beta_1}$, $\boldsymbol{\beta_2}$ and $\boldsymbol{\varepsilon}$ can all take default values, 0.9, 0.999 and 10^{-8} respectively.

When tuning hyperparameters in an ANN, a grid might not always be recommended, since in some cases, hyperparameters do not follow a convex distribution, and grid values migh just be skipping its peaks as shown in Figure 2.13. Instead, random values are more recommended for the first set up.

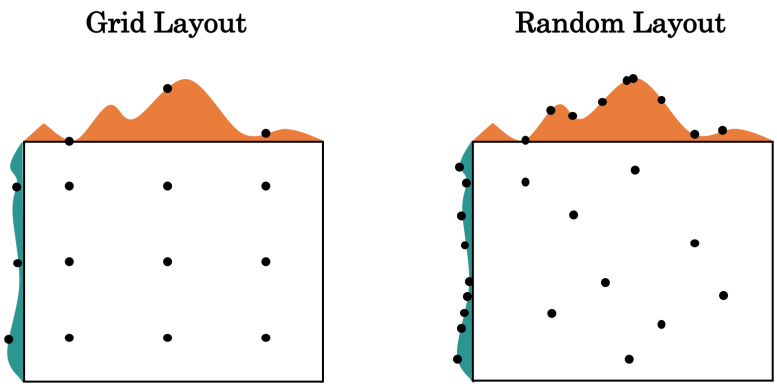


Figure 2.13: Grid-layout vs Random Seach

When trying random numbers, we can achieve combinations which would have never been found on a grid. This seach can be later refined by taking random numbers around the intervals that resulted in better accuracies.

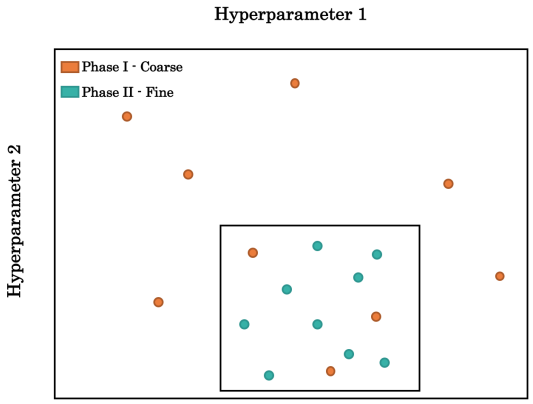


Figure 2.14: Coarse to Fine Search

2.5 Programming Frameworks

When dealing with larges amounts of data, implementing all the functions manually can be both time-expensive and costly. To have an idea, for a feed-forward ANN with one fully connected hidden layers such as shown in Figure 2.15, we already have $(3 \cdot 4) + (4 \cdot 2) = 20$ weights, plus its corresponding 20 bias parameters. If this hidden layer had 10 nodes instead, these would already have $(3 \cdot 10) + (10 \cdot 2) = 50$ each, over twice as many by just adding 6 nodes, and this keeps rising exponentially if we add more hidden layers. Therefore, even if it is interesting to know how all the implementations work, it is much more efficient to work with a programming framework.

Moreover, the bias parameter becomes insignificant when working with very large ANN, which is why many frameworks leave it aside.

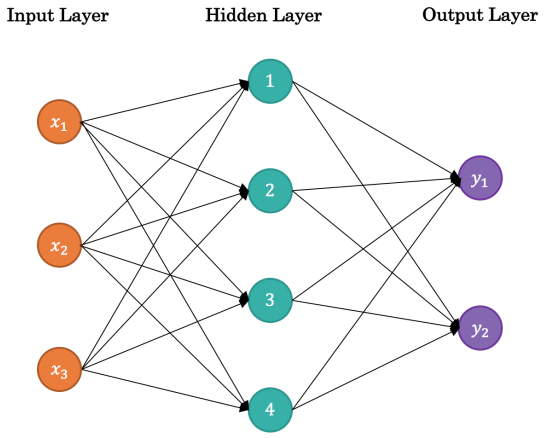


Figure 2.15: Single Layer ANN

Programming frameworks are platforms for developing software applications, Christensson (2013). These frameworks provide predefined object classes and functions which can be useful to build specific programs. They work in a similar way as APIs but they might also include, code libraries and a compiler. Additionally, these compilers might be prepared to run the code in parallel, speeding up the process.

These way, the specific syntax of a programming language becomes less relevant as it is the architecture what is actually implemented.

So programming frameworks offer some clear **advantages** when dealing with large architectures in ML. Some of the reasons to choose one are:

- **Ease of programming**. When building architectures one can really focus on the development and deployment of the programm.
- **Running speed**. Since they are prepared to deal with large data, functions have already been optimized and might even run in parallel.
- **Truly open**. Many programming frameworks are *open source*, which allows any advanced user to make updates and add missing functionalities.

When working with ANN two main frameworks stand out: **TensorFlow** and **Keras**. Both of them have been combined in this thesis to implement the Autoencoder ANN, which is why we are going to make a brief introduction to both of them.

2.5.1 TensorFlow

TensorFlow was originally developed by the Google Brain team for internal use, but it was released on open-source in 2015.

Nowadays, TensorFlow is an open source software library for Machine Intelligence, The TensorFlow Authors and RStudio (2018), broadly used for machine learning applications, such as ANN, thanks to its high efficiency.

It is called symbolic because it allows us to build structures over tensor placeholders which do not really contain any data. Tensors are the basic structure of TensorFlow, which imitates a mathematical tensor. When the full architecture is built, it can be run by chosing optimizers from predefined functions.

TensorFlow can run in parallell on multiple CPUs and GPUs and it has official APIs for Python and C. R creators have also developed a package to run the Python API from R and RStudio. This R interface, is able to work jointly with the high-level Keras and Estimator APIs, and at the same time, access the full core of the TensorFlow API when needed.

2.5.2 Keras

Keras is an open source Python ANN library, which is able to run on top of other programming frameworks such as Tensorflow. It provides a very user-friendly approach to ANN, by having implemented all the necessary building blocks: layers, objectives, activation functions and optimizers.

Although it can only be use distributed storage when working on a GPU, it provides several mechanisms to implement custom generators to reduce out-of-memory problems on CPU.

Next to the development of TensorFlow, R creators have also developed a Keras API to work from R and Rstudio. According to its creators, Allaire et al. (2018), it is a high-level ANN API developed with a focus on enabling fast experimentation. This API allows the creation of ANN using `dplyr` pipelines, `%>%`, to sequentially add layers to the model.

Chapter 3

Chatbots

Chatbots (also called ChatterBots or IM bots, among others) are AI instant messaging systems able to conduct a conversation with humans. These have been implemented for different purposes, including customer service or conversational commerce.

Their origin goes back to Alan Turing's formulation of the **Turing test**, in which he theorized that a truly intelligent machine would be indistinguishable from a human when talking to it. Turing is seen as one of the fathers of computation as we know it and his ideas set the ground for the revolutions we are living in ML and AI, including the rise of Chatbots.

The first program to pass the test was the chatbot *ELIZA*, created in 1966. *ELIZA* simulated the responses of a psychotherapist using word patterns and a predefined response database. That means that she was programmed to recognize keywords or phrases in the input for which a response was stored in the database. Therefore, it would give the same answer to any input that contained a particular word.

In 1972 a patient for *ELIZA* was developed, *PARRY*, a chatbot simulating a paranoid schizophrenic. Both chatbots have been set up to mantain a conversation with each other several times.

Since then, advances in technology have allowed the creation of much more complex AI systems. Turing himself, wrote about the idea of imitating behaviours in games which has bring to the imitation game being a controverted alternative formulation to the Turing test.

In this gaming branch, IBM's Watson1 made one of the most famous achievements when, in 2011, it won the game Jeopardy! against two of its best players. Originated in

2006, Watson is a referent in the NLP field.

Nowadays chatbots are implemented on several platforms, such as messaging apps, companies' internal platforms and toys; although only 4% of the companies are using chatbots as part of their websites or apps, according to a 2017 study, Capan (2018). Nevertheless, the leading technological have all developed their own AI assistants, such as Apple's Siri2, Amazon's Alexa3, Microsoft's Cortana4 or Google's Google Assistant5.

3.1 Building blocks

Chatbots generally use Natural Language (NL) techniques to communicate but some of them only use word patterns as was *ELIZA*'s case. Nonetheless, this latter version is not so common anymore, as the NLP field has advanced a lot over the last years. A third option is avaliable for very restricted communications, structured guided dialogs.

In this section, we introduce the NL Dialogue and the Guided Dialogue structures.

3.1.1 Natural Language

NLP is a branch of computer science and artificial intelligence that helps computers understand, interpret and manipulate human language, Inc. (2018). It draws from many disciplines and quick advances in its technology have been done among others thanks to an increased interest in this human-machine communications.

Basic NLP tasks break down language into samller elemental pieces, in order to understand relationships between them to create meaning. However, this tasks includes many different techniques for interpreting human language, such as ML methods. And there is not only a wide range in its approaches, practical applications also vary a lot from text to voice inputs.

Using NLP, some of the main capabilities are:

- **Content categorization**. Document contents can be summarized or indexed, allowing the identification of relevant information and duplication detection.
- **Topic discovery and modeling**. Text collections can be classified by their meaning or theme and advanced analytics, such as optimization and forecasting can be applied to these texts.
- **Contextual extraction**. Information from structured text-based sources can be

automatically retrieved.

- **Sentiment analysis**. Moods and subjective opinions within sentences or larger texts can be idenified and classified.
- **Speech-to-text and text-to-speech conversion**. Voir commands are written into text, and vice versa.
- **Document summarization**. Synopses of large text bodies can be automatically generated.
- **Machine translation**. Text or speech can be automatically translated from one language to another.

NLP is very important for text analytics, when dealing with large volumes of textual data and structuring unstructured data sources. For chatbots, it is usually the core element.

Chatbots with NLP cores, offer a much better user experience since they can ask anything using their own words. But to train a bot to understand questions in different ways and to develop a working NLP system which gives reasonable answers requires a large dictonary and generally very deep trained models.

Since the chatbot in this thesis is only to serve as a front-end to our developed RS, such an effort was considered to be out of scope, although it remains as future work. Therefore, we opted for a much more guided approach, which required little NLU.

3.1.2 Guided Dialogue

When the use of a chatbot is very limited to a specific purpose, a multiple-choice menu might usually be enough to gather required information.

The principle is to guide the user in the process to offer him what he wants. The bot is responsible for driving the conversation in a structured manner by asking a question and suggesting a limited range of answers from which the user should choose one. This process continues until the bot has retrieved all the necessary information.

It offers a very basic user experience but it is very convenient for most businesses as it goes straight to the point, Misoffe (2017). Complexity can be increased by letting users answer freely but this might just complicate the process even more than just opting for a NLP approach.

A guided dialogue will be implemented in this thesis using `InlineKeyboardMarkup` objects from the `telegram.bot` package.

3.2 A Chatbot as a Recommender System Front-End

As we defined in Section 1.2.3, the introduction of Constrain-Based Recommendations is usually set up as a dialog. The belief that such constraints were needed in a Book RS were the reason to implement the final model of this thesis on a chatbot.

Research on conversational bots is very intensive nowadays and may open source solutions are enabling their creation. However, most of them use simple dialog managers to program conversation and few chatbots have made it to the market and are generating profits.

From the intersection of AI and ML, come a lot of new opportunities for chatbots. For example, the lastest developments in RNN have achieved to even generate poems, Zhang and Lapata (2014), which is a huge step toward instant answer generation.

On the other hand, very few companies have taken advantage of chatbots to implement RS able to make the final purchase on their own website. Researching the most used messenger applications that support chatbots (Facebook's Messenger and Telegram), we were only able to find a few relevant examples, all of them on Messenger.

One of the most relevant bot is probably *Poncho*, the weather bot. It has been on the top rankings of chatbots since it was launched in early 2016 as one of the first bots of the Messenger platform when it opened to chatbots developers. It offers a simple feature but thanks to its use of NLP, it becomes a very user-friendly app.

The *CNN* also developed a similar bot updating users with daily news. But, in contrast to Poncho, it offers several items at the same time. On 2016, they spent over six months rolling out a variety of chatbots across several messaging apps, McEleny (2016). They considered messaging apps to be an important new channel for them to be the worldwide leader in mobile and video news and information.

When it comes to e-commerce businesses though only some of the largest retailers have developed their own bots which enable users to make their purchases during conversation, Quoc (2017). One of Messenger's best ranked chatbots in this area is *1-800-Flowers.com*, which allows customers to purchase and send flowers and personalized gifts for different events. Also in the retail industry we want to highlight *Burberry*, *H&M* and *Sephora* (the latter two have bots implemented in Kik, a new messaging app which has recently become popular among 13 to 17 year olds).

Finally, the most relevant chatbot for this Thesis is the *eBay ShopBot*. The idea was first developed as a simple tool to remind bidders 15 minutes before an auction listing was

about to end so that they could make their last-minute bids. Now, the bot acts as a virtual personal shopping assistant, to help people find items on eBay. Through conversation, users can make their specifications and even set price limits to their search. The bot doesn't specifically act as a RS but it has the capacity to filter results based on user's constrains, a feature which will be very important during our own development.

Two bookstores also seemed to have bots implemented on Messenger, but after several tries over the last months, we have determined that they are actually inactive. We could see their first question though, which is aligned with the idea we had on the development of Allyn, which is not only to be able to recommend a book for the users, but also to look for a gift for someone else.

Part I

An Economic Analysis

Chapter 4

State of the Art Recommender Systems

To date, many uses have been developed for RS but there are also many potential uses that still haven't been completely fulfilled. While the purpose of implementing a RS is clear, there isn't a full agreement on the evaluation of their economic impact. In this chapter, we discuss both their current and potential applications as well as their impact on revenues.

> "*When making a choice in the absence of decisive first-hand knowledge, choosing as other like-minded, similarly-situated people have successfully chosen in the past is a good strategy.*" - Hill et al. (1995)

Since the first papers on RS appeared, the concept has been clear and an entire research area of growing interest started to form, the first papers of which date from the mid-1990s as Adomavicius and Tuzhilin (2005) state. The same authors, point out that the interest in this rich research area still remained high because of the abundance of practical applications to help users to deal with information overload and provide personalized recommendations.

That was 2005, and over 10 years later this interest has done nothing but keep growing. Information overload is higher than ever, that is common knowledge, and little are those who manage to take the most out of it; retrieving pertinent information from the copious resources available is not only difficult but also time consuming (S. Vairavasundaram et al. 2015). In 2005, only 51% of the population in the developed world, were active users of the internet, according to the Internation Telecomunications Union (2005) but high-speed Internet connections have revolutionized the world since then.

Recommender Systems can be implemented in various ways, but it has been the Internet which made a change and facilitated its use in several different areas. As Bobadilla et al. (2013), point out, the most common research papers are focused on movie recommendation studies; however, a great volume of literature for RS is centered on different topics, such as music, television, books, documents, e-learning, e-commerce, applications in markets and web search, among others.

Over the years, the accuracy of this systems hasn't stood aground, nor have the algorithms to evaluate these. The kinds of filtering most used at the beginning of the RS were described in Breese, Heckerman, and Kadie (1998) who evaluated the predictive accuracy of different algorithms but the base for evaluating the CF RS wasn't described until years after.

Our application will be focusing on recommending books but this doesn't live on its own. Furthermore, our system, will lead to an e-commerce website which will allow users to buy the recommended books right away, thus, affecting a company's revenues. Therefore, we will present an analysis of how different RS have effectively boosted online commerce.

4.1 Current applications in the Market

In Chapter 1.2, we defined four main types of recommendation techniques (collaborative filtering, content-based filtering, kownledge-based and hybrid filtering). Here, we're going to analyse how and which companies actually implement them.

CF is probably the best known recommendation technique, to the point that, it has been regarded as a synonym of the entire field in several occasions. As we have seen, it is the utmost simple technique to deliver recommendations which mainly relies on a person's ratings, ignoring all additional information we might have. The broadly used and well-known dataset to learn about recommender systems, MovieLens, is a plain and clear example of this algorithm's application.

The MovieLens datasets were first released in 1998 (Harper and Konstan 2015). The datasets show people's preferences for movies and as to store the expressed preferences, {user, item, rating, timestamp} information is saved by rows. The website has used several different CF algorithms over the years. Starting by the rating cycles presented in Resnick et al. (1994), the previously cited article also gives us a precise insight on some of these architectures used over time. The 1997 and 1999 versions used user-user CF, while it turns over to item-item CF in 2003 (see more in Harper and Konstan (2015)).

CB filtering algorithms on the other side, have become popular over the years as websites started to retrieve information from their user's accounts. Thus, these are mainly implemented on websites that require us to login with a personal account but as we have seen in Chapter 1.2.2, content doesn't exlusively refer to the user, associated features can also be the ones from the item itself. Shopping websites use user-based content filtering, while news websites like the Washington Post use item-based content filtering.

User's features generally refer to sociodemographic characteristics of these. Therefore, we will briefly discuss user-based content filtering alongside the the demographic recommendation techniques posteriorly.

Regarding item-based filtering, we have seen that algorithms need further machine learning techniques such as text mining to identify an item's content. For this task, the Washington Post team developed Clavis. According to Graff (2015), the original idea of developing Clavis was for it to be a classifier that automativally added keyword to stories. The final though, is a full recommender system that runs term-frequency, inverse document frequency (tf-idf) algorithm through articles in order to determine its most relevant words. In a nutshell, it figures out what stories are about and does the same process for each reader to finally find it's best matches.

On behalf of demographic recommendation techniques, we have already mentioned that it's main benefit is the unnecessay history of user ratings. Inuitively it is the most common technique applied by retail sellers when a new customer enters the brick-and-mortar store, i.e., to recommend a book to a client you would pay attention to his age and genre, among others. To implement this online, we need a user profile since a simple IP address won't give us all the information we're looking for. A plain demographic recommender would sequentally classify each user in a different group according to it's preferences and show him the choices of the group. Nevertheless, this sociodemographic features are generally used jointly with another technique. Therfore, we move to hybrid filtering techniques.

Hybrid filtering techniques refer to any combination of at least two algorithms from above and are the most poopular in the market nowadays.

Amazon, uses personal information to segment users in different cluster models with internally similar profiles. This allows a new user to be assigned right away before having purchased or rated any item. Then, an item-item CF is performed in order to identify items with high rates, which the present customer hasn't seen yet (Linden, Smith, and York 2003). But Amazon's business itself has changed a lot over the last years and new assumptions have slowly been made in order to define *related* items for example (Linden and Smith 2017), more on that in Section 5.1.

YouTube is another great example of the evolution of Recommender Systems. Originally, the website focused on previous videos seen by the user to recommend further items. Currently, the system is composed of two Deep Neural Networks with many different inputs comprised (Covington, Adams, and Sargin 2016). The first network focuses on the candidate's history in order to select a reasonable amount of videos from the full YouTube corpus that will make it to the second network. The latter adds video features and other external information jointly with the candidate's reviews to output a handful of recomendations. The following feature, extracted from the mentioned publication, shows us the explained *funnel*.

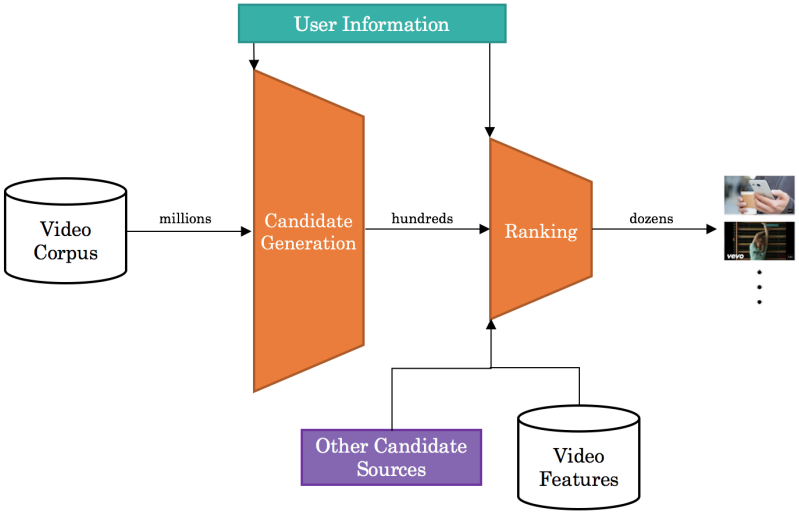


Figure 4.1: Recommendation System Architecture Demonstrating the Funnel where Candidate Videos are Retrieved and Ranked before Presenting only a few to the User. *Source: Covington, Paul et al. (2016)*

4.2 E-commerce and Recommender Systems

Nowadays, people are more inclined towards online shopping than rather goin to brick-and-mortar stores, and when they do, they stand in them, their smartphones in hand ready to compare prices and product reviews according to Mackenzie, Meyer, and Noble (2013). In the EU-28, enterprises realised 18 % of their total turnover from e-sales during 2016, consisting of orders via a website or apps or via EDI-type messages (Eurostat 2018). This drastically changes the market. There is no limited window or shop space which makes the retailer cut out on some products, even those that have a very low demand can be displayed. But the website can become overwhelming and retailers need to get to know their users preferences and interest. This is the main reason, for which RS are widely accepted (Chandak, Girase, and Mukhopadhyay 2015).

In this section, we will present the main advantages of using RS in e-commerce sites to help improve client retention. We have grouped them into four main groups: **personalization**, **customer satisfaction**, **discovery** and **revenue**.

Personalization

When browsing the web, we have a door open to almost every product in the world but we don't want to look unguided through it all. As the previous articles cites, we want to be taken care of by the system who should provide us with intelligent solutions and, at the same time, online retailers want to be able to satisfy our interests for us to become their long term customers. That hasn't changed from the brick-and-mortar store. But, the main difference is, we don't have a single shop window where we have to display our best matches for the majority of the users. The home screen provided to each individual customer can be sorted in a different way according to his/her preferences. Nevertheless, evidence shows that users can only process about the first 20 titles. Then, a question arises, is the relevant space of a website also limited?

In this process we can analogize the rows in a website to the aisles of a brick-and-mortar store as Ba and Pavlou (2002) suggest. Online interconnected products, would then resemble the products placed on neighboring shelves in the store. Therefore we will need to make sure, interesting contents are on the top of the page and, in order to know each person's best match, we need Recommender Systems.

Customer satisfaction

When setting up recommendations, another question arises: when do we need to provide the user with similar items to those he has previously liked? And when, need they be different? This brings us back to some basic economic concepts: some products are substitutes and others are complementary. Substitute products have the same function for the user such as having a PepsiCo drink or a Coke. It wouldn't make sense to recommend the second if the first is already in his cart, although it might get a potentially high rating from the user. On the other hand, complementary products work best together, for many, coffee would have little impact on the utility function without milk. Recommending the latter when the consumer added the coffee in his/her cart would be suitable, although he might not forget anyway but this is a topic we will address later.

Oestreicher-Singer, Gal and Sundararajan, Arun (2011) suggest that on average, the explicit visibility of a purchase resulting from a recommendation (also called co-purchase),

can increase up to three-fold the influence that complementary products have on each others' demand levels. A clear example of complementary suggestions would be, for example, that if we are looking for cookware in Amazon.com and a frying pan catches our attention, we are very likely to get a glass lid suggested as shown in Figure 4.2.

Figure 4.2: Outgoing Co-purchase Suggestion for a Cookware Product sold on Amazon.com. *Source: Amazon.com*

But then, the utility derived from consuming certain goods is not in their form or use, but in their content, such as in books or movies. In these cases, users will usually reveal a preference for a certain type of content, for a genre. Therefore, recommendations will generally focus on similar items and only show something new for the user to try from time to time. In Figure 4.3 two Fantasy Books about life facts. As the two products do not need to be consumed at the same time, they can both be purchased together.

Figure 4.3: Outgoing Co-purchase Suggestion for a Book sold on Amazon.com. *Source: Amazon.com*

Not missing out on the recommended copurchases can significantly increase customers' satisfaction.

Discovery

Some of the products previously shown as copurchases can be seen as newly discovered products by the user an aparently, this would strangle variety while consumers strive for it. On the contrary, Ba and Pavlou (2002) show that category diversity increases up to 1pp in copurchases.

But diversity isn't randomly introduced, assortative mixing, is a technique which introduces a bias in the recommendations in order to favor of connections between similar nodes in a network. Hence, it allows us to learn about new products or item from users with similar taste. Limit to it is that it won't suggest a categorically diverse product, therefore, we need network diversity, which captures full differences among its users.

Revenue

According to Sharma, Hofman, and Watts (2015), recommenders generate between 10 and 30% of site activity and revenue but this impact requires a deeper analysis we will provide in the following section.

In conclusion, Recommender Systems are widely accepted in e-commerce retailers for their various advantages. Among others, they help create faithful long-term customer relationships at the same time that they can be accountable for revenue increases.

4.3 Causal Impact on Revenues

Recommender Systems have a higher impact on the world's best known websites' revenues every day. Nevertheless, part of the accounted traffic would have most certainly also happened through other means in abscence of recommendations. As we saw before, Sharma, Hofman, and Watts (2015) estimate that up to a third of all traffic is generated through recommendations in the present setting but 75% of this activity would still take place if recommendations were absent.

To analyse this effect, we will detail three main points: user penetration, direct impact on revenues and indirect impact on revenues as Dias et al. (2008) suggest, adding some critical discussions over the estimated performances.

User penetration

This is the most simple measure of a RS's value. It doesn't measure how good recommendations are but only if the system is broadly used and accepted by the users. Thus, penetration is understood as the proportion of shoppers having added at least one of the recommended products to their online cart.

In most cases, this ratio is very likely to be overestimating the traffic RS actually generate. From (2015) we want to bring forward a very important point not to overestimate the impact of RS.

The activity caused by RS might not be as straightforward as the first authors presented, number of views coming from clicking on recommendations over the total pageviews. This would be overstate and optimistic since it would attribute all these sales or views to the Recommender System. However, let's consider the recommendation from Figure 4.2: if we were to buy a frying pan and did not have a lid for it, that would probably be our next search after adding the pan to our cart, i.e., we would be looking for it even if we weren't suggested to. In the case where we are suggested to buy the according glass lid, we are saving the time it would take us to find it if we buy it through the recommendation link, but our purchase is not necessarily an output of the RS. Thus, clicking through the recommender might just be convenient but not adding value.

Direct revenues

Under direct extra revenue, the authors define the total amount of money shoppers spent on purchasing items recommended by RS.

In their study, in collaboration with the biggest e-grocer in Switzerland, they obtain an increase in the total monthly turnover of 0.30%, which seems rather small but would be in line with the actual 25% of the newly generated page activity (in its turn, 10-30% of the total traffic) that can be exclusively accounted to the RS.

Moreover, a direct revenue generated in a grocery shop is less relevant in itself than in an electronics or apparel firm, for grocery products are generally cheaper to buy once but are bought regularly, generating indirect revenues as we'll see later.

Thus, for certain products, the direct effect of recommendations can be inflated by up to 200% if we consider all revenues from the user penetration ratio. Instead, the actual click-rate values are around 5% in durable goods while they can be below 1% in consumable

goods. Still, that has a considerable magnitude if we think of the total turnover these kind of companies have.

Indirect revenues

Indirect revenues can be defined as the money spent on items or categories first introduced to a user by the RS but not necessarily in the current session.

And why are they more relevant in grocery stores? As we mentioned, groceries are a regular purchase and it is only when we have these repeated activity that we can account for indirect revenue.

Let's contrast for example smartphones and whole cereals. They are both in now but while we can mainly profit from the use of a single smartphone at a time, we eat several times a day. The first is a durable good, while the second is a consumable good and the price of a smartphone is obvisouly much higher than a pack of cereal. These two products are not comparable in any of this cases and neither are they from a recommender's point of view. If we bought a new device which was recommended to us by a RS, it would only account to its direct turnover. Instead, if the RS notices that we have always been eating wheat and suggests us to try oats, we might buy them and if we like them, add these to our regular diet, purchasing them every week.

So, the second time we buy a product which was first introduced to us by the RS, can still be accounted as its result since we would still not be buying that particular product if we hadn't noticed it the first time.

When the study put together all these sales, the extra revenue generation in a grocery store ranged from 2% up to 26%, even higher than revenues from a single purchase in durable goods.

Summarizing, RS have a considerable impact on a company's turnover, although the calculation isn't always straigthforward. We might need to adjust convenience click-rates and take into account both first and repeated sales to accurately estimate performance.

Chapter 5

Actual Use Cases

Now that we have seen overall use statistics and potential economic impact, we provide a brief analysis on where big companies stand regarding the use of this kind of systems and which are the main techniques out in the market.

RS are widely implemented on the websites of the companies we present and we will see that these companies do not use a single recommendation technique. Details on how four big e-commerce companies have implemented their own RS and the outcomes they have achieved are provided. We break down the different algorithms, detailing what impact each one them has in the different businesses and when they first introduced them.

5.1 Amazon.com

Amazon.com is well-known for personalization and recommendations, Linden and Smith (2017), but even its system needed updates over the two decades Amazon.com has been setting up personalized stores for its users. This said, Amazon.com was the pioneer company implementing RS.

Just a few years after its foundation, Amazon.com started developing its own RS algorithm, which has become the most popular and well-known recommendation technique all over the world: **item-to-item CF**. In the first research paper on it, Linden, Smith, and York (2003), presented the system whose patent had already been filed in 1998.

Back then, many applications only used explicit ratings to represent their customer's interests. Instead, Amazon.com introduced the usage of items viewed, demographic data, click-through and conversion rates, among others. Although our system will not go that

far, it is important to introduce this groundbreaking system which set the foundations for RS and, for our interest, was built on an online bookstore.

5.1.1 System Architecture

The algorithm mainly developed by software eningeers Greg Linden and Brent Smith, takes from: *Traditional CF*, which just aggregates items from similar customers, eliminates items the user has already purchased or rated, and recommends the remaining items to the user; *Cluster Models*, that divide the customer base into many segments to which customers are later assigned; and *Search-Based Methods*, which search for content-related items.

Rather than matching the user to similar customers, item-based CF matches each of the user's purchased and rated items to similar items, then combines those similar items into a recommendation list, Linden, Smith, and York (2003).

Amazon.com's original model used cosine-based similarity to compute similar products in an iterative search over customers with common product pairs. This computation was an extemely time-consuming although it was done offline. However, it reduced from $O(N^2M)$ to $O(NM)$ as most pairs were unavaliable. Furthermore, the authors proposed a further extension sampling only customers who purchased best-selling titles, which highly reduced runtime with little impact on quality.

Notice, that this system was introduced in a time where Amazon.com primarily sold books. Over the years, Amazon.com has expanded its business to almost every retail sector, becoming the largest online retailer in the world, measured by revenue and market capitalization. It now has a catalog with hundreds of millions of items, which has led to the revision of assumptions from the original algorithm. Its authors highlight two main changes:

- **Definition of Related Items**: given two related items X and Y, it was originally assumed that X-buyers had the same P(Y) as the rest. Purchase histories have determined it to be a non-uniform distribution though. To calculate P(Y) over X-buyers aa binomial expansion is now used in order to make the algorithm robust to non-random occurences, which previously biased recommendations to too obvious or irrelevant items.

- **The Importance of Time**: the relationship among purchases heavily depends on their proximity in time. In this sense, some purchases are sequential, going back to our example from Figure 4.2, it wouldn't make sense to recommend the glass lid

before the frying pan. Furthermore, customer's needs and tastes change over time, which makes previous purchases become less relevant but at the same time, can give an insight of their life-cycle and help recommendations in another way. All these elements have been taken into account in further developments.

5.1.2 Outcomes

In the years following the implementation of this RS, recomendations became so extensively used that a Microsoft Research report, which we have already analysed - Sharma, Hofman, and Watts (2015) -, estimated that 30 percent of Amazon.com's page views were from recommendations. The report didn't leave it there though and as we saw in Section 5.2, only part of it wouldn't have occurred in abscence of recommendations.

5.2 Netflix

Netflix's RS became specially famous after the release of the *Netflix Prize*, an open competition to find the best CF algorithm, which started on October 2, 2006 and lasted almost three years.

The competition held by this company had the goal to predict user ratings for films without any futher information about the users and the films, beside the past ratings. The Grand Prize would be granted to the team which achieved a reduction in the Test RMSE of at least 10%. Over the years where no team achieved the goal, *progress* prizes were awarded. The Grand Prize was awarded on September 18, 2009 to a team that had achieved a 10.06% improvement on Netflix's own algorithm, setting their Test RMSE at 0.8567.

While the competition lasted, discussions rose around the evaluation metric and the goal improvement. It was claimed that even a 1% improvement significantly changed user's recommendations, which is relevant for our Thesis in order to determine whether we achieved a significant improvement regarding the baseline model or not.

5.2.1 System Architecture

The Netflix RS presents a very complex architecture built on six different algorithms based on Matrix Factorization. It uses matrix decomposition to derive P and Q which can be

used to make predictions.

Personalized Video Ranker: PVR

This engine, orders the entire catalog, or a subset for a particular genre, for each particular user, which indirectly limits the videos that will be seen by a user since studies have shown that attention is lost after the first 10 to 20 titles. It is widely used over the different sections, which limits actual personlizations.

Top-N Video Ranker

Another specific engine, which is a particular case of the previous one, is the *Top N*. Its goal is to find the best recommendations for a user which will be displayed in the Top Picks row. To do so, it is optimized and evaluated only over the head of the rankings, instead of he entire catalog.

Trending Now

Gomez-Uribe and Hunt (2015) found that short-term trends also affect a user's interest. They identify two particular trends: **periodic** and **one-off**. Repeated trends summarize yearly events or celebrations such as Christmas or Valentine's Day in North America, determining recurrently seen movies. The latter on the other hand, are related to short-term events or news such as natural disasters.

Continue Watching

This engine, which is mainly useful for TV series with several sequential episodes, helping users remember where they left off, is also used for some movies. This continue watching ranker only presents videos that are estimate to still be interesting to the user though, in contrast to other RS which present all unviewed titles that have been previously searched. Therefore, it estimates whether a user will continue watching or stopped because he lost interest using the time elapsed and the abandonment point among others.

Video-Video Similarity

Also called *sims*, this is the only non-personalized algorithm included which focuses on content. This engine computes ranks from each videos content, comparing its features with those of previously seen elements. As it ranks videos compared to a single element, it allows users to have an idea of what they are similar to and form their own expectations.

Page Generation: Row Selection and Ranking

The final engine, which allows Netflix to put together the entire home page estimated which of the precomputed rows represent a user's best choices. Therefore, it uses the output of the other five algorithms in order to construct a single page ensuring a balance between relevance and diversity. The number of displayed rows of each type is completely free on the current implementation, which makes the system more flexible.

5.2.2 Outcomes

According to Gomez-Uribe and Hunt (2015), the RS is directly responsible for about 80% of the hours streamed and also makes part of the results presented by the search engine used in the 20% remaining. However, we need to take into account that the authors do represent the company and as we presented in the previous chapter, causal impact is not so straightforwardly computed.

Regarding diversity, Netflix estimated that the effective catalog size quadruples though personalized recommendations, as it finds niches for generally unpopular videos that wouldn't have made it to the top ranks of a general classification.

Furthermore, improved engagement and take-rates are proven to be related with user retention, which is very important for a company.

5.3 YouTube

Over the years, the world's largest video-sharing website - YouTube -, has developed one of the most sophsiticated RS there exist, Covington, Adams, and Sargin (2016). With its recommendations, YouTube helps its users to find relevant content in an ever-growing video corpus.

According to these authors, YouTube's RS faces three main challenges: the *scale* of their corpus, the *freshness* and the dynamism of this corpus, and the *noise* from the historical behaviour which doesn't necessarily indicate current preferences. In order to overcome all these challenges and present interesing recommendations to the users, YouTube combines several features including their search engine, a front page highlight, and related videos.

This use case is extremely relevant for our Thesis since it is one of the few examples

of RS that use ANN. Furthermore, YouTube's RS was built by Google Brain using TensorFlow, before they open-sourced this technology which we will also be using.

Beside this example work on ANN for RS has been done in few papers, such as J. Liu, Dolan, and Pedersen (2010), Huang et al. (2015) and Tang et al. (2015).

5.3.1 System Architecture

The main structure of the YouTube RS are two sequential ANN: one for *candidate generation* and one for *ranking*.

This two-stage approach allows them to make recommendations from their millions of videos and make sure that the small number of suggested videos are relevant and engaging for the particular user.

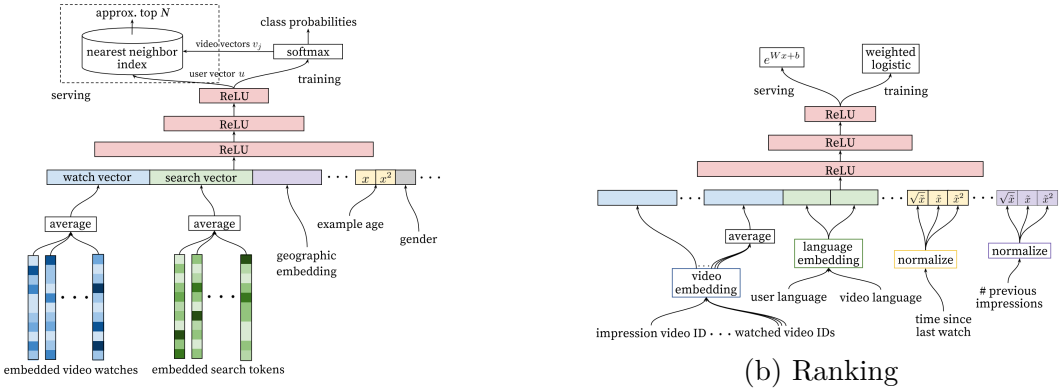
Since there are a lot of avaliable features, ranking the entire corpus would be extremely time-consuming and a previous filter is needed. The first stage in particular, **candidate generation**, uses user-based CF to retrieve a small subset from the full corpus which will be later ranked. That is, potentially relevant videos are filtered according to previous watches, search queries and demographics during this stage. According to its engineers, this system started mimicking the previous matrix factorization approach they had implemented, thus the current setting can be described as a non-linear generalization of factorization techniques.

On the other hand, the **ranking** stage scores all these preselected videos in order to present only a few *best* recommendations, ranked by their score, to the user. Its main role is to summarize all avaliable features in order to calibrate predictions, specially in candidate videos which are not directly comparable either because of their source or their authors. In this stage, some relevant relevant videos with not so appealing thumbnails might come up.

Both stages are built as DNN with several fully connected layers with ReLU activation functions on the hidden layers as represented in Figure 5.1. Their input and output layers differ though.

The **candidate generation** input, as we can see from Figure 5.1a is a numerical vector composed out of three main elements: average video watches, average search tokens and user demographic embeddings with several engineered features. Its output uses a softmax activation function to compute the class probabilities for each video, returning only the top N.

The **ranking** input on the other hand is shown in Figure 5.1b. Again it is a numerical vector with hundreds of video features, embedding categorical features and normalizing numerical ones. Its output uses a weighted logistic activation function to return expected watch time.



(a) Candidate Generation

(b) Ranking

Figure 5.1: Deep Neural Networks Structure for YouTube's RS. *Source: Covington, Paul et al. (2016)*

5.3.2 Outcomes

According to the authors, **candidate generation** improves the offline holdout precision results and increased the watch time dramatically on recently uploaded videos in A/B testing. On the other hand, **ranking**, performed much better on watch-time weighted ranking evaluation metrics compared to predicting click-through rate directly.

In their study, Zhou, Khemmarat, and Gao (2010) estimate that almost 51% of the views of a particular video come from suggested related videos, accounting for about 30% of the overall page views. They also show that the click through rate diminishes as the position in the ranked video list generated by the RS lowers. However, it decreases at a lower rate than in other RS, which would indicate that any of the recommended videos is actually potentially interesting to the user.

Furthermore, they found that the RS helps increase the diversity of video views gaining engagement from their users.

5.4 LinkedIn

LinkedIn, the largest online professional social network, uses RS to promote engagement among people, jobs, companies, groups, and other entities. Therefore, a navigation panel is implemented for each entity type on the site, which that allows members to browse and discover other content, L. Wu et al. (2014).

The original design was intended to showcase co-occurrence in profile views, that is, provide users with similar profiles to those they were viewing. But over time this infrastructure has grown to a full platform called **Browsemap**.

Browsemap is a horizontal platform with mostly shared components. It uses a pipeline to construct co-ocurrence matrices which are later used to compute similarities. All computations are done offline on Hadoop and then forwarded online via API. Thus, it is easy to specify new needs by setting up the location of the input data and needed parameter changes in the pipe. Its authors define three properties:

1. It supports all entity types and creating a new browsemap requires little effort.
2. The platform is flexible to deal with different characteristics, such as the inclusion of expiry dates.
3. It is a scalable system which easily permits the inclusion of new features and products.

5.4.1 System Architecture

The browsemaps use item-to-item CF to build a latent graph of co-occurrences of entities using member browsing history. Its architecture presents two main elements: **offline batch computation** and **online query API**.

Offline Batch Computation

Browsing events are transported to Hadoop via a distributed publish-subscribe messaging system for event collection, Zhuang (2013). The latent browsemap is then computed offline using several techniques to dampen overly correlated individuals and weight newer views more than older ones.

To put together the different characteristics of each entity, a collection of modules was developed to describe how to build a browsemap. Each module performs a particular task, one of the most relevant ones being to remove expired offers. These modules are

internally implemented as a set of Hadoop jobs, where each job produces an output that is the input for the next job.

Online Query API

Latent browsemap graphs are loaded into an open source distributed storage system, Voldemort, Sumbaly et al. (2012). This system provides low response time and high throughput which allow responding to user requests in good time. Furthermore, this API is built to be non-dependant on the entity type, which allows it to load any provided browsemap.

Additionally to the Browsemap, LinkedIn has also implemented different CB filtering engines over time in the "Similar entity" panel.

5.4.2 Outcomes

Initially, some of LinkedIn entities were not compatible with the developed profile browsemap and were better off using CF, although they had computational difficulties with the incrmental data volume. Since the Browsemap platform was developed, all entites and actions have been using it to produce recommendations.

Nevertheless, both CF and CB filtering can coexist on the same page without acting in detriment of one another. Moreover, similarity rankers of these RS can be augmented with additional browsemap elements such as co-occurrences of views, follows, likes, comments and searches, as latent features.

One particular case that had a lot of impact on recommendations was the inclusion of a picture in the member module which lifted click-through rate 50%. This performance increase surpassed any algorithmic improvements by a sizable margin. Additionally, the inclusion of a job browsemap at the end of an application process increased in 500% the job application rate, according to an A/B test split. Both of these achievements have been very important for LinkedIn.

Part II

Implementation

Chapter 6

The Dataset

The Book-Crossing (BX) dataset was mined during a 4-week scrape from August to September 2004 by Cai-Nicolas Ziegler (2005) from the Book-Crossing community with permission from Ron Hornbaker, CTO of Humankind Systems.

The original dataset is structured in three different tables:

- **users:** has 3 variables about **278,858** users (anonymized but with demographic information):
  - UserID: *numeric* with a unique level for each individual.
  - Location: *character* with generally three elements structured as "city, state, country", although some have more and others less.
  - Age: *numeric*.
- **books:** has 8 variables and **271,379** items:
  - ISBN: *character* id with a unique level for each item (6-13 characters, although the standard format are 10).
  - BookTitle: *character* strings.
  - BookAuthor: *character* strings.
  - YearOfPublication: *numeric*.
  - Publisher: *character* strings.
  - ImageURLS: *character* strings. An Amazon link to the small thumbnail image.

  - ImageURLM: *character* strings. An Amazon link to the medium size frontpage image.
  - ImageURLL: *character* strings. An Amazon link to the large size frontpage image.

- **reviews:** has 3 variables and **1,149,780** observations which represent either explicit or implicit ratings:
  - UserID: *numeric* id for the user providing the rating.
  - ISBN: *character* id of the rated book.
  - BookRating: *numeric* rating provided by a unique (UserID, ISBN) pair. Implicit ratings, namely that the book has been read but not rated, are represented with zeros.

This original dataset presents an extreme sparsity, only 1,149,780 out of 278,858 · 271,379 = 75,676,205,182 potential ratings are available, which implies that **99.9986%** of the values are **missing**. Furthermore, out of these available reviews only 37.88% have a specific value (explicit ratings) with the rest being zero, namely implicit ratings. Thus, some condensation steps were necessary.

For our approach, only explicit ratings were kept. After that, we removed books with less than 5 mentions and only members with at least 3 ratings were kept (opposed to the minimum of 20 mentions and 5 ratings, respectively, that were selected for the original work on this dataset).

Thus, the resulting dataset had more moderate dimensions, with **11,573 users**, **10,500 books** and **127,397 ratings**. The latter is smaller than in the original work, due to the elimination of implicit ratings which don't have a specific value. This step **reduced missings to 99.90%**, now having 127,397 out of the 11,573 · 10,500 = 121,516,500 possible. Sparsity still being extremely high will be the first problem to solve.

In this chapter we will talk about the preprocessing and exploitation of present variables as well as about the use of further mining techniques to add additional book taxonomy variables.

6.1 Data Enrichment

In this first section the focus is set on two particular elements: the expoitation of user locations and the addition of book taxonomy.

6.1.1 Geospatial data

Analysing user's locations, we found out that not all of them had exactly "city, state, country". Nevertheless, the city was always in first position and country at the end; and with a given city name, the rest of variables can eventually be recovered. MaxMind for example offers structured databases with cities, coordinates and countries.

We decided to actually keep `city` and `country` as two separate variables. These were split from the original location using a simple regular expression to find the position of the first and last comma in a string. Those users with no text behind the last variable take missing values for `country`.

Further details can be consulted in Appendix Section B.1.

The top 10 most popular cities in the dataset only represent 8.42% of the users, an average of less than 1% of users from each city as we can see in Figure 6.1. Thus, this factor variable will not be of much use for itself. Country on the other side, has two very predominant levels. Figure 6.2 shows the top 5 most popular countries. The USA gather 76.8% of the data, followed by Canada with almost 10%; both being North America, we will be interested in checking if the behaviour in these countries has the same distribution as the rest of the dataset since one might expect differences in tastes over cultures.

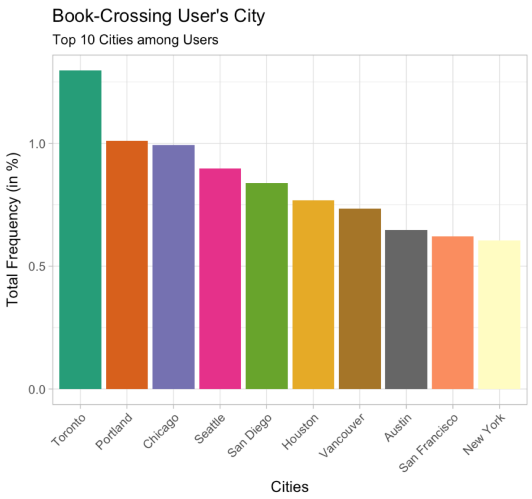


Figure 6.1: Relative Frequences of the Top 10 Cities

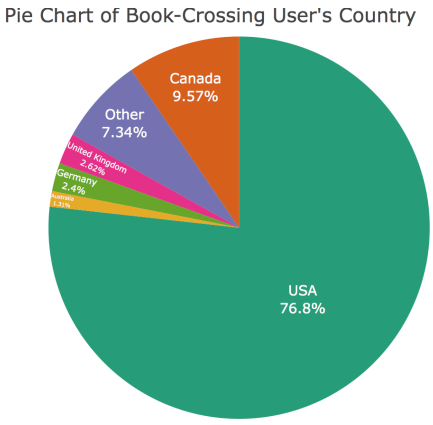


Figure 6.2: Relative Frequences of the Top 5 Countries

Additionally, the user's continents are added using `countycode` with `origin = "country.name"` and `destination = "continent"`.

6.1.2 Webscraping

In order to enrich data with external information, we mined Amazon.com's highes level of book taxonomy and some other content information, such as length, format and recommended ages.

To do that, the DOM parsing approach to data scraping will be used. This relies on the CSS selectors of the webpage to find the relevant fields which contain the desired information. It also requires to previously know the link where the information is.

In `R`, the `rvest` package (2016) is an easy tool to harvest information from webpages. It mainly requires the creation of the html link with `read_html()`. We can then access the CSS path using `html_nodes()` and `[[]]`. `html_text()` is used at the final step to get the text from the latest node, and then `as.character()` or `as.numeric()` can be used to convert it into the desired format.

Let's see an example to parse the genre of a particular book. First, we need to find the specific link were to the desired website. We chose to work with Amazon.com in particular because it has a very structured links which can be reconstructed using the following logic:

https://www.amazon.com/the-book-title/dp/ISBN/

We built these links as a new column in our dataset, `link_string`. Given `link_string`, we then need to convert it into an html link:

```
booklink <- xml2::read_html(link_string)
```

Next, we need to find the CSS path to sequentially access these nodes. The CSS of a website element can be found through `(right-click) > inspect > Copy > Copy selector` in the browser which retrieves something like:

```
#wayfinding-breadcrumbs_feature_div > ul > li:nth-child(3) > span > a
```

To access it in `R` we will also need to define its position in the main page first. Moreover a last to convert it into a new character string will also be required. This translates to:

```
booklink %>%
  html_nodes('div#a-page') %>%
  html_nodes('div#dp') %>%
```

```
html_nodes('div#wayfinding-breadcrumbs_container')%>%
html_nodes('div#wayfinding-breadcrumbs_feature_div') %>%
html_nodes('ul') %>%
html_nodes('li') %>%
.[3] %>%
html_text()
```

Regarding selected variables, we were mainly interested in three:

- **Genre**: it was the most relevant book taxonomy we were missing in the original dataset. We expected people in general to have a preference for some specific genres.
- **Number of Pages**: when recommending a book we found that it might be interesting to know how much time the user wants to invest in this reading. Being able to filter out shorter or longer books could help to do so.
- **Recommended Age**: for children's books it is very important to know for which ages they are recommended. We consider this variable to be the equivalent of genre in adult books but for kids.

Since many books on ont he BX dataset are rare, non-English books, or outdated titles as Ziegler et al. (2005) mention in their work, not all the links we built up worked and not all of them were found. We managed to add up with just two simple functions:

- **99.17%** of the book's **Genres**, thus, we will be using this variable.
- **58.76%** of the book's **Number of Pages**. As it wasn't as much as we expected we decided not to use it for now.
- **65.15%** of the **Recommended Ages** for children's books.

6.2 Data Preprocessing

Categorical and numerical variables require different preprocessing measures. Techniques considered in this project have been the following:

1. **Categorical variables**.
   - Text cleansing
   - Merging small factor levels
2. **Numerical variables**.
   - Outlier detection
   - Re-escalation

Categorical Variables

In the filtered and completed dataset most variables are character strings or factors without a standard notation. Over 10.56% of the book titles are repeated but either written slightly different or published by different editorial companies. Nevertheless, the following title is essentially the same for a user:

BookTitle
Wuthering Heights
Wuthering Heights (Penguin Classics)

Table 6.1: Example of a Repeated Book Title

In order to standardize all character strings, several **text cleansing** steps are applied:

1. All elements between brackets and parentheses are removed;
2. Final spaces are removed and extra whitespaces are stripped;
3. All punctuation signs are removed;
4. The entire strings are set to lower case;
5. Strings are converted to factors.

Regarding genres mined from Amazon.com, 37 different levels were originally scraped with some of them being redundant or having very few observations. Genres are merged into 8 final levels as follows:

Grouped genre	**Original genres**	**Number of books**
literature and fiction	literature and fiction	4,784
mistery and thriller	mistery, thriller and suspense	1,896
kids and teens	children's books, kids and teens	769
sci-fi and fantasy	sci-fi, science fiction and fantasy	550
reference	business, computers, engineering, math, medical, law, politics and education	477
romance	romance and gay and lesbian romance	396
biographies and memoirs	biographies and memoirs	377
other	religion, history, arts, photograpy, entertainment, comics, cookbooks, ...	1,251

Table 6.2: Final Merged Book Genres

To limit the extent of this Thesis, we decided to leave children's books out of scope, whose recommendation process is completely different than the rest since children do not

generally choose their books, nor rate them. These were kept in a separate "Kids & Teens" subset for future work. After applying this last selection, our datasets contain: 11,480 users who gave 119,065 ratings to 9,731 different books.

Numerical Variables

There are two main numerical explanatory variables in the dataset: `YearOfPublication` for the books and `Age` for users; plus the response variable `BookRating`.

After setting zero values to `NA` since they have no meaning regarding the year of publication, Figure 6.3 shows that most books were published in the last decade (regarding the original mining date). Only a few books date from the early 20th century. None of them has extreme or dubitous values though. The oldest novel in particular, is from a Nobel Prize-winning British author, William Golding, published in 1920 and generally well received.

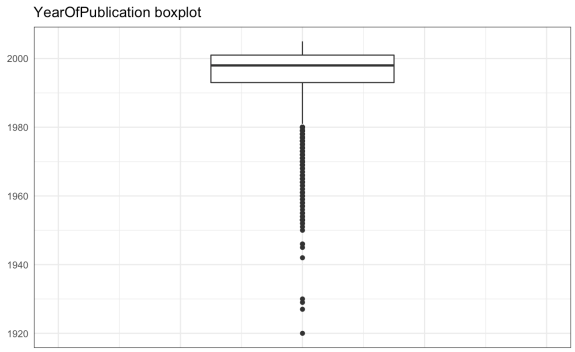


Figure 6.3: Years of Publication

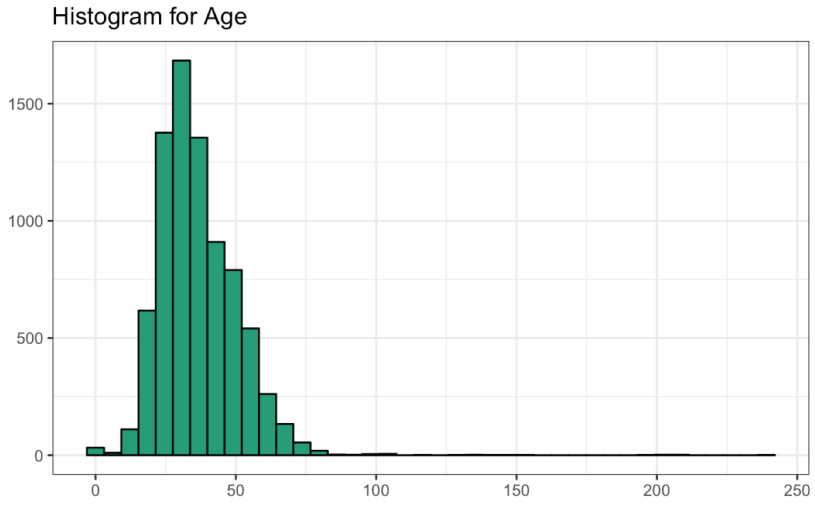


Figure 6.4: User Ages

Ages present a more delicate distribution with unexpectedly low values and impossibly high ones as shown in Figure 6.4. Age is aproximately normally distributed between 0 and 75 years; and values between 75 and 100 are rare but there is no clear evidence to treat them as errors. The world's oldest person is 116 years old which makes higher values impossible and the average life expectancy at birth worldwide is around 70.5 ages. Although keeping high ages, we'll cut at 110 years since users could have been registered some years before the scrape.

On the other hand, there are children's ages in adult books. Manually checking some books read by users under 5 years old, Table 6.3, we observe that these must be all mistaken.

BookTitle	**BookAuthor**
Clara Callan	Richard Bruce Wright
The Kitchen God's Wife	Amy Tan
The Testament	John Grisham
Beloved (Plume Contemporary Fiction)	Toni Morrison

Table 6.3: Sample of Books Read by Users Under 5 Years Old

Nevertheless, books for 5 year olds that were left out in the "Kids & Teens" subset, seemed to be accurate for children who start to read, Table 6.4.

BookTitle	BookAuthor
A Kiss for Little Bear	Else Holmelund Minarik
101 Dalmatians	Walt Disney
Lion King Disney	Disney
Prince and the Pauper Walt Disney	Disney

Table 6.4: Sample of Books for 5 Year Olds

Therefore, ages under 5 were also removed from the initial dataset but children's ages were kept for the "Kids & Teens" subset.

Finally, book ratings also present a particularity. As seen in Figure 6.5, there are values for all possible ratings but on average these are generally high. The average book rating is 7.74 and the median is 8 with 50% of the values between 7 and 9. This values aren't rare though if we think that a user starts to read a book with the expectation that he/she will actually like it, not even choosing undesired books. Notwithstanding, we need to take into account that this can impair learning in the ANN but normalization is not appropriate in this case and neither is rescaling although it was planned to use it, since both books rated with 1 and 10 exist, but are just rare, re-escalation has no actual effect. No further action is taken on behalf of this during the preprocessing stage but it will need to be taken into account when building up the ANN.

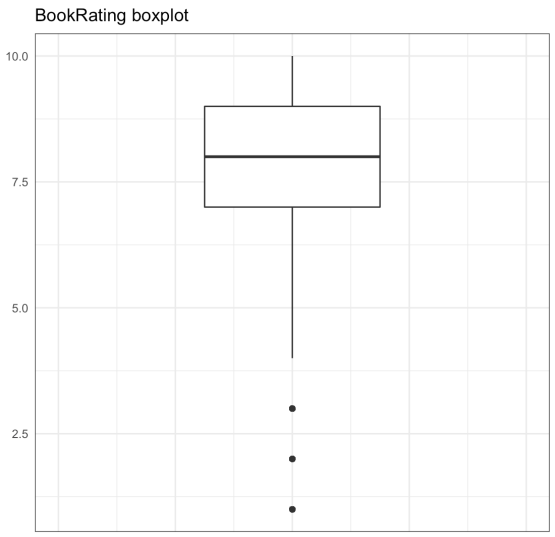


Figure 6.5: Book Ratings

6.3 Dimensionality reduction

Ideally, neither feature selection or extraction would be needed if the algorithm is able to discard irrelevant variables by setting their coefficients to zero. However, high data dimensionality can reduce computational memory and simpler models are not only more robut but also easier to interpret.

There are two main methods to reduce data dimensionality: **feature selection** and **feature extraction**. Hierarchical clustering is used in this Thesis to analyse variable importance to determine which variables to select as well as to determine potential latent variables. Nevertheless, using the full dataset exceeds the RAM limit we have, leaving the R session out-of-memory. Thus a subset will need to be used.

Around a tenth of the users dataset is used to analyse clusters. It contains 1200 users who have done 15185 ratings on 6068 books, an average of 2.5 ratings per book. After several runs with different samples we found that this subset ensures a representative behaviour while not being exessively costly. A hierarchical clustering is done over ratings enriched with user profiles and relevant book information. In particular, seven variables are considered:

- BookTitle
- BookAuthor
- Genre
- BookRating
- Users - Age
- Users - Country
- Users - Continent

Three clearly distinguished classes appear in Figure 6.6.

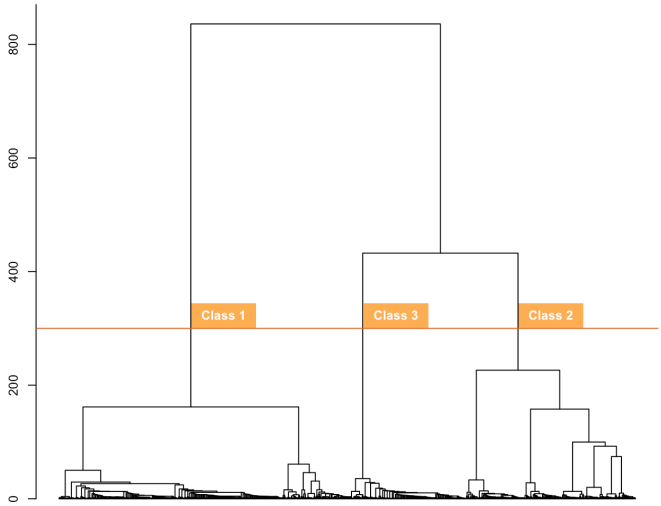
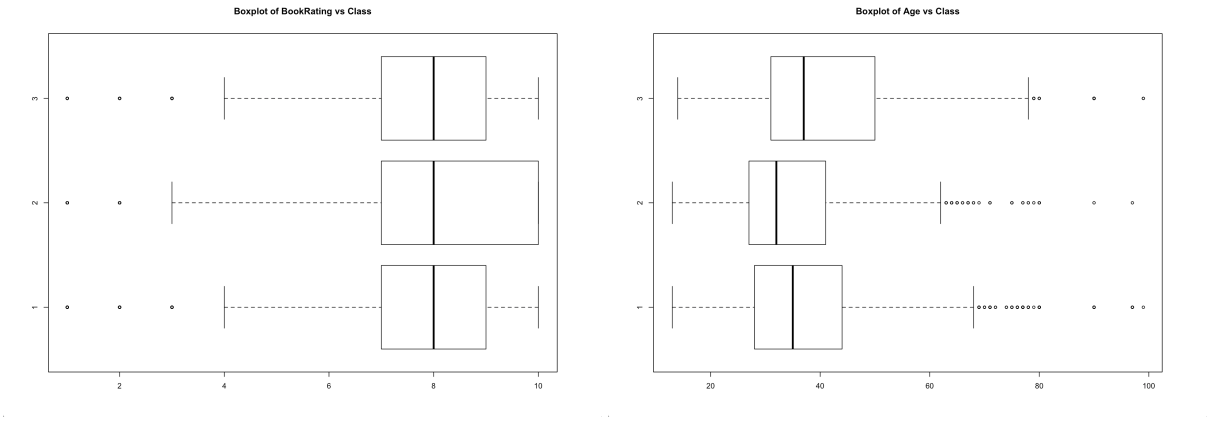


Figure 6.6: Hierarchical Clustering of Ratings with User Profile Information

Profiling the resulting classes, Figure 6.7a shows that there are no significant differences over clusters regarding book ratings and user ages, both with ANOVA p-values which are almost 0.



(a) Conditional Book Ratings Distribution (b) Conditional Age Distribution

Figure 6.7: Cluster Conditional Distribution for Numerical Variables

While this result is interesting for book ratings, because it ensures that these are equally distributed over all groups, it gives us a first indication that `Age` might not be relevant to explain book ratings. Also continents present a similar distribution over the different clusters, Figure 6.7a, with Americas accounting for up to 93.11% of the users in class 3, 91.47% in class 2 and 86.21% in class 1. In the rest of continents, the predominant class is 1. With a 9.35% of the observations, Europe is the second most frequent in this class.

On the other hand, clusters present a very distinct distribution regarding book genres, Figure 6.9. Cluster 1 in particular is 96.65% Literature & Fiction while Mistery & Thriller accounts for up to 99.37% of cluster 3. These two were also the most popular genres in the dataset. Thus, two separate ANN will be built to predict their ratings, while the rest of books will be considered in a general ANN also used for users who don't specify the desired genre.

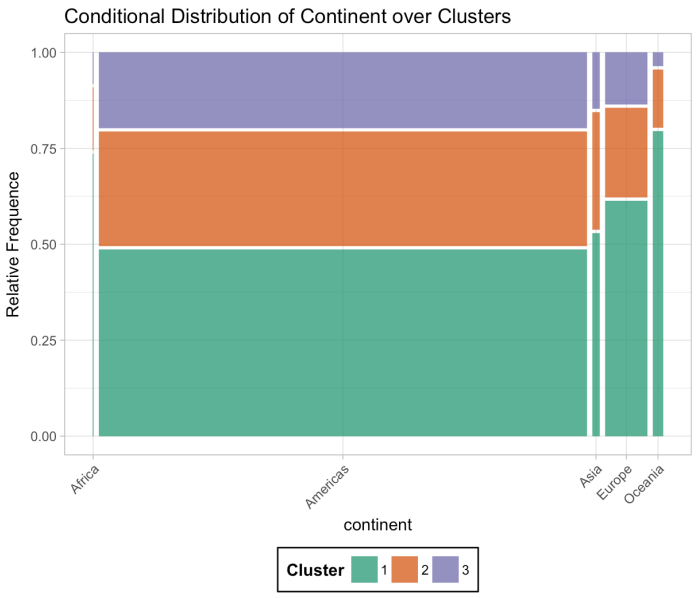


Figure 6.8: Conditional Continent Distribution

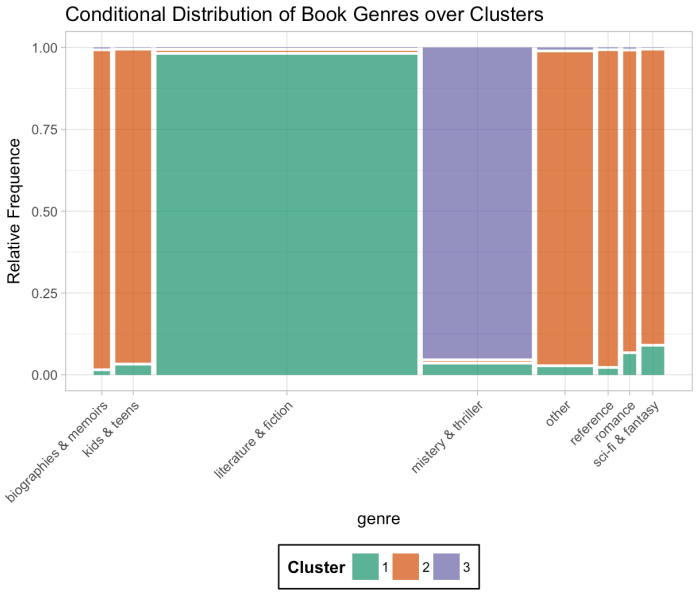


Figure 6.9: Conditional Genre Distribution

Chapter 7

Back-End Development

The system built in this thesis is based on Balabanovic and Shoham (1997), who propose a hybrid recommendation engine that analyses content to define user profiles and identify clusters of similar users. According to the authors, this approach has the advantage of making good recommendations to users that do not share similarity with other users by building their profiles. Additionally, a constrain element will be added to select the genere specifically required by the user.

The final dataset on which we are working contains **11,480 users**, **9,731 books** and **119,065**, with user profiles only having two relevant variables which can be directly used, which limits the possibility of building user profiles.

In the development of this thesis, a UBCF model, implemented with `recommenderlab` is used as a baseline model. The dataset is randomly split 90/10 into training and test samples using the `evaluationScheme` function with `method = "split"` which separates users either into one or the other. The same split will be used for the ANN.

The **training** dataset contains **10,330 users** who have provided **97,694 ratings** to any of the **9,731 books**; while the **test** set contains **1,148 users** and their corresponding **9,756 ratings** to the **9,731 books**.

The baseline RS is built using the `Recommender` function with Pearson correlation as the similarity measure. The following results are achieved:

RMSE	MSE	MAE
2.32	5.38	1.67

Table 7.1: Baseline Model Results

All these metrics have a scaling problem, which means that their magnitude depends

on the units of the variables and there is no measure on how good or bad a particular value is. There is generally no upper bound. Nevertheless, they allow us to compare models with the same input and output variables among them.

MAE has an interpretability advantage over RMSE, since it indicates on average how many points predictions differ from real values. In this particular case, the range of values is also known, ratings always go from 1 to 10, thus this model predicts on average a rating that is either 1.67 points above or below the actual rating. Therefore we will define a custom metric for the ANN based on the MAE, in order to make it comparable.

Three different AE types are built in this Thesis: a **Denoising Autoencoder**, an **Autoencoder with external variables** and **Genre Specific Autoencoders**. All share the same hyperparameter specifications which are tuned on a traditional AE. Keras and Tensorflow for R are used in order to develop the models.

7.1 Experimental Framework

To define the optimal structure of the built AE, we start by setting the specifications of the ANN on which hyperparameters and then parameters are tuned.

When tuning hyperparameters, the focus is set on the validation sample, while the test sample is used to choose the best model.

7.1.1 Model Specifications

Activation Function

Since we are interested in clearly differentiating good and bad ratings without loosing diversity, the **tanh** activation function will be used in this thesis for the all the hidden layers. This is the most common choice in literature for this type of regression problems.

For the output layer, a custom activation corresponding to the normalization and rescalation of the ratings will be used. The following formula is implemented:

$$\sigma(z_i) = \frac{z_i - \min(\boldsymbol{z})}{\max(\boldsymbol{z}) - \min(\boldsymbol{z})} \cdot 9 + 1 \tag{7.1}$$

Cost Function and Accuracy Metric

The problem discussed in this thesis presents a continuous output, so both RMSE and MAE implemented in Keras could be used for the cost function. Pointing to the interpretability advantage described above, MAE loss is used as the cost function and to define the accuracy metric.

Given $x \in [1, 10]$, we can define the upper bound for MAE ($\max MAE = 9$). The custom accuracy metric is defined as:

$$\textbf{Accuracy} = 1 - \frac{MAE(\hat{\boldsymbol{y}}, \boldsymbol{y})}{9} \tag{7.2}$$

This metric defines the percentual accuracy of predictions scaled from 0 to 1. In order to implement it, the cost function will be limited to the known input nodes, $k = i_1, ..., i_d$ where $i_j \neq 0$.

Furthermore, since an AE inputs and outputs the same values, in the testing framework all true values would be known if used as an input of the AE. Therefore a fix proportion of the ratings provided by each individual is masked and the accuracy will only be computed on these.

Optimization algorithm

To explore the effects of using different optimmization algorithms, three of the most popular choices have been tested over 50 iterations on a 3 layer AE with 256-16-256 hidden units.

Adam was found to outperform all optimization algorithms over the 10^{th} iteration. RMSprop performs slightly better than Adam during these 10 first but then slows down improving earlier. Gradient Descent on the other hand, learns

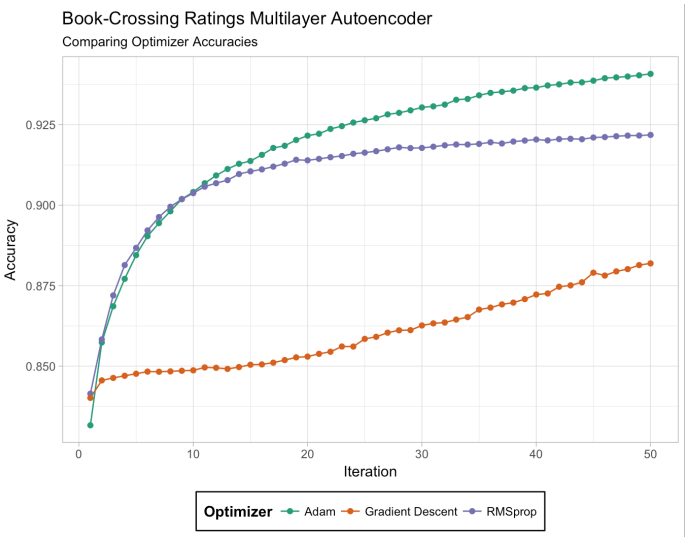


Figure 7.1: Training Accuracy per Iteration on 3 Layer Autoencoders with 256-16-256 Hidden Units using Different Optimization Algorithms

slowlier but is still improving on the 50^{th} iteration which probably indicates that the learning rate not optimal.

Hence, we conclude that Adam is the best optimizer choice for the problem.

7.1.2 Hyperparameter Tuning

Two steps of random search are implemented in order to tune the **learning rate**, α, and the **number of layers**. At each step 50 training interations are performed. The four initial sampled combinations are:

1. **H1**: Learning rate = 0.0005 with 5 hidden layers (128-64-16-128-256)
2. **H2**: Learning rate = 0.001 with 1 hidden layer (64)
3. **H3**: Learning rate = 0.001 with 3 hidden layers (128-16-256)
4. **H4**: Learning rate = 0.002 with 3 hidden layers (128-16-256)

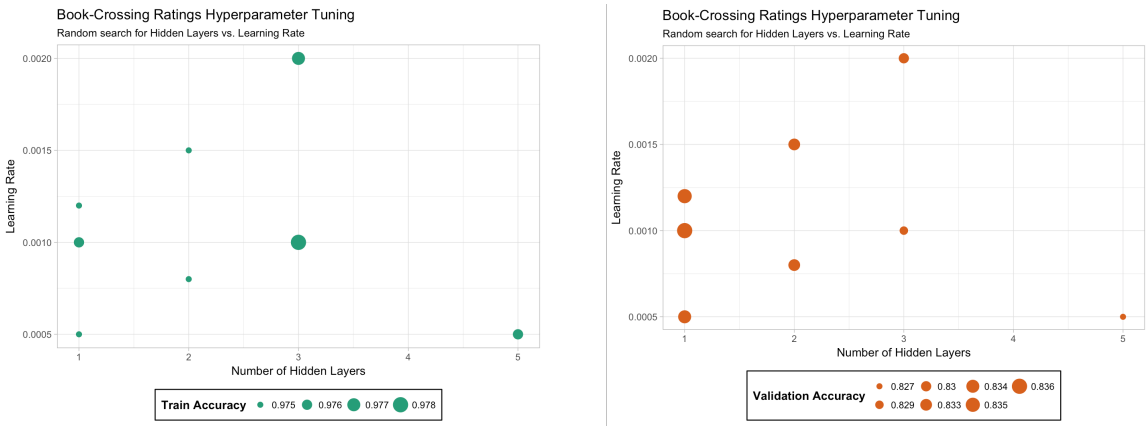
Model **H2** achieves the best validation accuracy as seen in Figure 7.2b. Thus, four additional combinations are sampled around it. In this case, we consider two assymetric AE with 2 hidden layers since actually known values in the input vector are much less than the full output. This should allow the decoder to better learn how to reconstruct ratings. The following combinations are considered:

5. **H5**: Learning rate = 0.0005 with 1 hidden layer (64)
6. **H6**: Learning rate = 0.0008 with 2 hidden layers (64-256)
7. **H7**: Learning rate = 0.0012 with 1 hidden layer (64)
8. **H8**: Learning rate = 0.0015 with 2 hidden layers (64-256)

By just one tenth of difference, **H2** still achieves the best validation accuracy, with all three 1 hidden layer models leading. As all these differences were rather low, a one-way ANOVA was built to test if they actually are significant. Each model is considered a class.

$$\begin{cases} H_0: & \mu_1 = \mu_2 = \cdots = \mu_8 \\ H_1: & \exists \mu_i \neq \mu_j, \text{for some } i \neq j \end{cases}$$

With a `p-value` $\ll$ `0.05`, we have evidence to say that there is at least one pair of



(a) Training Accuracies Grid (b) Validation Accuracies Grid

Figure 7.2: Random Search for Initial Hyperparameter Tuning. Bigger points represent higher accuracies at iteration = 50

models with significantly different validation mean accuracies. To test among which models the differences are, we evaluate all pairwise $(\mu_i - \mu_j)$ comparisons through a Tuckey HSD test, which reveals that accuracy among pairs with a different number of layers presents significant differences, while there is no evidence to determine whether differences among those models with the same number of layers but a different learning rate are. More details on the specific values in Appendix Section A.2.

This said, models with one hidden layer (H2, H5 and H7) present a significantly better accuracy than those with more hidden layers, but it doesn't reveal a better learning rate among these three. Thus, we will keep the most common learning rate, 0.001 which means that the architecture of **H2** is chosen to develop further variations.

Figure 7.2 refers only to the final accuracy. Details on the evolution of the training and validation accuracies over iterations are provided in Appendix Section A.3. As we can see, validation accuracy remains almost constant. Thus, we will stop iterations at 20 from now on.

Next, the number of hidden units are tuned for which four models are trained.

Model	**Number of hidden units**	**Test Accuracy**	**95% CI**	**Total params**
M1	32	0.8310	±0.0073	632,547
M2	**64**	**0.8348**	**±0.0072**	**1,255,363**
M3	128	0.8329	±0.0072	2,500,995
M4	256	0.8340	±0.0073	4,992,259

Table 7.2: 1 Hidden Layer Autoencoders Performance

As we can see, the number of parameters increases exponentially while accuracy

doesn't necessarily improve. According to the test accuracies achieved, `m2` is the best model, but again differences are very small and performing a one-way ANOVA in this case, `Pr(>F) = 0.9066`. Thus, there is no evidence to say that any of these models is significantly different.

According to the **parsimony principle**, we should choose the simplest model but we have to take one more thing into account. As we can see in Figure 7.3, there is a huge gap between train and validation accuracies, while the latter are similar to the observed test accuracy. This is a sign of overfitting and to reduce it, we will incorporate dropout, which can highly reduce the number of nodes in this hidden layer depending on dropout rates. Thus, we will keep a slightly larger model, `m2`, and the largest one we evaluated, `m4`, to test the impact of this dropout on the different sizes.

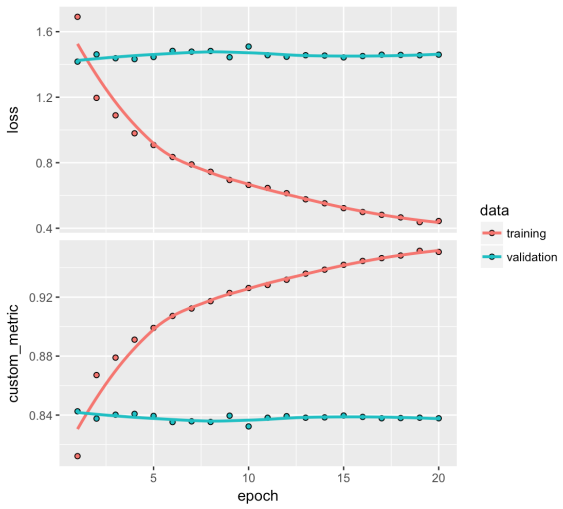


Figure 7.3: Train and Validation Accuracy History for M2

Full history details avaliable in Appendix Section A.3.

7.1.3 Overfitting the Data

When training single layer encoders and decoders, training data can be quickly overfitted even for a small layer size of 32. Dropout is a regularization techinique that randonmly drops out units from an ANN, preventing the parameters to overfit patterns. The dropout rate indicates, the percentage of individual nodes that will not be used to optimize weights in each iteration. These nodes are reinserted to the ANN afterwards with their previous weigths.

Using dropout, the number of trainable parameters at each iteration reduces significantly, although over the full network these remain the same as all weigths are trained at some point.

Three dropout rates are introduced on the two previously best fit models, achieving the following results:

In both cases, test accuracy slightly improves as dropout rates increase, with `dropout = 0.65` achieving the best test results in both cases. Train accuracy is slightly reduced,

Model	Dropout rate	Train Accuracy	Test Accuracy
m2	0	0.9507	0.8348
m5	0.4	0.9399	0.8349
m6	0.5	0.9351	0.8355
m7	**0.65**	**0.9245**	**0.8376**

Table 7.3: Performance Improvement Adding Dropout to the 1 Layer Autoencoder with 64 Hidden Units

Model	Dropout rate	Train Accuracy	Test Accuracy
m4	0	0.9488	0.8340
m8	0.4	0.9472	0.8350
m9	0.5	0.9438	0.8322
m10	**0.65**	**0.9381**	**0.8377**

Table 7.4: Performance Improvement Adding Dropout to the 1 Layer Autoencoder with 256 Hidden Units

which slims down the overfitting gap. Furthermore, dropout improves the accuracy on the 256 hidden units AE slightly more than for the 64 hidden units one where differences, although still not statistically significant, have a much lower `Pr(>F)`.

Since computational cost of the best fitted model is not higher than the cost of the simplest model, `m10` is used as the baseline AE.

7.2 Model Variations

Three main variations are developed over the `m10` model in order to try to improve accuracy further. Avaliable data and the clustering results are used to define these.

7.2.1 Denoising Autoencoder

The reconstruction criterion alone has been unable to fully extract useful features. Even after applying dropout, parameters overfit the training dataset and accuracy on the masked inputs is not presumably high. Here we present the application of a new strategy focused on not only masking but corrupting the input. In doing so, the AE is forced to denoise information recovering only relevant data. According to Vincent and Larochelle (2010) this approach has two underlying ideas:

- A higher level representation should be rather stable and robust under corruptions of the input.

- Second, it is expected that performing the denoising task well requires extracting features that capture useful structure in the input distribution.

The input is corrupted using a gaussian or white noise $\varepsilon \sim N(0, 0.0001)$. This is provided as the first layer of the AE. Keras has a predefined layer in order to do so:

```
model <- keras_model_sequential()
model %>%
  layer_gaussian_noise(stddev = 0.0001, input_shape = c(n))
```

Test accuracy is slightly improved up to 83.92%.

7.2.2 Autoencoder with External Variables

Next, we tried to add user's information to the defined Denoising AE although the previous clustering did not reveal a strong relationship. An external matrix is created with the following variables:

$$Age + Age^2 + America$$

The latter is a binary variable indicating whether the user is from America or not. When adding this external matrix, two input layers are built and then concatenated as follows.

```
model <- keras_model(
  inputs = c(input, aux_input),
  outputs = output)
```

This increases the number of parameters up to 5,168,221. This increase, added to the memory increase of loading the full external matrix generated an out-of-memory problem in our R session. Therefore, we had to change the approach when training the model. Instead of loading the entire dataset - with both the rating matrix and the external matrix - into the session, a **sampling generator** is created. This generator only loads to the memory a random sample of the training set with the corresponding mini batch size. The following code presents an example sampling generator:

```
sampling_generator <- function(sample_name, batch_size) {

  filename = paste0("data/fit_", sample_name, "_input.csv")
  aux_file = paste0("data/fit_aux_", sample_name, ".csv")
```

```
  data = fread(filename)
  aux_data = fread(aux_file)

  function() {
    rows <- sample(1:nrow(in_data), batch_size, replace = TRUE)
    list(list(as.matrix(data[rows,]), as.matrix(aux_data[rows])),
         as.matrix(data[rows,]))
  }
}
```

Working with a sampling generator allows us to train bigger models but is much more time consuming and computationally costly. Testing the generator on the traditional model it increases computation time up to 150%.

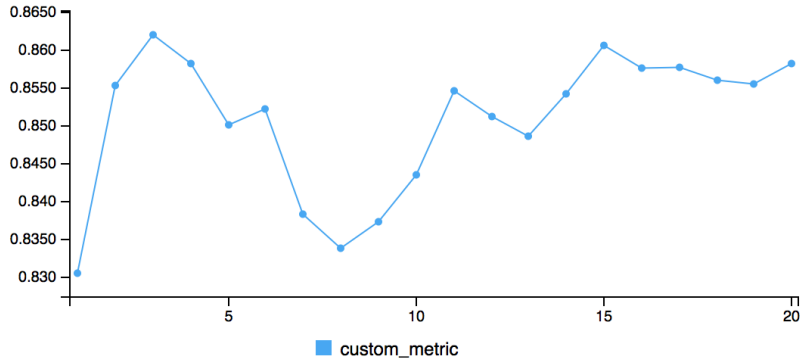


Figure 7.4: Training Accuracy per Iteration and Mini Batch on 1 Layer Autoencoder with 256 Hidden Units, Gaussian Noise and External Variables

In this case, as random samples are taken at each step, the accuracy doesn't improve linearly as shown in Figure 7.4. The model is very sensitive to the selected random sample and over 20 iterations no stable results have been achieved. But there hasn't been an accuracy peak up to the previously best training accuracies either.

The model achieves a test accuracy of 75,08%, far from the previous models at a much higher cost. Thinking on the final implementation which takes place in a dynamic environment, we cannot have such a time consuming model, which would lose user's attention.

7.2.3 Genre Specific Autoencoders

From the hierarchical clustering performed in Section 6.3 we highlighted a differentiated behaviour in two specific genres: Literature & Fiction and Mistery & Thriller. Here two

constrained models are built to see if we can improve accuracy on these subsets.

Both models were built using the optimal AE structure defined in Section 7.1 with Gaussian noise but only the subset for each genre is considered in these two specific AE. This reduced both the number of input and output nodes and thus, the number of trainable parameters. Nevertheless the expected accuracy increase was not achieved. This might be due to the simulatenous reduction of the training sample, since not all the users have rated book in these categories: 73.29% have rated at least one Literature & Fiction book but only 34.77% have rated a Mistery & Thriller book.

We considered retuning the number of hidden units in the layer but experiments determined that 256 was still the optimal size. Using this setting, the following results are achieved:

	Denoising AE	**Lit. & Fict. AE**	**Mist. & Thrill. AE**
Train Accuracy	0.9396	0.9226	0.8969
Validation Accuracy	0.8409	0.8304	0.8420
Test Accuracy	**0.8392**	0.8356	0.8329
Test 95% CI	±0.0071	±0.0077	±0.0095

Table 7.5: Performance Comparison for Genre Specific AE

We can observe that the CI is wider the more specific the model is and less books it comprises. Again, as differences are not statistically significant, we conclude that using the general Denoising AE to predict ratings for all genres and filtering a posteriori will ensure the best performance for the chatbot.

7.2.4 Results

We started by tuning two hyperparameters, namely the number of hidden layers and the learning rate. While the first showed significant differences in the validation accuracy, the second didn't and we opted for its default value in `R`, 0.001.

Next, the number of hidden nodes were also tuned. As for the learning rate, no significant differences were found and we opted for larger but not computationally more costly models on which dropout was applied to reduce overfitting.

Four models have been trained on the full dataset having more or less success predicting ratings. The Denoising AE has achieved the overall best performances.

Furthermore, we did not fins statistically significant improvements in the accuracy of genre-specific models and decided not to implement them separately to guarantee a better

	Baseline Model	Baseline AE	Denoising AE	AE with External Vars
Train Acc.	-	0.9381	0.9396	0.8582
Validation Acc.	-	0.8409	0.8409	-
Test Acc.	0.8143	0.8377	**0.8392**	0.7508

Table 7.6: Performance Comparison for General Models

performance in the chatbot.

Last, we analysed the errors of the best fitted model as $(\hat{\boldsymbol{y}} - \boldsymbol{y})$. These errors are randomly distributed with its 95% CI being (-0.1929, 0.0161). This interval includes the 0, which means that we do accurately predict some values but we can observe a slight assymetry around 0. This tendence for negative values means that we are generally underrating books. Nevertheless, we consider that a more appropriate error measure should be implemented, as predicted values itselves are not as much relevant as the final order.

7.3 Filtering and Recommendation Phase

7.3.1 Collaborative Filtering

CF has been the common link element through all the models. Similarity over the past rating history of users is calculated on the baseline model using Pearson correlation, a memory-based technique. On the other hand, AE apply model-based CF which guarantees higher diversity.

Some of the advantages and disadvantages mentioned in Section 1.2.1 have been proven. Regarding efficiency for example, the baseline UBCF model is trained in just 0.05 seconds while the Denoising AE takes up to 5.25 minutes. Nevertheless, training is an offline computation in the final application and focus should be set on the prediction time. Here, the baseline UBCF suffers from a much higher cost taking up to 20.99 minutes to predict the entire test set, which the Denoising AE does in just 0.01 seconds.

7.3.2 Knowledge-Based Filtering

Constrained-based recommendations are applied if the user specifies a particular genre. In this sense, only books meeting user's requests are recommended, no matter if books out of

this subset have higher predicted ratings.

For our analysis, each book is mapped exclusively to one general book taxonomy. Thus, when the user specifies a genre through the front end application, the `genre` variable is simply used to filter ISBNs in scope.

7.3.3 Content-Based Filtering

User's age and location have not only proven not to be relevant in the present setting but to increase computational cost, slowing down learning and decreasing accuracy if the algorithm is stopped beforehand. CB filtering is not applied in the final model.

7.3.4 Hybridization

Cascade hybridization is applied to obtain personalized recommendations for each user, although further hybridization techniques which could improve recommendations even more are discussed in Future Work. The cascade method is applied sequentially as follows:

1. First, the full ratings are predicted using the model-based CF technique, namely the Denoising AE;

2. Then, elements are filtered according to user constraints.

Selected elements are then are sorted according to predicted ratings and only top items are presented to the user.

7.4 Setting Up Accessible Files and Models

Our RS presents a particularity over all discussed implementations in chapter 5. It works in a dynamic environment where recommendations are specifically asked for, being the main element of the interface. In this first version of the chatbot in particular, users would be newly logged in each time they start the Bot. Thus, we are always on the cold-start edge which is overcome asking for specific ratings and using a pre-trained model.

In order to have this model avaliable we need to set it up in an accessible format for the chatbot. This specifically applies to: all **data files**, the **predictive modeling engine** and the **recommender engine**. In this section we detail how all the datasets

and files needed to run the chatbot are structured.

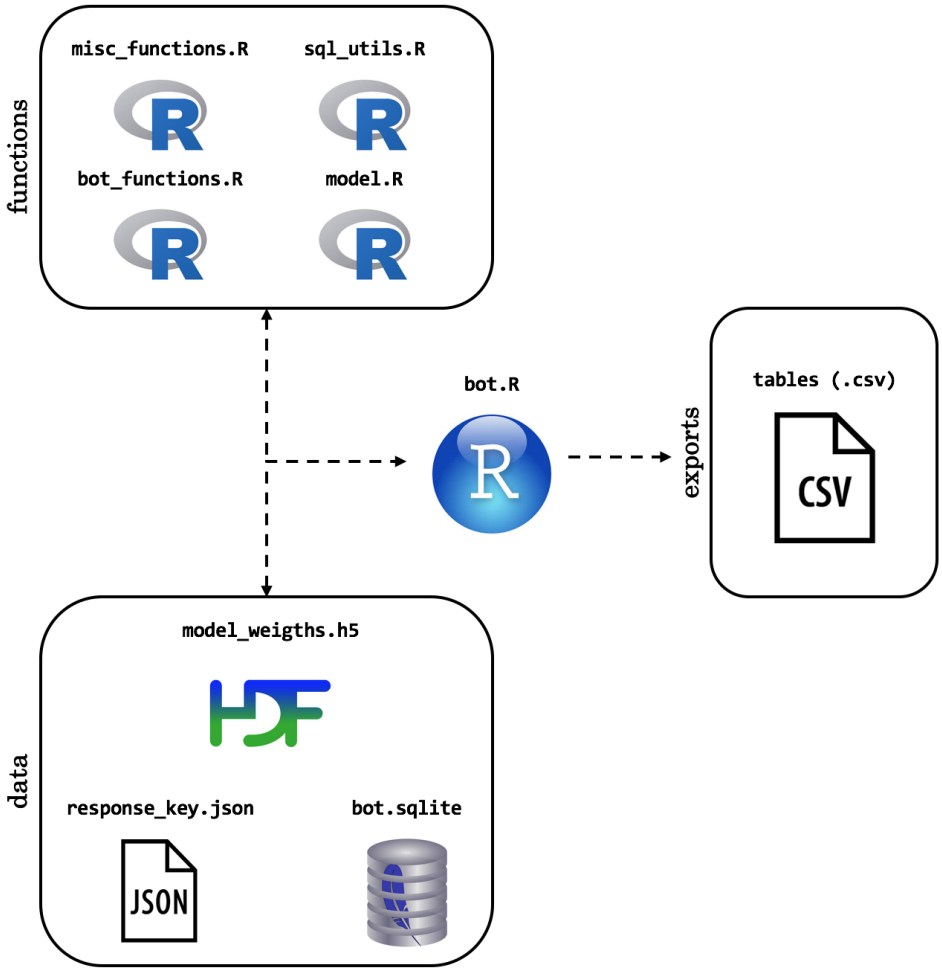


Figure 7.5: Diagram of the Data Structure

7.4.1 Data Structure

There are several tables that must be accessed or updated while the Bot is running.

To do so, we decided to manage our DB through the R package RSQlite. This package embeds the SQLite database engine in R, providing a DBI-compliant interface. SQLite is a public-domain, singleuser, very light-weight database engine that implements a decent subset of the SQL 92 standard, including the core table creation, updating, insertion, and selection operations, plus transaction management, Mueller et al. (2018).

There are three main elements that needed to be stored: **top picks** both for all genres as for each specific genre, **book informations** and the **model input order**.

7.4.1.1 Top Picks

A subset of the Top 20 books for each genre and a global subset are generated using a weigthed average among **popularity** and **average rating**. Popularity is defined as the number of users who have given an explicit rating to the book scaled from 1 to 10 in order to match the rating scale.

$$BookValue = 0.6 \cdot Popularity + 0.4 \cdot AvgRating$$

We decided to give a slightly lower weigth to average ratings because of their litle variablity, described in Section 6.2.

The generated tables are stored in the SQLite database using the convention `genre_top` (with `genre = 'all'` for the global Top 20).

7.4.1.2 Book Informations

Only a small subset with the necessary variables of the complete `books` table was uploaded to SQLite. We decided to reduce its dimension because the table has to be accessed every time a user gets to the end of the personalization phase and runtime is a very precious element when working in a real-time environment. On the other hand, no books were removed since all of them can be potentially recommended to a user over time.

7.4.1.3 Model Input Order

In order to ensure that the model can be executed accurately, a small two column table was developed; one column indicating the order and the other the corresponding ISBN. By only storing these two we minimized not only storage memory used but also computation time when accessing it. The reason the order wasn't included in the previous book information table is that both actions are accessed at different time points, thus, loading unnecessary variables before computing predictions.

In R, each numeric id used 48 bytes of memory and the ISBN string 104 bytes.

7.4.2 Predictive Modeling Engine

There are two main ways to store a pretrained Keras model:

1. **Storing the entire object** using the `keras_save()` function and then reload it through `keras_load()`.

2. **Storing the model weights** and the **model structure** as two separate objects. The first is stored using `keras_save_weights()` and then reloaded with `keras_load_weights()`. The latter can be saved as a .json, .txt or .R plain script.

Although the first option seems slightly simpler it has the disadvantage that the full object takes a lot of memory. Thus, loading it can also be time consuming, which would strongly work against our application. To reload a model in the second format we just have to make sure that the model structure is read into the environment before loading the weigths on top of it.

For this Thesis, the latter option is used for its higher efficiency. This implies that we have two files currently stored in a local environment that need to be moved to the cloud server for the bot to be able to run uninterruptedly:

- **model.R**, the plain text file with the model architecture; and

- **model_weigths.h5**, a hierarchically formatted data file which contains the multi-dimensional array with the weigths.

Having both files in the environment, a new prediction can be retrieved instantaneouly by passing a new vector into the `model %>% predict()` pipe. We use the model input order table to make sure that provided ratings are introduced into the correct input node.

7.4.3 Recommender Engine

Given the predicted output vector, which comes in the same input order, some quick sequential step are taken to retrieve final recommendations:

1. Add corresponding ISBN and join Book Informations through it;
2. Filter out items for which an explicit rating has been provided;
3. If required, filter specified genre;
4. Sort list by predicted rating;
5. Return top rows with all the required information to the Bot.

Chapter 8

Front-End Implementation

The front-end solution developed for this Thesis is a chatbot called Allyn. Although chatbots were not our first option we opted for this approach as soon as we found out about them because they offer a novel real time solution which could change the traditional vision of RS, implemented on static websites and, at the same time, give chatbots a new purpose which hasn't been explored much.

There are may things to be taken into account when building a chatbot but above all, the tool must ensure a friendly and useful interactions with its users. In this sense, the tool must be adapted to the target users, meeting all requirements at the lowest cost possible.

Miller (1968) analyses the response time in this type of interactions to determine that up to 0.1 seconds are the limit for a user to feel it as an instantaneous reply but up to 1 second will still keep the conversation flow uninterrupted. We will need to take this into account if we want users to be comfortable using the tool and feel like they have the Bot's attention. Nevertheless, up to 10-15 seconds might be acceptable when computing complex replies such as the recommendations, but the user needs to get a notification of the underlying process taking place.

Furthermore, a chatbot doesn't interact with a single user at a time, thus, it's implementation should be scalable to an increase in target users. Nevertheless, our chatbot is a Proof of Concept which still doesn't have defined target users. Therefore, we will focus on programming it as efficiently as possible but we will not be able to test all its limits.

Without having further specifications on the target groups, we decided to set the Bot's inteface language to english, which would allow us to potentially make the tool avaliable for more bookstores.

8.1 Chatbot Creation

The first decisions to make when building a chatbot are two:

1. **Choosing the programming language**

   There are many different programming languages that allow us to create chatbots such as Python, Clojure, PHP, Java or Ruby, Miteva (2017). Some of these are faster than others but we had to take into account that efficiency also depends on the quality of the code. After testing Python for a while, we decided to develop the chatbot in **R** where the tools for building a chatbot, although scarcer, were also avaliable. R had been the programming language used to train the models, thus this would also avoid efficiency losses from the translation of commands from one language to the other.

2. **Selecting the App channel**

   On the other hand, we had little knowledge of the potential front-end applications where a chatbot could be integrated. But one thing was clear, it had to be a commonly used channel, supported by mobile and/or computer devices, which would remove initial barriers to actually implement the tool in a bookstore. Our first contact with chatbots had been over Telegram but no current enterprise use cases had been found (see Section 3.2). We decided to look futher to see whether Telegram was actually a valuable option or platforms such as Messenger should be used. Afther reading both about avaliable platforms, Brisson (2016), and about Telegram specifically, we found some key facts about the latter that led us to the decision to use **Telegram** as the front-end channel. These are:

   - Telegram is well known for its **focus on privacy** which is specially **relevant** when we think that potential applications include **private company's informations**. In particular, it allows users to define secret chats which stored on the device, not in the cloud and can be set to destruct after a certain time.

   - Although it is not the most popular platform, it is **one of the most common messaging apps** both worldwide and in Spain with over 62M active monthly users.

   - Telegram has a **specific feature** which allows users **to make secure payments** over the App which would be a key feature in a final client application.

   - Telegram Messenger is not only **accessible on multiple devices** (included mobile, computer and tablet which were on our scope) but also on **multiple**

platforms such as Andoid, Microsoft Windows and macOS, which **ensures** potential client **compatibility**.

- Telegram has an official API to create chatbots.
- Two different **R packages** to communicate with telegram and build chatbots are already **avaliable**: `telegram` and `telegram.bot`. The latter was created at the beggining of this year by a student of our same university from whom we had had the first references about chatbots. This facilitated a lot, what would be our first development.

8.1.1 Setting up the Chatbot

The official way to create a chatbot couldn't be another than a chatbot itself, namely the *BotFather*. To create a Bot, the developer just needs to talk to BotFather and follow a few simple steps, Telegram (2015). To start, two parameters have to be provided:

- **Name**. This is the name which is displayed during the conversation. Our Bot is called **Allyn**, a diminutive for Alan and more specifically in honor of Alan Turing, considered the father of modern computation who laid the foundations for AI. We found the name convenient for a chatbot, specifically concerning the Turing Test.
- **Username**. This username is the id name through which bots, as users, are found over Telegram. In the case of bots it must end with "bot". We set it to **booksRS_bot**.

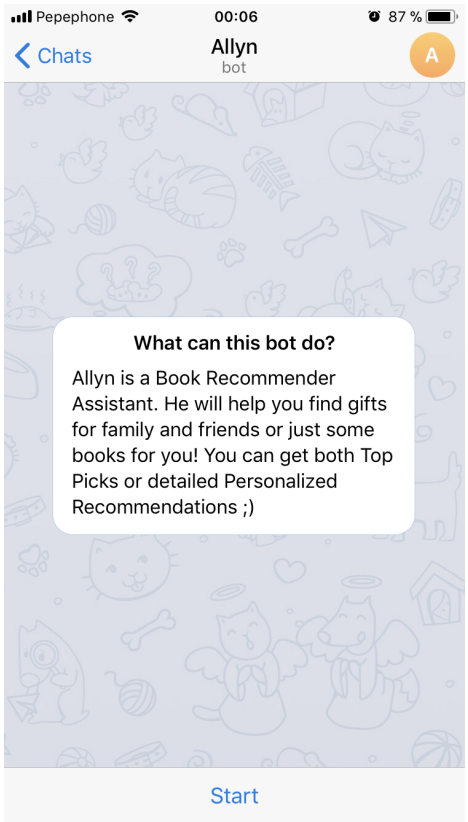


Figure 8.1: Bot Initialization Page

Once a bot is created, the developer recieved an authorization token, required to authorize the bot and to make the HTTPS requests to Telegram's Bot API. It is used to control the bot, thus, it must be kept private but it should be something along the lines of `110201543:AAHdqTcvCH1vGWJxfSeofSAs0K5PALDsaw`.

8.2 Bot API Requests

The `telegram.bot` package is built using Long Polling (LP). LP is a variation of the traditional Polling, a techique which retrieves updates by sending an HTTPS GET request to the Bot API, specifying the Bot in scope, through the `getUpdates` method.

In LP, no notification is sent while there aren't any updates, the connection is simple kept open instead. The connection can't remain open forever though. In this sense, it stays until the URL status changes or a specific time passes. This time lapse is specified through the `timeout` argument. Thus a chatbot created using this package will be active through a main loop over the following steps:

1. **Calling 'getUpdates'**. HTTPS requests to Telegram using LP are done through this method. As the Bot is still on a Beta Phase and no one has access to it, the 'timeout' argument is set to 10 seconds in order to minimize requests but keeping 'start' replies under an acceptable time for human attention.

2. **Receive Answer**. When a user presses a reply button, the status of the URL changes and thus the actual GET request is done, retrieving the message update information.

3. **Process Answer**. The selected option is retrieved using `update$callback_query-$data`. This option is then processed and the corresponding branches of the Dialog Tree are activated.

4. **Send Response**. The generated answer is sent to the user through another HTTPS request sent to Telegram, specifying the user id to whom the response is addressed.

5. **Back to 1**

Further information about the package in its CRAN Reference Manual Documentation, Benedito (2018).

8.3 Answer formats

A rule based classifier is used to process all answers. In this sense we have implemented a fully guided dialog that follows the structure provided in Figure 8.2.

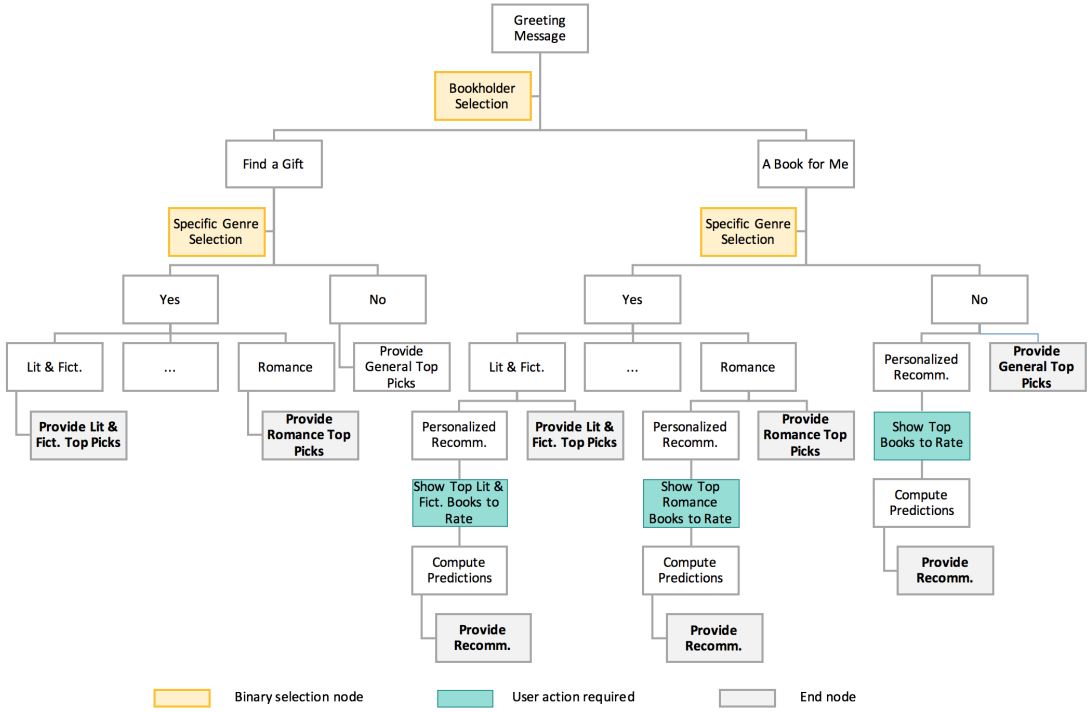


Figure 8.2: Bot's Guided Dialog Answer Tree

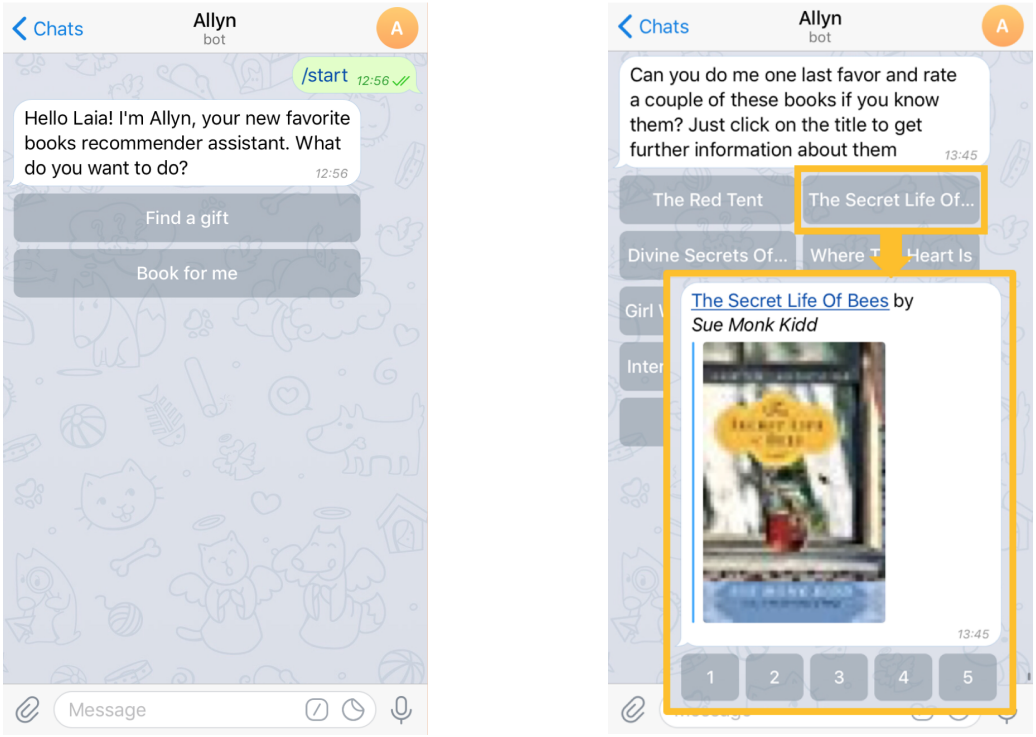
One can observe that there are mainly two types of end nodes but messages will depend on who the bookholder is and provided results vary according to specified genres. The structure is implemented so that these two nodes are only defined once. Using a variable answer code of the type `genre_bookholder`, the different messages from the `response_key.json` and `SQLite` tables are accessed.

On the other hand, both binary selections and required actions are predefined using `ReplyKeyboardMarkup` objects with the reply options as shown in Figure 8.3a. When a user is asked to rate books, an initial set of the 8 most popular titles for the selected taxonomy is presented and when a specific selection is made, more detailed information is provided. As shown in Figure 8.3b. This list is updated each time a user provides a rating.

8.4 Retrieving Recommendations

Users are allowed to rate as many books as they like up to a total of 20. At each step, the new rating and its corresponding ISBN are appended to a list which is set to a sorted vector at the final step.

In the current setting, the model is loaded into the environment the whole time but should be moved to an external server from which the bot would also run. This allows us



(a) Binary Reply Keyboard Markup Example (b) Detailed zoom-in to a Selected Book to be Rated

Figure 8.3: Bot Reply Keyboard Markup Examples

to directly pass the column vector into the model and return the top picks in less than 0.25 seconds.

Top Picks are formatted in `parse_mode = 'Markdown'` which allows us to include hyperlinks. We present: the book **title**, the **author** and a **link** to the specific books in an online bookstore of our choice for this Proof of Concept.

Conclusions

Chapter 9

Conclusions and Future Work

During this project many things have been learned but among all, we take away a global overview on Recommender Systems and more specifically on Autoencoders for Recommender Systems. In this chapter we expose the overall outcomes that have been achieved and propose some future work in which we have already started to work.

As many obstacles have been faced during the development of this Thesis, we present many key take-aways that can help people willing to develop similar projects by highlighting the most relevant contributions.

The final outcome of this project has covered all the initial goals that had been set:

- A detailed chapter on **Recommender Systems** has been developed. Part I, presents a full economic impact analysis to cover their penetration in **e-commerce platforms**.
- Throughout the Recommender System chapter, details on the most common dataset structures are provided and a thorough introduction to ANN, including an analysis on the common ground shared between ML and Statistics, points out how these ML techniques can overcome challenges of traditional Statistics.
- There were large amounts of information available on **Artificial Neural Networks**. To get a better insight on their algorithms and be able to develop one of our own, we had to summarize relevant information from many different documents, which was a slow process.
- A Denoising AE able to retrieve rating predictions has been developed.
- Chatbots on the other hand, were much less documented and there were barely

use cases of Recommender Assistants.

- A Chatbot for Telegram, Allyn, has been set up. It is able to retrieve basic recommendations, although communication is still limited.

9.1 Conclusions

The key take-aways from this project can be split into two main parts, namely the back-end and the front-end.

On the **Back-End Development**, we highlight:

- Predicting ratings using Machine Learning techniques is very sensitive to data and might be unstable. Our dataset wasn't centered, presenting a higher prevalence of high ratings. This has led to some problems during the optimization of the Artificial Neural Network. Quality data is key on developing any system.

- Besides quality data, we have faced some problems due to the enormous amount of missing data (99.90% of the ratings). It was our main problem to solve and we have provided a solution with 83.92% of accuracy but we consider that it could improve much further with just a little bit more information or using another hybridization technique such as Feature Augmentation. One of the key elements that could improve these recommendations alone is a temporal reference on when each rating was provided.

- Model-based Collaborative Filtering has proven to be much more time efficient than Memory-based Collaborative Filtering. Nonetheless, it has a relevant drawback when it comes to incorporating new items as the entire model has to be retrained. We present a further alternative whose potential is still to be determined in Section 9.3 to develop a slightly more complex algorithm able to deal with it more efficiently.

- Creating a rich Recommender System does not only require quality data but also high computation ability available. The number of parameters in Artificial Neural Networks grows exponentially when new nodes are introduced. This led to the need of creating both a sample and a predict generator in order for our computer to be able to process all information without running out-of-memory.

On the other hand, regarding the **Front-End**:

- We have been able to implement this system into an actual text messaging application which gives us the opportunity to reach out to a lot of users. A Telegram Bot has been implemented with a friendly user interface and fast replies. The solution is scalable to different e-commerce businesses with enough data availability to retrain the model.

- Implementing a chatbot is not as straightforward as one may think. Many different things were taken into account when developing this front-end application. Data storage has to be minimized and set up in an accessible format and functions have to be connected to the server.

Thus, many things took a higher effort than was initially expected, such as the sampling generators needed to overcome out-of-memory problems which were hard to identify and collapsed progress for some weeks. These have introduced an over-cost along the entire project.

Furthermore, a lot of autonomous learning has been done during the development of the project, which has provided a deeper knowledge about several fields and it is not always easy to identify relevant information on its own.

Looking back, an entire Bachelor's Thesis could be done just on Artificial Neural Netwoks or other algorithms for Recommender Systems, and another one for the implementation of a Chatbot to provide users with an even better experience. We have not been able to develop all these aspects at a full level of detail, but a great effort was put to achieve a complete end-to-end solution, for which we are very satisfied. We also hope that this work opens up the doors for this innovative solution for Recommender Systems.

9.2 Contributions

With this project, two main contributions are provided.

- Regarding **Statistics**, we have laid the foundations for an innovative approach on rating predictions and further work is already being tested. More details in Section 9.3.

- Regarding **Economics**, an end-to-end solution is provided ready to be implemented in actual online bookstores or even in any other e-commerce business. This solution mainly requires booklinks to be updated to the corresponding bookstore and models

need to be updated using more updated data. Further work on scaling the solution is also being developed.

9.3 Future Work

Taking into account the different contributions made and problems faced, we consider that the present project opens the doors to three main developments: **ANN for Feature Augmentation**, **Improving the Chatbot Experience** and **Scaling the Solution**. As work on these items is already being developed by the author of this Thesis, we here present some further details.

9.3.1 Artificial Neural Networks for Feature Augmententation

In this project, the different filtering techniques have been implemented sequentially, which is called Cascade Hybridization. Nevertheless, we have found ourselves stuck around 83% of accuracy. Then, a question arose: what would happen if these features were actually fed as an input of the original UBCF model?

This approach, called Feature Augmentation has been tested on a smaller set. In particular, the original dataset was first split in two (corresponding to 5740 users each), one for the ANN and another one for the UBCF. In each one of these sets a 90/10 train/test split was done. Thus, we have 5166 in each train set and 574 in tests.

Training the Denoising AE with this smaller set, didn't affect its accuracy, which had been constant over most epochs. We achieve a **test** accuracy of 83.99%. Additionally, when feeding the UBCF algorithm with these condensed vectors instead of the original sparse vectors, accuracy increases up to 90.91%.

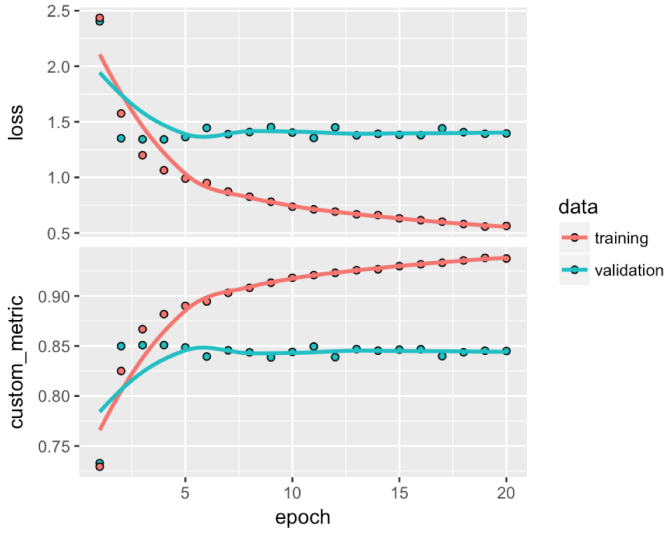


Figure 9.1: Training and Validation Accuracy per iteration on 1 layer Denoising AE on 5166 users

This unusual technique can be assimilated to a mathematical operation done in several engineering fields called **Linear Prediction**,

where previous samples are used to estimate discrete-time signals from them as a linear function. In order to validate this technique though, original and new correlations among user vectors need to be checked.

Correlations among all vectors on a sample of 20 of the users with the highest number of ratings (`orig_corr`) have been tested against their new correlation (`new_corr`). On 35.23% of the permutations, new correlation was in `orig_corr` ± 0.1. We consider this value to be low although we had a small sample. Thus, further comprobations need to be done.

Besides, the accuracy of the ANN itself could be improved with a temporal reference, building RNN which take into account common trends of the present period.

9.3.2 Improving the Chatbot Experience

The current version of Allyn works with a fully guided dialog. The chatbot experience can be improved through NLP enabling users to give free replies. This hasn't been implemented during the development of this Thesis because it requires to additionally train a full model to process text and identify intents which requires an extra dataset of common user messages classified to their meaning. The bot, although not processing these type of replies, is enabled to store them into a user-logs database. By gathering this information, we expect to be able to develop this model in the future.

We also intend to add some small new functionalities in the guided dialog itself such as "Go back" buttons, nonetheless, these need to be added from scratch, thus updating the `telegram.bot` package will be required. Moreover, speech-to-text algorithms could be potentially added a posteriori too. Using R, a Google API to do this is available, `googleLanguageR`.

9.3.3 Scaling the Solution

By using ANN as dense refeeding for traditional UBCF, we would be improving scalability, since this latter memory-based algorithm is able to adapt to newcoming items, although no rating reference exists while no user gives it an explicit evaluation.

Retraining frequence of the rating predictions using the ANN, which was more time expensive can be reduced, focusing on intermediate sucessive retraining of the UBCF replacing predictions with new actual ratings, which was faster.

Appendices

Appendix A

Auxiliary Data

A.1 ANOVA Test Values for Clusters

A.1.1 BookRating

```
Detailed Analysis for: BookRating
-----------------------------------------------------------

[1] "Summary by Class:"

    Cluster 1
    ---------
      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      1.00    7.00    8.00    7.63    9.00   10.00

    Cluster 2
    ---------
      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
     1.000   7.000   8.000   7.952  10.000  10.000

    Cluster 3
    ---------
      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
     1.000   7.000   8.000   7.529   9.000  10.000
```

```
[2] "p-value ANOVA: 1.46525531631466e-27"
[3] "p-value Kruskal-Wallis: 7.03901640049976e-36"
```

A.1.2 Age

```
Detailed Analysis for: Age
-----------------------------------------------------------

[1] "Summary by Class:"

    Cluster 1
    ---------
      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
     13.00   28.00   35.00   37.03   44.00   99.00    3129

    Cluster 2
    ---------
      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
     13.00   27.00   32.00   34.58   41.00   97.00    1707

    Cluster 3
    ---------
      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
     14.00   31.00   37.00   41.11   50.00   99.00    1415

[2] "p-value ANOVA: 8.54732967361984e-57"
[3] "p-value Kruskal-Wallis: 1.11708515218191e-51"
```

A.1.3 Country

```
Detailed Analysis for: Country
-----------------------------------------------------------

[1] "Relative Weight in Cluster:"
```

```
P      australia      austria      belgium      bermuda       canada
  1 0.0139161755 0.0016371971 0.0070399476 0.0006548788 0.0874263261
  2 0.0042529062 0.0011341083 0.0019846895 0.0005670542 0.0819393252
  3 0.0008726003 0.0004363002 0.0143979058 0.0000000000 0.0785340314

P   cayman islands        chile        china       cyprus czech republic
  1   0.0003274394 0.0016371971 0.0004911591 0.0001637197   0.0009823183
  2   0.0002835271 0.0000000000 0.0053870145 0.0000000000   0.0002835271
  3   0.0004363002 0.0000000000 0.0013089005 0.0026178010   0.0021815009

P          egypt      finland       france      germany       greece
  1 0.0014734774 0.0008185986 0.0026195154 0.0085134250 0.0004911591
  2 0.0005670542 0.0005670542 0.0019846895 0.0096399206 0.0000000000
  3 0.0004363002 0.0000000000 0.0000000000 0.0056719023 0.0000000000

P      hong kong      ireland        italy        japan       kuwait
  1 0.0003274394 0.0018009168 0.0011460380 0.0006548788 0.0004911591
  2 0.0002835271 0.0028352708 0.0045364332 0.0002835271 0.0000000000
  3 0.0000000000 0.0004363002 0.0004363002 0.0008726003 0.0000000000

P       malaysia        malta       mexico  netherlands  new zealand
  1 0.0019646365 0.0004911591 0.0004911591 0.0029469548 0.0153896529
  2 0.0014176354 0.0000000000 0.0002835271 0.0017011625 0.0053870145
  3 0.0030541012 0.0000000000 0.0008726003 0.0021815009 0.0026178010

P    philippines       poland     portugal      romania    singapore
  1 0.0075311067 0.0006548788 0.0096594630 0.0044204322 0.0018009168
  2 0.0045364332 0.0014176354 0.0014176354 0.0002835271 0.0014176354
  3 0.0008726003 0.0008726003 0.0017452007 0.0000000000 0.0008726003

P    south korea        spain  switzerland united kingdom          usa
  1 0.0001637197 0.0207924034 0.0013097577   0.0281597904 0.7691552063
  2 0.0002835271 0.0014176354 0.0031187978   0.0292032889 0.8296002268
  3 0.0004363002 0.0030541012 0.0008726003   0.0226876091 0.8486038394

[2] "Test Values:"

P       australia       austria       belgium       bermuda        canada
  1  6.518122e+00  3.580549e+00  8.347023e+00 -1.752654e+00  3.446962e+01
```

```
  2  2.301935e+00 -1.656513e+02 -9.482815e-02 -1.385665e+00  1.877697e+01
  3 -3.520774e+00  1.516862e-01  8.849428e+00 -1.394162e+00  1.502456e+01

P   cayman islands         chile         china        cyprus czech republic
  1   4.638679e-01  4.060218e+00  5.862482e-01 -1.047678e+01   2.119010e-01
  2  -6.033199e+00 -6.149878e-01  7.557036e+00 -2.915335e+01  -1.209336e+01
  3  -4.584537e+00 -3.703180e+00 -1.209746e+00 -6.044155e+00  -2.790913e-01

P            egypt       finland        france       germany        greece
  1  -1.260070e-01 -4.352490e+00 -1.513039e+01  9.878732e+00 -2.450325e+00
  2   7.042951e-01 -7.845357e+01  1.172755e+00 -5.807383e+01 -9.353994e+00
  3  -1.884748e+01 -1.764520e+03 -8.449184e+00  3.133025e+00 -1.971394e+00

P       hong kong       ireland         italy         japan        kuwait
  1 -4.788686e+00 -3.324232e-01  2.233431e+00  3.140403e-01  1.690791e+00
  2 -3.224648e+00 -9.013466e+00  4.250907e+00  9.349344e-02 -2.244959e+00
  3 -1.971394e+00  2.293685e-01 -1.873201e+00 -2.506450e+00 -7.885577e+00

P        malaysia         malta        mexico    netherlands   new zealand
  1 -1.165572e+00 -3.041913e+00 -5.079807e+00  -1.273407e+01  1.675930e+01
  2 -1.351718e+01 -2.544286e+01  1.009802e-01   6.372782e-01 -5.604238e+01
  3 -2.747950e+00 -7.885577e+00 -2.561125e+01  -5.167054e+00  1.223245e+00

P     philippines        poland      portugal       romania     singapore
  1  1.025025e+01 -1.294693e+00  1.148902e+01  9.638504e+00  3.499223e+00
  2  2.363875e+00  9.593649e-01 -6.469196e+00 -2.282633e+00  2.125807e+00
  3  2.083033e-01  5.013279e-01  1.056707e+00 -2.214747e+00 -1.563768e+00

P     south korea         spain   switzerland united kingdom           usa
  1 -6.195790e-01  1.914211e+01  1.034140e+00   1.826108e+01  2.155935e+02
  2 -9.796897e-01 -5.644070e+00 -4.359297e+00   1.254677e+01  1.425373e+02
  3 -6.420056e+00 -1.147401e+00 -1.874053e+00   3.599017e+00  6.193203e+00

[3] "P-values:"

P       australia       austria       belgium       bermuda        canada
  1  3.559658e-11  1.714364e-04  3.500241e-17  9.601693e-01 1.144602e-260
  2  1.066941e-02  1.000000e+00  5.377743e-01  9.170753e-01  5.827315e-79
  3  9.997849e-01  4.397172e-01  4.398432e-19  9.183656e-01  2.534981e-51
```

P	cayman islands	chile	china	cyprus	czech republic
1	3.213712e-01	2.451346e-05	2.788544e-01	1.000000e+00	4.160921e-01
2	1.000000e+00	7.307186e-01	2.061782e-14	1.000000e+00	1.000000e+00
3	9.999977e-01	9.998935e-01	8.868119e-01	1.000000e+00	6.099126e-01

P	egypt	finland	france	germany	greece
1	5.501368e-01	9.999933e-01	1.000000e+00	2.574011e-23	9.928636e-01
2	2.406245e-01	1.000000e+00	1.204471e-01	1.000000e+00	1.000000e+00
3	1.000000e+00	1.000000e+00	1.000000e+00	8.650732e-04	9.756606e-01

P	hong kong	ireland	italy	japan	kuwait
1	9.999992e-01	6.302151e-01	1.276028e-02	3.767452e-01	4.543837e-02
2	9.993694e-01	1.000000e+00	1.064533e-05	4.627558e-01	9.876146e-01
3	9.756606e-01	4.092913e-01	9.694797e-01	9.939025e-01	1.000000e+00

P	malaysia	malta	mexico	netherlands	new zealand
1	8.781063e-01	9.988246e-01	9.999998e-01	1.000000e+00	2.421556e-63
2	1.000000e+00	1.000000e+00	4.597831e-01	2.619718e-01	1.000000e+00
3	9.970015e-01	1.000000e+00	1.000000e+00	9.999999e-01	1.106185e-01

P	philippines	poland	portugal	romania	singapore
1	5.901810e-25	9.022869e-01	7.490021e-31	2.749183e-22	2.333078e-04
2	9.042451e-03	1.686875e-01	1.000000e+00	9.887740e-01	1.675968e-02
3	4.174961e-01	3.080702e-01	1.453227e-01	9.866113e-01	9.410640e-01

P	south korea	spain	switzerland	united kingdom	usa
1	7.322325e-01	5.631118e-82	1.505353e-01	8.445954e-75	0.000000e+00
2	8.363803e-01	1.000000e+00	9.999935e-01	2.070244e-36	0.000000e+00
3	1.000000e+00	8.743920e-01	9.695385e-01	1.597109e-04	2.947678e-10

A.1.4 Continent

```
Detailed Analysis for: Continent
--------------------------------------------------------
```

```
[1] "Relative Weight in Cluster:"

P         Africa     Americas         Asia       Europe      Oceania
  1 0.0014734774 0.8621480026 0.0135887361 0.0934839555 0.0293058284
  2 0.0005670542 0.9146583499 0.0136092997 0.0615253757 0.0096399206
  3 0.0004363002 0.9310645724 0.0100349040 0.0549738220 0.0034904014

[2] "Test Values:"

P          Africa      Americas          Asia        Europe       Oceania
  1     1.6494271   281.9377162  -869.5242931    31.5278499    13.6020797
  2 -1987.3957334   203.4261948     0.4371695    16.1000629   -35.1551120
  3   -20.9604880   158.9150269   -31.4220979  -167.3904016    -5.9401365

[3] "P-values:"

P          Africa      Americas          Asia        Europe       Oceania
  1  4.953009e-02  0.000000e+00  1.000000e+00 1.804127e-218  1.946032e-42
  2  1.000000e+00  0.000000e+00  3.309942e-01  1.274227e-58  1.000000e+00
  3  1.000000e+00  0.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00
```

A.1.5 Genre

```
Detailed Analysis for: Genre
------------------------------------------------------------

[1] "Relative Weight in Cluster:"

P   bio & memoirs   kids & teens    lit & fict  mist & thril
  1  0.0006499415   0.0037696607  0.9664630183  0.0127388535
  2  0.1013680197   0.2209015474  0.0044853106  0.0004485311
  3  0.0003297066   0.0000000000  0.0023079459  0.9937355753

P          other    reference      romance  sci-fi & fantasy
  1 0.0048095671 0.0012998830 0.0029897309      0.0072793449
```

```
  2 0.3449203857 0.1211033864 0.0753532182      0.1314196008
  3 0.0029673591 0.0003297066 0.0003297066      0.0000000000

[2] "Test Values:"

P   bio & memoirs   kids & teens    lit & fict  mist & thril
  1    -21.546222    -100.692742    235.477584    -16.712294
  2     16.069514      37.107095     -6.006420    -96.612867
  3   -176.763910      -8.948538     -3.429984    120.253033

P          other    reference      romance  sci-fi & fantasy
  1   -40.648851   -27.363512  -400.878431        -17.999956
  2    80.223869    38.639315   -67.744221         42.529628
  3    -4.177202   -21.875134   -42.077646        -12.934815


[3] "P-values:"

P   bio & memoirs    kids & teens    lit & fict   mist & thril
  1  1.000000e+00    1.000000e+00  0.000000e+00   1.000000e+00
  2  2.086721e-58   1.079376e-301  1.000000e+00   1.000000e+00
  3  1.000000e+00    1.000000e+00  9.996982e-01   0.000000e+00

P           other     reference      romance  sci-fi & fantasy
  1  1.000000e+00  1.000000e+00  1.000000e+00     1.000000e+00
  2  0.000000e+00  0.000000e+00  1.000000e+00     0.000000e+00
  3  9.999852e-01  1.000000e+00  1.000000e+00     1.000000e+00
```

A.2 ANOVA and Tukey's HSD Test Values for Models

A.2.1 Hyperparameter Tuning - Number of Hidden Layers and Learning Rate

```
Tuning Number of Hidden Layers and Learning Rate
-------------------------------------------------------
```

```
ANOVA
----------

              Df   Sum Sq   Mean Sq F value Pr(>F)
  model        7 0.003077 0.0004396   98.95 <2e-16 ***
  Residuals  392 0.001742 0.0000044

  Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1


Tuckey HSD
----------

Tukey multiple comparisons of means
95% family-wise confidence level

Fit: aov(formula = lm(accuracy ~ model, data = anova_table))

$model
               diff           lwr           upr     p adj
h2-h1  0.0070780640  5.793348e-03  0.0083627797 0.0000000
h3-h1  0.0014191798  1.344642e-04  0.0027038955 0.0188063
h4-h1  0.0012265140 -5.820164e-05  0.0025112297 0.0734641
h5-h1  0.0072250818  5.940366e-03  0.0085097975 0.0000000
h6-h1  0.0047711821  3.486466e-03  0.0060558977 0.0000000
h7-h1  0.0072664961  5.981780e-03  0.0085512118 0.0000000
h8-h1  0.0036676241  2.382908e-03  0.0049523397 0.0000000
h3-h2 -0.0056588842 -6.943600e-03 -0.0043741686 0.0000000
h4-h2 -0.0058515500 -7.136266e-03 -0.0045668344 0.0000000
h5-h2  0.0001470178 -1.137698e-03  0.0014317334 0.9999698
h6-h2 -0.0023068820 -3.591598e-03 -0.0010221663 0.0000022
h7-h2  0.0001884321 -1.096284e-03  0.0014731477 0.9998381
h8-h2 -0.0034104400 -4.695156e-03 -0.0021257243 0.0000000
h4-h3 -0.0001926658 -1.477381e-03  0.0010920498 0.9998121
h5-h3  0.0058059020  4.521186e-03  0.0070906177 0.0000000
h6-h3  0.0033520023  2.067287e-03  0.0046367179 0.0000000
h7-h3  0.0058473163  4.562601e-03  0.0071320320 0.0000000
h8-h3  0.0022484442  9.637286e-04  0.0035331599 0.0000045
```

```
h5-h4  0.0059985678  4.713852e-03  0.0072832835 0.0000000
h6-h4  0.0035446681  2.259952e-03  0.0048293837 0.0000000
h7-h4  0.0060399821  4.755266e-03  0.0073246978 0.0000000
h8-h4  0.0024411100  1.156394e-03  0.0037258257 0.0000004
h6-h5 -0.0024538998 -3.738615e-03 -0.0011691841 0.0000003
h7-h5  0.0000414143 -1.243301e-03  0.0013261300 1.0000000
h8-h5 -0.0035574578 -4.842173e-03 -0.0022727421 0.0000000
h7-h6  0.0024953141  1.210598e-03  0.0037800297 0.0000002
h8-h6 -0.0011035580 -2.388274e-03  0.0001811576 0.1526592
h8-h7 -0.0035988721 -4.883588e-03 -0.0023141564 0.0000000
```

A.2.2 Hyperparameter Tuning - Layer Sizes

```
Tuning Layer Sizes
----------------------------------------------------------

ANOVA
----------

               Df Sum Sq  Mean Sq F value Pr(>F)
  model         3   0.01 0.002986   0.185  0.907
  Residuals   4724  76.22 0.016136

  Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1


Tuckey HSD
----------

Tukey multiple comparisons of means
95% family-wise confidence level

$model
               diff          lwr        upr     p adj
m2-m1  0.0035866880 -0.009841661 0.01701504 0.9023134
m3-m1  0.0018590631 -0.011569286 0.01528741 0.9845675
m4-m1  0.0030498336 -0.010378515 0.01647818 0.9370274
m3-m2 -0.0017276249 -0.015155974 0.01170072 0.9875351
```

```
m4-m2 -0.0005368544 -0.013965203 0.01289149 0.9996119
m4-m3  0.0011907705 -0.012237578 0.01461912 0.9958302
```

A.2.3 Dropout on M2

```
Dropout on M2
------------------------------------------------------------

ANOVA
----------

               Df Sum Sq  Mean Sq F value Pr(>F)
  model         3   0.01 0.002289   0.143  0.934
  Residuals   4724  75.60 0.016003

  Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1


Tuckey HSD
----------

Tukey multiple comparisons of means
95% family-wise confidence level

$model
              diff         lwr         upr     p adj
m5-m2 0.0003840442 -0.01298899 0.01375708 0.9998559
m6-m2 0.0007898598 -0.01258318 0.01416290 0.9987545
m7-m2 0.0030989649 -0.01027407 0.01647200 0.9334518
m6-m5 0.0004058157 -0.01296722 0.01377885 0.9998300
m7-m5 0.0027149207 -0.01065812 0.01608796 0.9538836
m7-m6 0.0023091050 -0.01106393 0.01568214 0.9708289
```

A.2.4 Dropout on M4

```
Dropout on M4
------------------------------------------------------------
```

```
ANOVA
----------

               Df Sum Sq  Mean Sq F value Pr(>F)
  model         3   0.02 0.006425   0.406  0.749
  Residuals  4724  74.79 0.015833

  Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1


Tuckey HSD
----------

Tukey multiple comparisons of means
95% family-wise confidence level

$model
                diff         lwr         upr     p adj
m4-m10 -0.0037612162 -0.01706303 0.009540595 0.8864519
m8-m10 -0.0029290734 -0.01623088 0.010372738 0.9421783
m9-m10 -0.0055966300 -0.01889844 0.007705181 0.7010153
m8-m4   0.0008321428 -0.01246967 0.014133954 0.9985216
m9-m4  -0.0018354138 -0.01513723 0.011466398 0.9847163
m9-m8  -0.0026675566 -0.01596937 0.010634255 0.9554412
```

A.2.5 Denoising AE and Genre-Specific AE

```
Denoising AE and Genre-Specific AE
------------------------------------------------------------

ANOVA
----------

               Df Sum Sq  Mean Sq F value Pr(>F)
  model         2   0.01 0.004315   0.269  0.764
  Residuals  2908  46.70 0.016060
```

```
  Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1


Tuckey HSD
----------

Tukey multiple comparisons of means
95% family-wise confidence level

$model
                   diff         lwr         upr     p adj
m10-lit    0.0007823712 -0.01181954 0.013384283 0.9883840
mist-lit  -0.0035761237 -0.01821004 0.011057797 0.8345193
mist-m10  -0.0043584948 -0.01866792 0.009950934 0.7550893
```

A.3 ANN Training Figures

A.3.1 Hyperparameter Tuning - Number of Hidden Layers and Learning Rate

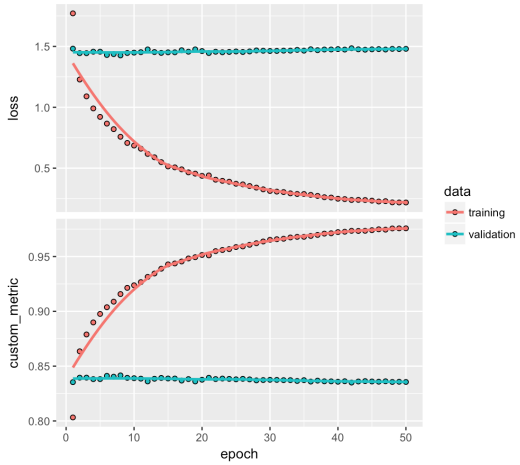


Figure A.1: H1 Training and Validation History

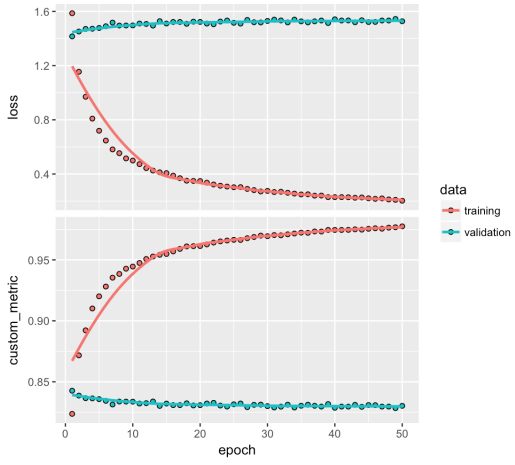


Figure A.2: H2 Training and Validation History

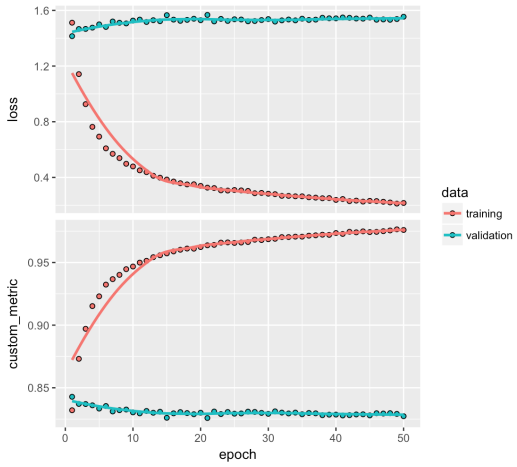


Figure A.3: H3 Training and Validation History

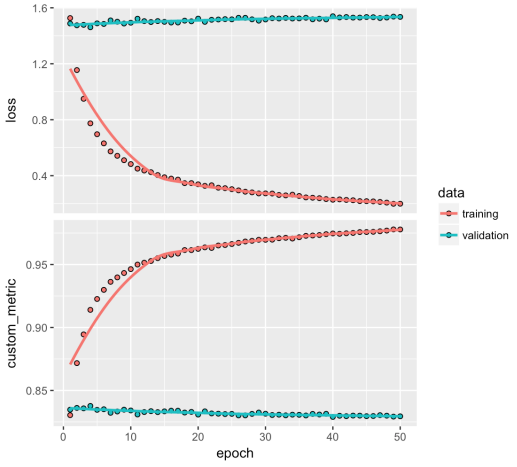


Figure A.6: H4 Training and Validation History

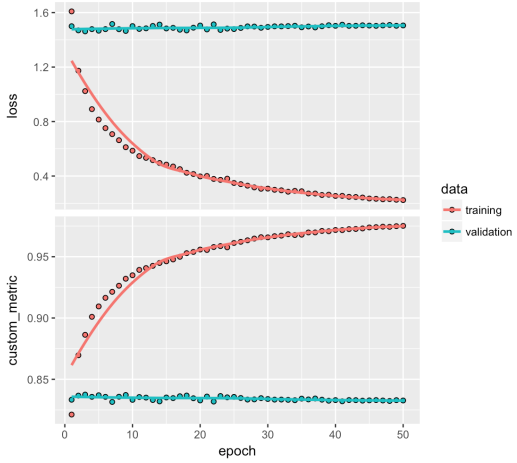


Figure A.4: H5 Training and Validation History

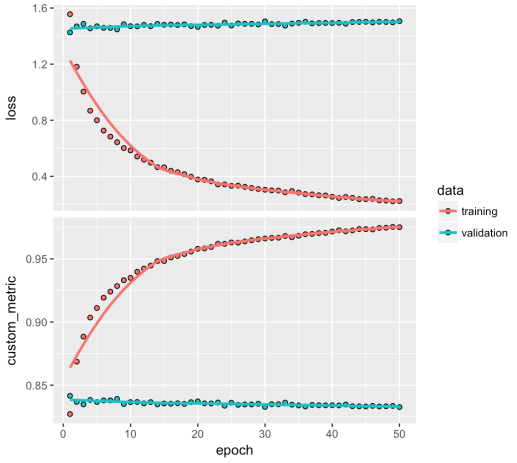


Figure A.7: H6 Training and Validation History

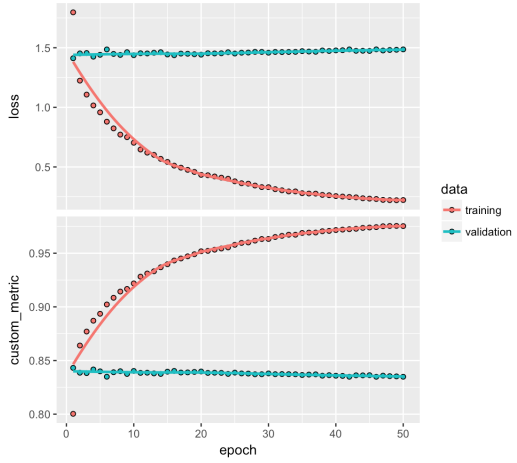


Figure A.5: H7 Training and Validation History

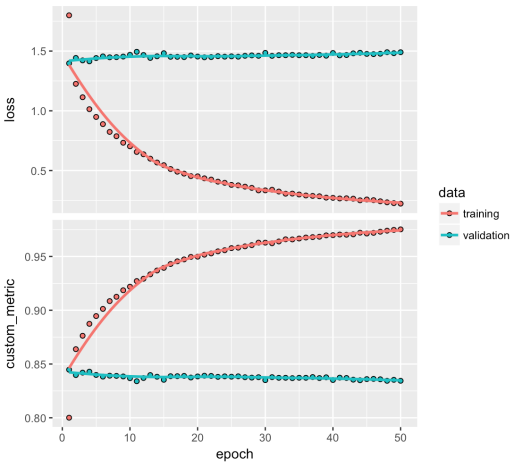


Figure A.8: H8 Training and Validation History

A.3.2 Hyperparameter Tuning - Layer Sizes

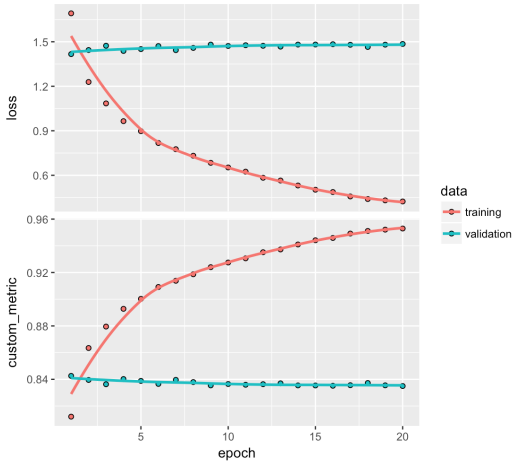


Figure A.9: M1 Training and Validation History

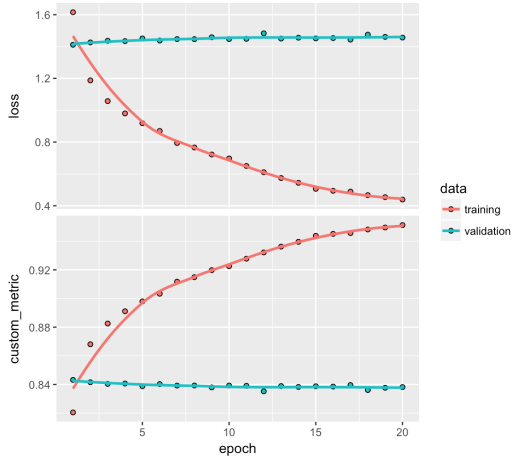


Figure A.10: M3 Training and Validation History

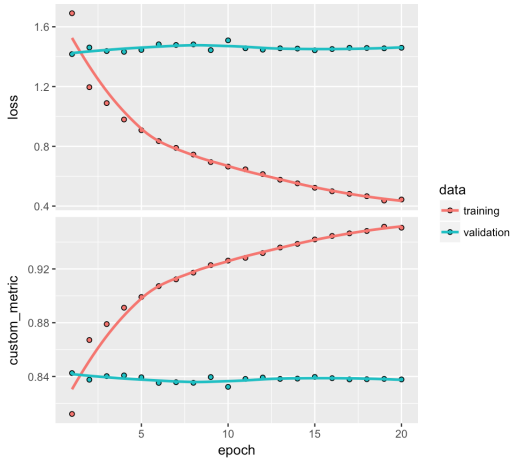


Figure A.11: M2 Training and Validation History

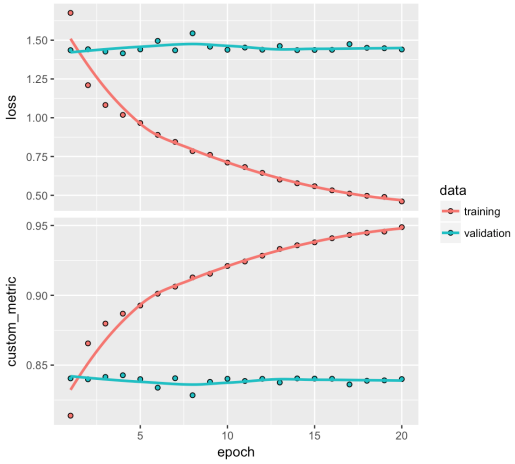


Figure A.12: M4 Training and Validation History

A.3.3 Dropout

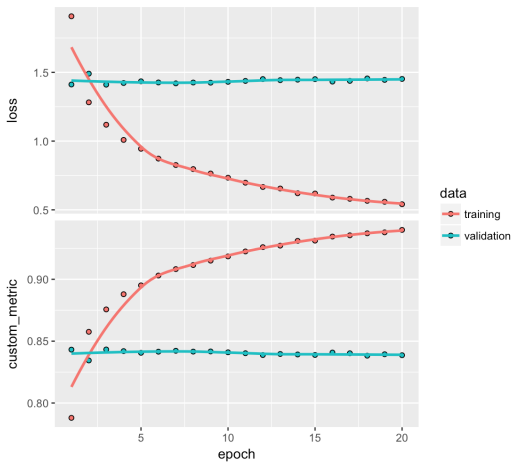


Figure A.13: M5 Training and Validation History

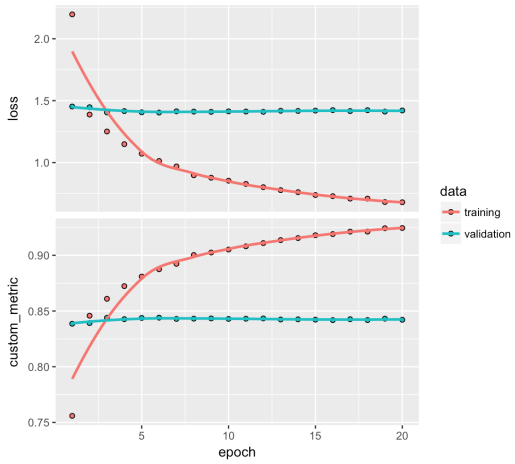


Figure A.14: M7 Training and Validation History

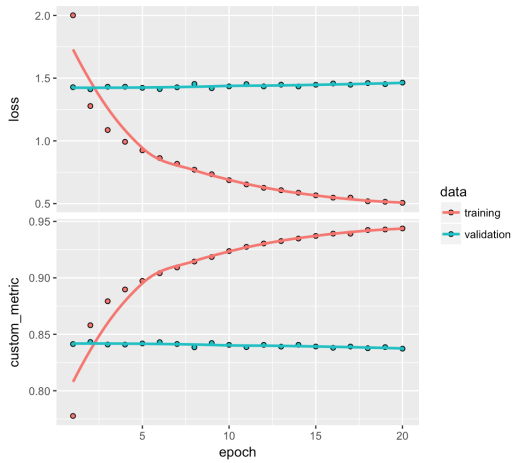


Figure A.15: M9 Training and Validation History

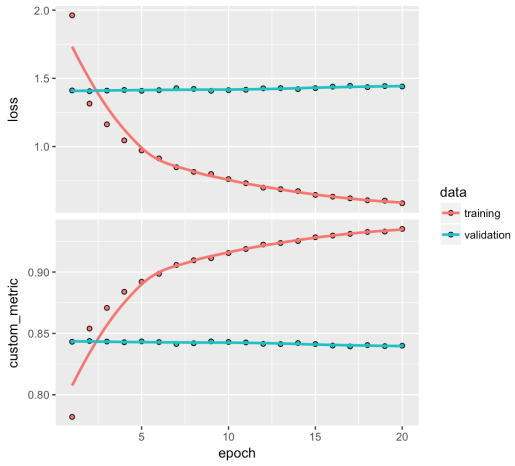


Figure A.16: M6 Training and Validation History

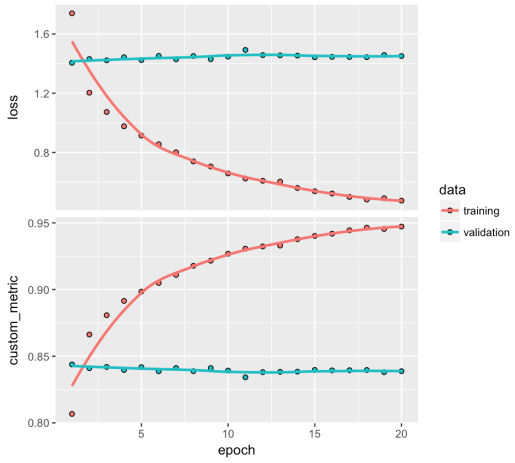


Figure A.17: M8 Training and Validation History

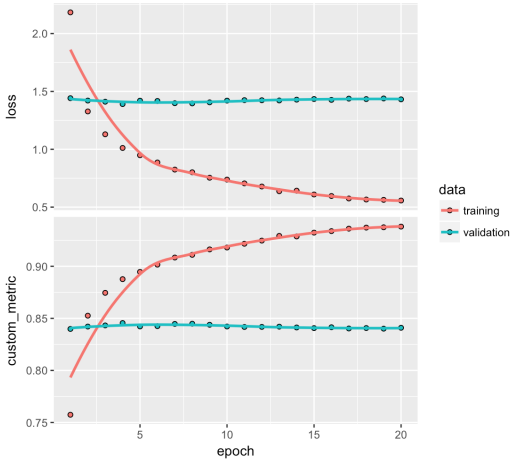


Figure A.18: M10 Training and Validation History

A.3.4 Model Variations

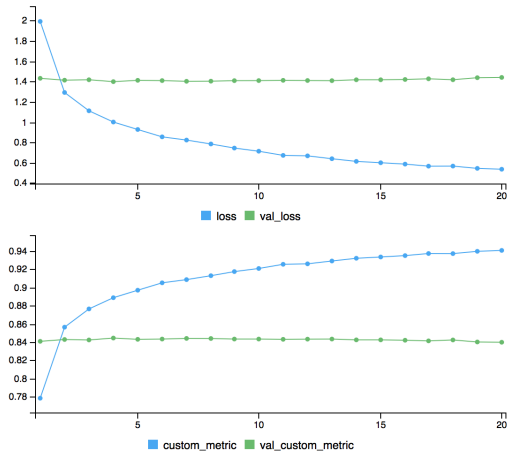


Figure A.19: Gaussian Noise Training and Validation History

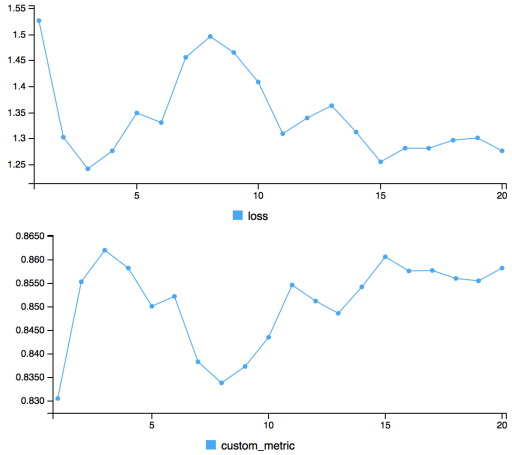


Figure A.21: External Variables Training History

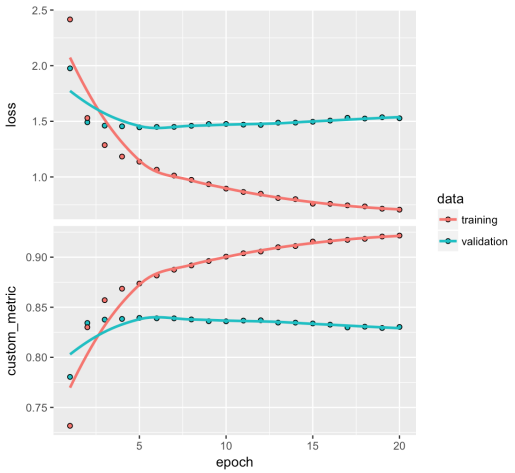


Figure A.20: Lit & Fict. Training and Validation History

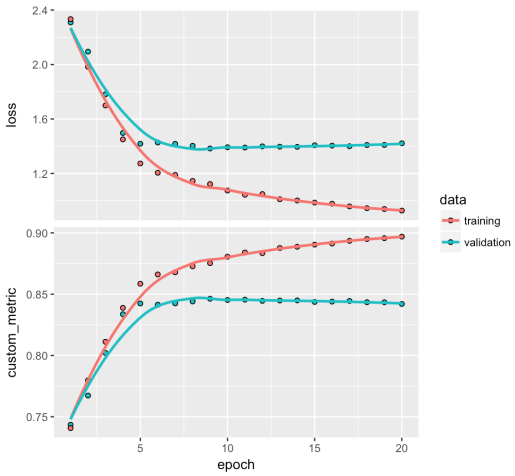


Figure A.22: Mist & Thrill. Training and Validation History

Appendix B

R Code

B.1 Preprocessing

```
#STEP 00: clean up environment and load packages ----------------------
rm(list=ls(all=TRUE))

source("r/misc_functions.R")
p_inst(c("data.table", "tidyr", "countycode"))

load("data/final_filtered_data.RData")


#STEP 01: identify repeated books -------------------------------------

table(table(books$BookTitle)>1)/length(unique(books$BookTitle))
rep_books <- names(table(books$BookTitle)[table(books$BookTitle)>1])


#STEP 02: set all character values tolower ----------------------------

## BOOK TITLES

#remove elements between brackets and parentheses
books$BookTitle <- gsub("\\[([^]]*)\\]","",books$BookTitle)
books$BookTitle <- gsub("\\(.*?\\)","",books$BookTitle)
#remove final spaces
```

```
books$BookTitle <- gsub("\\s+$", "",books$BookTitle)
#remove punctuation signs and lower case
books$BookTitle <- tolower(gsub("[^[:alnum:][:space:]]", "",
                                books$BookTitle))
#remove double spaces and turn into factors
books$BookTitle <- gsub("\\s\\s", " ", books$BookTitle)

## BOOK AUTHORS

books$BookAuthor <- gsub("not applicable na ", NA,
                         tolower(gsub("[[:punct:]]", "",
                                 gsub("\\s+$", "", books$BookAuthor))))

sort(table(books$BookAuthor), decreasing = T)
sort(table(books$Publisher), decreasing = T)[1:50]

## PUBLISHER

books$Publisher <- gsub("amp", "", tolower(gsub("[[:punct:]]", "",
                                                books$Publisher)))
length(unique(books$Publisher))


#STEP 03: split users location to city/county/country -----------------

commas <- lengths(gregexpr(",", gsub(", ", ",", users$Location,
                                     fixed = TRUE)))

users[, new_loc := strsplit(gsub(", ", ",", Location, fixed = TRUE), ",")]
users[, city := unlist(lapply(users$new_loc, function(x) x[[1]]))]
users[, country := unlist(lapply(users$new_loc, last))]

users[city == "n/a", city := NA]
users[country == "n/a", country := NA]

endcomma <- which(regexpr("\\,$", gsub(", ", ",", users$Location,
                                       fixed = TRUE))!=-1)
users[endcomma, country := NA]
```

```
rm(endcomma)

users <- users[, -c("Location", "new_loc"), with=F]


#STEP 04: add continents ------------------------------------------------

users[, continent := countrycode(country, "country.name", "continent")]

## manual correction of error values
users[country %in% c("phillipines"), `:=` (contintent = "Asia")]
users[country %in% c("united state", "america", "van wert"),
      `:=` (country = "usa", contintent = "Asia")]
users[country %in% c("antarctica"), `:=` (contintent = "Antarctica")]
users[country %in% c("catalonia", "euskal herria"),
      `:=` (country = "spain", contintent = "Europe")]
users[country %in% c("england"),
      `:=` (country = "great britain", contintent = "Europe")]
users[country %in% c("far away...", "n/a universe"),
      `:=` (city = NA, country = NA, contintent = NA)]


#STEP 05: remove probable Age typos -----------------------------------

users[[which(colnames(users)=="Age")]] <-
  as.numeric(users[[which(colnames(users)=="Age")]])

# the world's oldest person is 116 years old
table(users$Age)
# although improbable high ages are there, we'll cut at 110
users[Age > 110, Age := NA]

# manually checking books read by users under 5 years old,
# we observe that these must be all mistaken
ids <- users[Age < 5, UserID]
isbns <- reviews[UserID %in% ids, ISBN] %>% unique()
books[ISBN %in% isbns, .(BookTitle, BookAuthor)]

# but, 5yo (age at which most kids start learning to read)
```

```
# books are accurate
ids <- users[Age == 5, UserID]
isbns <- reviews[UserID %in% ids, ISBN] %>% unique()
books[ISBN %in% isbns, .(BookTitle, BookAuthor)]

# removing ages below 5
users[Age < 5, Age := NA]

rm(list=c("ids", "isbns"))


#STEP 06: create booklink variable ----------------------------------

## clean up necessary variables
books[, nchar_ISBN := nchar(ISBN), by="ISBN"]
books[, full_ISBN := paste0(paste(rep(0, 10-nchar_ISBN), collapse=""),
                            ISBN), by="ISBN"]
books[, clean_title := gsub("\\s", "-",
                            gsub("((?:\b| )?([.,:;!?]+)(?: |\b)?)",
                            "", BookTitle)), by="BookTitle"]

## merge booklink
books[, booklink := paste("https://www.amazon.com", clean_title, "dp",
                          paste0(full_ISBN, "/"), sep="/")]


save.image("data/post_preprocessing.RData")

#STEP 00: clean up environment and load packages -----------------------
rm(list=ls(all=TRUE))

source("r/misc_functions.R")
p_inst(c("data.table", "tidyr", "countrycode"))

load("data/clean_postwebscraping_datasets.RData")


#STEP 01: set all character values tolower ----------------------------
```

```
books[, genere := tolower(genere)]
books[, other := tolower(other)]
books[, pages := tolower(pages)]
books[, ages := tolower(ages)]


#STEP 02: merge redundant levels -------------------------------------

table(books[,genere])

books[grep("business", genere), genere := "business"]
books[grep("suspense", genere), genere := "mistery & thriller"]
books[grep("science fiction & fantasy", genere),
      genere := "sci-fi & fantasy"]
books[grep("computers", genere), genere := "reference"]
books[grep("engineering", genere), genere := "reference"]
books[grep("math", genere), genere := "reference"]
books[grep("medical", genere), genere := "reference"]
books[grep("law", genere), genere := "reference"]
books[grep("politics", genere), genere := "reference"]
books[grep("textbooks", genere), genere := "reference"]
books[grep("education", genere), genere := "reference"]
books[grep("boys", genere), genere := "kids & teens"]
books[grep("children", genere), genere := "kids & teens"]
books[grep("teen", genere), genere := "kids & teens"]
books[grep("gay", genere), genere := "romance"]
books[grep("religion", genere), genere := "religion"]
books[grep("bible", genere), genere := "religion"]
books[grep("entertainment", genere), genere := "entertainment"]
books[grep("comics", genere), genere := "entertainment"]

sort(table(books[,genere]), decreasing = T)


#STEP 03: input generes by knn? ---------------------------------------

save.image("data/post_second_preprocessing.RData")
```

B.2 Webscraping

```
#STEP 00: clean up environment and load packages ----------------------
rm(list=ls(all=TRUE))

source("r/misc_functions.R")
p_inst(c("data.table", "tidyr", "parallel", "doParallel", "foreach",
         "dplyr", "rvest", "rapportools"))

load("data/post_feature_engineering.RData")

#STEP 01: set up parameters -------------------------------------------

spec <- 3
cl <- makeCluster(spec)
registerDoParallel(cl)


#STEP 02: create placeholders  ----------------------------------------

downloads <- books[genere == "other",.(ISBN, booklink, genere)] %>%
  setDT()
downloads[, `:=` (genere=as.character(NA_character_),
                  pages=as.character(NA_character_), counter=0)]

downloads[[3]] <- as.character(downloads[[3]])
downloads[[4]] <- as.character(downloads[[4]])


#STEP 03: scrape variables in parallel  -------------------------------

iter <- which(is.na(downloads[,genere]))
n <- ceiling(length(iter)/1000)

foreach(i = iter, .packages = c('rvest', 'xml2', 'rapportools'),
        .verbose = T, .combine = rbind) %dopar% {

  a <- (j-1)*1000+1
```

```
b <- min(c(j*1000, length(iter)))

for(i in iter[b:a]){
  print(paste("###--- ", which(iter == i), "/", length(iter) ," ---###"))
  downloads[i,"counter"] <- downloads[i,"counter"] + 1

  booklink <- tryCatch(xml2::read_html(downloads[i, booklink]),
                       error=function(e){NA})

  # Get book genre from website
  webgenre <- NULL
  webgenre <- tryCatch(
    booklink %>%
      html_nodes('div#a-page') %>%
      html_nodes('div#dp') %>%
      html_nodes('div#wayfinding-breadcrumbs_container')%>%
      html_nodes('div#wayfinding-breadcrumbs_feature_div') %>%
      html_nodes('ul') %>%
      html_nodes('li') %>%
      .[3] %>%
      html_text(),
    error = function(e){}
  )

  if(!(rapportools::is.empty(webgenre))) {
    downloads[i, 3] <- gsub("[[:space:]]*$", "", gsub("^\\s+", "",
                            gsub("\n", "", webgenre)), perl=T)
  }

  downloads[i,,drop = F]

  # Get book genre from website
  npage_age <- NULL
  npage_age <- tryCatch(
    booklink %>%
      html_nodes('body') %>%
      html_nodes('div#a-page') %>%
      html_nodes('div#dp') %>%
      html_nodes('div#dp-container')%>%
```

```
        html_nodes('div#detail-bullets') %>%
        html_nodes('table') %>%
        .[1] %>%
        html_nodes('tr') %>%
        html_nodes('td') %>%
        html_nodes('div') %>%
        html_nodes('ul') %>%
        html_nodes('li') %>%
        .[1] %>%
        html_text(),
      error = function(e){}
    )

    if(!(rapportools::is.empty(npage_age))) {
      downloads[i,4] <- npage_age
    }

    downloads[i,,drop = F]
  }
}

downloads_clean <- downloads[-which(is.na(downloads[,3])),]
stopCluster(cl)

save.image("data/post_webscraping.RData")
```

B.3 Clustering and Profiling

```
#STEP 00: clean up environment and load packages ----------------------
rm(list=ls(all=TRUE))

source("r/misc_functions.R")
p_inst(c("data.table", "tidyr", "cluster"))

load("data/post_second_preprocessing.RData")

#STEP 01: subset only 4000 users to do the clustering  ----------------
```

```
## select indices
set.seed(1644)
ind <- sample(1:nrow(users), 1200)

## subset and merge datasets
subset <- users[ind,]
subset <- reviews[subset, nomatch=0,on="UserID"]
subset <- books[,.(ISBN, BookTitle, BookAuthor, genere)][subset,,
                                                         on="ISBN"]
subset[!(genere %in% c("literature & fiction", "mistery & thriller",
                       "kids & teens", "sci-fi & fantasy", "reference",
                       "biographies & memoirs", "romance")),
       genere := "other"]

for(i in 1:length(colnames(subset))){
  if(is.character(subset[[i]])){
    subset[[i]] <- as.factor(subset[[i]])
  }
}

subset[["UserID"]] <- as.factor(subset[["UserID"]])
subset <- subset[,.(BookTitle, BookAuthor, genere, BookRating, Age,
                    country, continent)]

rm(list=c("i", "ind"))

dissimMatrix <- daisy(subset, metric = "gower", stand=TRUE)

#Dendogram
par(mar=(c(0,4,3,4)),cex=0.75)
h <- hclust(dissimMatrix,method="ward.D")
plot(h, xaxt="n", xlab="", sub="", ylab="", labels=FALSE, hang=-50,
     main="")
abline(h = 300, col="#D95F02")


#clearly K=3
k <- 3
```

```
cut <- cutree(h, k[1])
cut <- as.factor(cut)

#Adding the variable to the subset:
subset[,URcluster := cut]


rm(list=setdiff(ls(), c("subset", "dissimMatrix", "h", "cut", "k")))
save.image("data/clustering.RData")


#STEP 00: clean up environment and load packages ----------------------
rm(list=ls(all=TRUE))

source("r/misc_functions.R")
p_inst(c("data.table", "tidyr", "cluster", "ggmosaic", "RColorBrewer",
         "lettercase"))

#load("data/post_webscraping.RData")
load("data/clustering.RData")


#STEP 01: select relevant variables -----------------------------------

r_var <- c("BookRating", "Age", "country", "continent", "genere",
           "URcluster")
data <- as.data.frame(subset[,(r_var), with=F])


#STEP 02: defining the functions --------------------------------------

#Compute test values for all the classes of P for a numerical variable
TestXnumVal <- function(Xnum,P){
  nk <- as.vector(table(P));
  n <- sum(nk);
  xk <- tapply(Xnum,P,mean);
  txk <- (xk-mean(Xnum))/(sd(Xnum)*sqrt((n-nk)/(n*nk)));
  pxk <- pt(txk,n-1,lower.tail=F);
  return (pxk)
```

```
}

#Compute test values for all the classes of P for a categorical variable
TestXqualiVal <- function(P,Xquali){
  taula <- table(P,Xquali);
  n <- sum(taula);
  pk <- apply(taula,1,sum)/n;
  pj <- apply(taula,2,sum)/n;
  pf <- taula/(n*pk);
  pjm <- matrix(data=pj,nrow=dim(pf)[1],ncol=dim(pf)[2]);
  dpf <- pf - pjm;
  dvt <- sqrt(((1-pk)/(n*pk))%*%t(pj*(1-pj)));
  zkj <- dpf/dvt;
  pzkj <- pnorm(zkj,lower.tail=F);
  return (list(rowpf=pf,vtest=zkj,pval=pzkj))
}


#STEP 03: evaluate ----------------------------------------------------

##CUT SELECTION

P<-data$URcluster
K<-dim(data)[2] - 1

nc<-length(levels(as.factor(P)))
pvalk <- matrix(data=0, nrow=nc,ncol=K, dimnames=list(levels(P),
                                                      names(data)[1:5]))
nameP<-"Class"
n<-dim(data)[1]

for(k in which(!names(data)%in%"URcluster")){
  if (is.numeric(data[,k])){
    #numeric
    print(paste("Detailed Analysis for:", names(data)[k]))

    boxplot(data[,k]~P, main=paste("Boxplot of", names(data)[k], "vs",
                                   nameP ), horizontal=TRUE)
```

```
    print("Summary by Class:")
    for(s in levels(as.factor(P))) {print(summary(data[P==s,k]))}
    o<-oneway.test(data[,k]~P)
    print(paste("p-value ANOVA:", o$p.value))
    kw<-kruskal.test(data[,k]~P)
    print(paste("p-value Kruskal-Wallis:", kw$p.value))
    pvalk[,k]<-TestXnumVal(data[,k], P)
  }else{
    #categoric
    print(paste("Detailed Analysis for:", names(data)[k]))
    xtab = as.data.frame(table(data[,k], data[,"URcluster"]))
    names(xtab)[2] = "Cluster"

    ggplot(data = xtab) +
      geom_mosaic(aes(weight = Freq, x = product(Var1), fill=Cluster)) +
      scale_fill_brewer(palette = "Dark2") +
      theme_light() +
      theme(legend.position="bottom",
            legend.background = element_rect(linetype="solid",
                                             colour ="gray4"),
            legend.title = element_text(face= "bold"),
            axis.text.x=element_text(angle = 45, hjust = 1)) +
      labs(title = paste0("Conditional Distribution of ",
                          str_title_case(names(data)[k]), " over Clusters"),
           x = names(data)[k], y = "Relative Frequence")
    print("Values:")
    print(TestXqualiVal(P,data[,k]))
  }
}
```

B.4 Artificial Neural Network

B.4.1 Model.R

```
#STEP 01: select users info to add ----------------------------------

if(!exists("train_output")){
```

```
  train_output <- train_input
  test_output <- test_input
}

n <- ncol(train_input)


#STEP 02: define custom functions ------------------------------------

K <- backend()

custom_activation <- function(x){

  x <- (x-K$min(x))/(K$max(x)-K$min(x))*9+1

}
attr(custom_activation, "py_function_name") <- "custom_activation"

custom_loss <- function(y_true, y_pred) {

  # convert tensors to R objects
  on <- K$greater_equal(y_true, K$constant(1))

  # calculate the loss
  loss <- loss_mean_absolute_error(tf$boolean_mask(y_true, on),
                                   tf$boolean_mask(y_pred, on))

  return(loss)
}

attr(custom_loss, "py_function_name") <- "custom_loss"

custom_metric <- function(y_true, y_pred) {

  # convert tensors to R objects
  on <- K$greater_equal(y_true, K$constant(1))

  # calculate the loss
  mae <- metric_mean_absolute_error(tf$boolean_mask(y_true, on),
```

```
                                    tf$boolean_mask(y_pred, on))

  return(1-mae/9)
}

attr(custom_metric, "py_function_name") <- "custom_metric"


#STEP 03: define model structure -------------------------------------

model <- keras_model_sequential()
model %>%
  layer_gaussian_noise(stddev = 0.0001, input_shape = c(n)) %>%
  layer_dense(units = 256, activation = 'tanh') %>%
  layer_dropout(rate=0.65) %>%
  layer_dense(units = n, activation = custom_activation)

summary(model)

optimizer_adam(lr = 0.001, beta_1 = 0.9, beta_2 = 0.999,
               epsilon = NULL, decay = 0)

model %>% compile(
  loss = custom_loss,
  optimizer = 'adam',
  metrics = custom_metric
)
```

B.4.2 Model_train.R

```
#STEP 00: clean up environment and load packages ---------------------
rm(list=ls(all=TRUE))

source("r/misc_functions.R")
p_inst(c("data.table", "tidyr", "keras", "kerasR", "tensorflow",
         "reticulate"))

choices = c("all", "lit", "mist")
```

```
gen = menu(choices = choices, title = "Select genere:")

gen = choices[gen]

load(paste0("data/", gen, "_selected_train_test.RData"))

mod="gaussian"

#STEP 01: load model ------------------------------------------------

source("r/model/model.R")


#STEP 02: train the model -------------------------------------------

time = Sys.time()
history <- model %>% fit(
  train_input, train_output,
  epochs = 20, batch_size = 32,
  validation_split = 0.2
)
round(Sys.time() - time,2)


plot(history)
keras_save_weights(model, path = paste0("model_weights/", mod, "_w.h5"))
```

B.4.3 Model1.R

```
#STEP 00: clean up environment and load packages ---------------------
rm(list=ls(all=TRUE))

source("r/misc_functions.R")
p_inst(c("data.table", "tidyr", "keras", "kerasR", "tensorflow",
         "reticulate"))

gen="all"
```

```
load(paste0("data/", gen, "_selected_train_test.RData"))


#STEP 01: select users info to add ----------------------------------

n <- ncol(train_input)
m <- nrow(train_input)
rm(list=setdiff(ls(), c("n", "m")))


#STEP 02: define custom functions -----------------------------------

K <- backend()

custom_activation <- function(x){

  x <- (x-K$min(x))/(K$max(x)-K$min(x))*9+1

}
attr(custom_activation, "py_function_name") <- "custom_activation"

custom_loss <- function(y_true, y_pred) {

  # convert tensors to R objects
  on <- K$greater_equal(y_true, K$constant(1))

  # calculate the loss
  loss <- loss_mean_absolute_error(tf$boolean_mask(y_true, on),
                                   tf$boolean_mask(y_pred, on))

  return(loss)
}

attr(custom_loss, "py_function_name") <- "custom_loss"

custom_metric <- function(y_true, y_pred) {

  # convert tensors to R objects
  on <- K$greater_equal(y_true, K$constant(1))
```

```
  # calculate the loss
  loss <- loss_mean_absolute_error(tf$boolean_mask(y_true, on),
                                   tf$boolean_mask(y_pred, on))

  return(1-loss/9)
}

attr(custom_metric, "py_function_name") <- "custom_metric"


#STEP 03: define model structure ---------------------------------------

input_t <- layer_input(shape = c(n), name = 'input')
aux_input_t <- layer_input(shape = c(3), name = 'aux_input')

output <- layer_concatenate(c(input_t, aux_input_t), axis=1) %>%
  layer_gaussian_noise(stddev = 0.0001) %>%
  layer_dense(units = 265, activation = 'tanh') %>%
  layer_dropout(rate = 0.65) %>%
  layer_dense(units = 9731, activation = custom_activation)

model <- keras_model(
  inputs = c(input_t, aux_input_t),
  outputs = output)

summary(model)

model %>% compile(
  loss = custom_loss,
  optimizer = 'adam',
  metrics = custom_metric
)


#STEP 04: define generator ---------------------------------------------

sampling_generator <- function(sample_name, batch_size) {
```

```
  input_filename = paste0("data/fit_", sample_name, "_input.csv")
  output_filename = paste0("data/fit_", sample_name, "_output.csv")
  aux_file = paste0("data/fit_aux_", sample_name, ".csv")

  in_data = fread(input_filename)
  out_data = fread(output_filename)
  aux_data = fread(aux_file)

  function() {
    rows <- sample(1:nrow(in_data), batch_size, replace = TRUE)
    list(list(as.matrix(in_data[rows,]), as.matrix(aux_data[rows])),
         as.matrix(out_data[rows,]))
  }
}


#STEP 05: train the model ------------------------------------------

history <- model %>% fit_generator(
  sampling_generator("train", batch_size = 64),
  steps_per_epoch = m/64, epochs = 20)

plot(history)
keras_save_weights(model, path = paste0("model_weights/external_w.h5"))


#STEP 06: evaluate model -------------------------------------------

err <- model %>% evaluate_generator(
  sampling_generator("test", batch_size = 32),
  steps = m/32
)
err

pred_generator <- function(sample_name, batch_size) {

  filename = paste0("data/fit_", sample_name, "_input.csv")
  aux_file = paste0("data/fit_aux_", sample_name, ".csv")
```

```
  data = fread(filename)
  aux_data = fread(aux_file)

  function() {
    rows <- sample(1:nrow(data), batch_size, replace = TRUE)
    list(as.matrix(data[rows,]), as.matrix(aux_data[rows]))
  }
}

pred <- model %>% predict_generator(
  pred_generator("test", batch_size = 64),
  steps = m/64
)
```

B.5 Chatbot

```
start <- function(bot, update){
  bot$sendMessage(
    chat_id = update$message$chat_id,
    text = sprintf(response_key[["Start"]],
                   update$message$from$first_name),
    reply_markup = InlineKeyboardMarkup(
      list(list(InlineKeyboardButton("Find a gift")),
           list(InlineKeyboardButton("Book for me")))
    )
  )
}

genre <- function(bot, update){
  bot$sendMessage(
    chat_id = update$callback_query$message$chat$id,
    text = response_key[["Genre"]],
    reply_markup = InlineKeyboardMarkup(
      list(list(InlineKeyboardButton("Yes!")),
           list(InlineKeyboardButton("Nope, let's explore what you've got!")))
    )
  )
```

```
}

genre_selection <- function(bot, update){
  bot$sendMessage(
    chat_id = update$callback_query$message$chat$id,
    text = response_key[["SpecifyGenre"]],
    reply_markup = InlineKeyboardMarkup(
      list(list(InlineKeyboardButton("Literature & Fiction")),
           list(InlineKeyboardButton("Mistery"),
                InlineKeyboardButton("Romance")),
           list(InlineKeyboardButton("Sci-Fi & Fantasy")),
           list(InlineKeyboardButton("Biographies"),
                InlineKeyboardButton("Reference")),
           list(InlineKeyboardButton("All generes"))
      )
    )
  )
}

pers_rec <- function(bot, update){
  bot$sendMessage(
    chat_id = update$callback_query$message$chat$id,
    text = response_key[["Pers"]],
    reply_markup = InlineKeyboardMarkup(
      list(list(InlineKeyboardButton("Yes, please!")),
           list(InlineKeyboardButton("Let's see the top picks"))
      )
    )
  )
}

rat_grid <- function(bot, update){
  bot$sendMessage(
    chat_id = update$callback_query$message$chat$id,
    text = sprintf(response_key[["Rat"]],
                   update$callback_query$message$chat$first_name),
    reply_markup = InlineKeyboardMarkup(
      list(list(InlineKeyboardButton(simple_cap(bbdd$BookTitle[1])),
                InlineKeyboardButton(simple_cap(bbdd$BookTitle[2]))),
```

```
          list(InlineKeyboardButton(simple_cap(bbdd$BookTitle[3])),
               InlineKeyboardButton(simple_cap(bbdd$BookTitle[4]))),
          list(InlineKeyboardButton(simple_cap(bbdd$BookTitle[5])),
               InlineKeyboardButton(simple_cap(bbdd$BookTitle[6]))),
          list(InlineKeyboardButton(simple_cap(bbdd$BookTitle[7])),
               InlineKeyboardButton(simple_cap(bbdd$BookTitle[8]))),
          list(InlineKeyboardButton("Done"))
      )
    )
  )
}

rate <- function(bot, update){
  bot$sendMessage(
    chat_id = update$message$chat_id,
    parse_mode = 'Markdown',
    text = paste0('[', simple_cap(title), '](', link, ') by \n_',
                  simple_cap(auth), '_'),
    reply_markup = InlineKeyboardMarkup(
      list(list(InlineKeyboardButton(1),
                InlineKeyboardButton(2),
                InlineKeyboardButton(3),
                InlineKeyboardButton(4),
                InlineKeyboardButton(5)))
    )
  )
}

send_top <- function(bot, update){
  bot$sendMessage(chat_id = update$callback_query$message$chat$id,
                  parse_mode = 'Markdown',
                  disable_web_page_preview = T,
                  text = llista)
}

send_rec <- function(bot, update){
  bot$sendMessage(
    chat_id = update$callback_query$message$chat$id,
    parse_mode = 'Markdown',
```

```
    text = paste0('[', simple_cap(pred_top_picks$BookTitle[1]),
                  '](', pred_top_picks$booklink[1], ') by \n_',
                  simple_cap(pred_top_picks$BookAuthor[1]), '_')
  )
}
```

References

Adomavicius, Gediminas, and Alexander Tuzhilin. 2005. "Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions." *IEEE Transactions on Knowledge and Data Engineering* 17 (6). IEEE: 734–49.

Allaire, JJ, François Chollet, RStudio, and Google. 2018. "R interface to Keras." https://keras.rstudio.com/index.html.

Alpaydın, Ethem. 2010. *Introduction to Machine Learning Second Edition*. Vol. 56.

Ba, S, and P Pavlou. 2002. "The Demand Effects of Product Recommendation Networks: An Empirical Analysis of Network Diversity and Stability." *Mis Quarterly* 26 (3): 243–68.

Balabanovic, Marko; and Yoav Shoham. 1997. "Content-Based, Collaborative Recommendation." *Communications of the ACM* 40 (3).

Benedito, Ernest. 2018. "Package 'Telegram.bot'." *CRAN-R*, 17. https://cran.r-project.org/web/packages/telegram.bot/telegram.bot.pdf.

Bobadilla, J., F. Ortega, A. Hernando, and A. Gutiérrez. 2013. "Recommender systems survey." *Knowledge-Based Systems* 46. Elsevier B.V.: 109–32.

Breese, John S., David Heckerman, and Carl Kadie. 1998. "Empirical analysis of predictive algorithms for collaborative filtering." *Proceedings of the 14th Annual Conference on Uncertainty in Artificial Intelligence*, 43——52.

Brisson, Keith. 2016. "11 best messaging platforms for your chatbot or conversational app." https://blog.init.ai/pick-your-platform-wisely-c5ab5bc7555d.

Burke, Robin. 2002. "Hybrid Recommender Systems: Survey and Experiments." *User Modeling and User-Adapted Interaction* 12 (4): 331–70.

Capan, F. 2018. "The AI Revolution is Underway." https://www.pm360online.com/

the-ai-revolution-is-underway/.

Chandak, Manisha, Sheetal Girase, and Debajyoti Mukhopadhyay. 2015. "Introducing hybrid technique for optimization of book recommender system." *Procedia Computer Science* 45 (C). Elsevier Masson SAS: 23–31.

Cheng, B, and Dm Titterington. 1994. "Neural networks: A review from a statistical perspective" 9 (1): 2–54.

Christensson, P. 2013. "Framework Definition." https://techterms.com/definition/framework.

Covington, Paul, Jay Adams, and Emre Sargin. 2016. "Deep Neural Networks for YouTube Recommendations." *Proceedings of the 10th ACM Conference on Recommender Systems - RecSys '16*, 191–98.

Dias, M. Benjamin, Dominique Locher, Ming Li, Wael El-Deredy, and Paulo J.G. Lisboa. 2008. "The value of personalised recommender systems to e-business." *Proceedings of the 2008 ACM Conference on Recommender Systems - RecSys '08*, no. January: 291.

Dogtiev, Artyom. 2017. "Telegram Revenue and Usage Statistics (2017)." http://www.businessofapps.com/data/telegram-statistics/.

Eurostat. 2018. "E-commerce statistics," no. December 2017: 1–10. http://ec.europa.eu/eurostat/statistics-explained/pdfscache/14386.pdf.

Felfernig, Alexander, and Robin Burke. 2008. "Constraint-Based Recommender Systems: Technologies and Research Issues." *ICEC '08 Proceedings of the 10th International Conference on Electronic Commerce*, 10.

Ge, Xinyang, Jia Liu, Qi Qi, and Zhenyu Chen. 2011. "A New Prediction Approach Based on Linear Regression for Collaborative Filtering" 4 (August): 2586–90.

Gomez-Uribe, Carlos A., and Neil Hunt. 2015. "The Netflix Recommender System." *ACM Transactions on Management Information Systems* 6 (4): 1–19.

Graff, Ryan (Northwestern University). 2015. "How the Washington Post used data and natural language processing to get people to read more news." https://knightlab.northwestern.edu/2015/06/03/how-the-washington-posts-clavis-tool-helps-to-make-news/-personal/.

Gunawardana, Asela, and Christopher Meek. 2009. "A Unified Approach to Building

Hybrid Recommender Systems." *RecSys*, 117–24.

Harper, F. Maxwell, and Joseph A. Konstan. 2015. "The MovieLens Datasets." *ACM Transactions on Interactive Intelligent Systems* 5 (4): 1–19.

Huang, Wenyi, Zhaohui Wu, Chen Liang, Prasenjit Mitra, and C Lee Giles. 2015. "A Neural Probabilistic Model for Context Based Citation Recommendation." *AAAI 2015: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2404–10.

Inc., SAS Institute. 2018. "Natural Language Processing." Accessed July 4. https://www.sas.com/en_us/insights/analytics/what-is-natural-language-processing-nlp.html.

Internation Telecomunications Union, ITU. 2005. "ICT facts and figures 2005." *Itu*.

Kaushik, Saurav. 2017. "Beginner's Guide on Web Scraping in R (using rvest) with hands-on example." https://www.analyticsvidhya.com/blog/2017/03/beginners-guide-on-web-scraping-in-r-using-rvest-with-hands-on-knowledge/.

Li, Qibing, Xiaolin Zheng, and Xinyue Wu. 2017. "Collaborative Autoencoder for Recommender Systems."

Linden, Greg, and Brent Smith. 2017. "Two Decades of Recommender Systems at Amazon.com." *IEEE Internet Computing* 7 (1): 12–18.

Linden, Greg, Brent Smith, and Jeremy York. 2003. "Amazon.com recommendations: Item-to-item collaborative filtering." *IEEE Internet Computing* 7 (1): 76–80.

Liu, Jiahui, Peter Dolan, and Elin Rønby Pedersen. 2010. "Personalized news recommendation based on click behavior." *Proceedings of the 15th International Conference on Intelligent User Interfaces - IUI '10*, 31.

Mackenzie, Ian, Chris Meyer, and Steve Noble. 2013. "How retailers can keep up with consumers." https://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers.

McEleny, Charlotte. 2016. "What CNN has learnt after six months of chatbot experimentation." http://www.thedrum.com/news/2016/11/16/what-cnn-has-learnt-after-six/-months-chatbot-experimentation.

Melville, Prem, Raymond J Mooney, and Ramadass Nagarajan. 2002. "Content-boosted collaborative filtering for improved recommendations." *"Proceedings of the 18th*

National Conference on Artificial Intelligence (AAAI)", no. July: 187–92.

Miller, Robert B. 1968. "Response time in man-computer conversational transactions." *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I on - AFIPS '68 (Fall, Part I)*, 267.

Misoffe, Augustin. 2017. "[ChatBot] Who wins? Guided dialogue and Natural language dialogue." https://tutorials.botsfloor.com/chatbot-who-wins-guided-dialogue-and/-natural-language-dialogue-781ecb1d96af.

Mitchell, Tom M. 2006. "The Discipline of Machine Learning." *Machine Learning* 17 (July): 1–7.

Miteva, Sara. 2017. "5 Programming Languages You Can Use to Create Chatbots." http://valosohub.com/blog/2017/08/21/programming-languages-chatbots/.

Mueller, Kirill, Hadley Wickham, David A James, and Seth Falcon. 2018. "Package 'RSQLite'." *CRAN-R*, 12.

Oestreicher-Singer, Gal and Sundararajan, Arun. 2011. "The Visible Hand? Demand Effects of Recommendation Networks in Electronic Markets." *Management Science*, 1–42.

Pazzani, Michael J. 1999. "A framework for collaborative, content-based and demographic filtering." *Artificial Intelligence Review* 13 (5): 393–408.

Quoc, Michael. 2017. "10 Ecommerce Brands Succeeding with Chatbots." https://www.abetterlemonadestand.com/ecommerce-chatbots/.

Resnick, Paul, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. 1994. "GroupLens : An Open Architecture for Collaborative Filtering of Netnews." *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, 175–86.

Ricci, Francesco, Lior Rokach, and Bracha Shapira. 2011. *Recommender Systems Handbook*.

Schein, Andrew I., Alexandrin Popescul, Lyle H. Ungar, and David M. Pennock. 2002. "Methods and metrics for cold-start recommendations." *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval - SIGIR '02*, no. August: 253.

Sharma, Amit, Jake M. Hofman, and Duncan J. Watts. 2015. "Estimating the Causal

Impact of Recommendation Systems from Observational Data."

Strub, Florian, Jeremie Mary, Romaric Gaudel, Florian Strub, Jeremie Mary, Romaric Gaudel, Hybrid Collaborative, Florian Strub, Florian Strub, and Inria Fr. 2016. "Hybrid Collaborative Filtering with Neural Networks To cite this version : Hybrid Collaborative Filtering with Neural Networks."

Sumbaly, Roshan, Jay Kreps, Lei Gao, Alex Feinberg, Chinmay Soman, and Sam Shah. 2012. "Serving Large-scale Batch Computed Data with Project Voldemort." *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST '12)*, 223–36.

Tang, Duyu, Bing Qin, Ting Liu, and Yuekui Yang. 2015. "User modeling with neural network for review rating prediction." *IJCAI International Joint Conference on Artificial Intelligence* 2015-January (Ijcai): 1340–6.

Telegram, LLC. 2013. "Telegram a new era of messaging." https://telegram.org.

———. 2015. "Bots: An introduction for developers." https://core.telegram.org/bots.

The TensorFlow Authors and RStudio. 2018. "R Interface to TensorFlow." https://tensorflow.rstudio.com.

Vairavasundaram, Subramaniyaswamy, Vijayakumar Varadharajan, Indragandhi Vairavasundaram, and Logesh Ravi. 2015. "Data mining-based tag recommendation system: An overview." *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 5 (3): 87–112.

Vargiu, Eloisa, and Mirko Urru. 2012. "Exploiting web scraping in a collaborative filtering- based approach to web advertising." *Artificial Intelligence Research* 2 (1): 44–54.

Vincent, Pascal, and Hugo Larochelle. 2010. "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion Pierre-Antoine Manzagol." *Journal of Machine Learning Research* 11: 3371–3408.

Wang, Hao, Naiyan Wang, and Dit-Yan Yeung. 2014. "Collaborative Deep Learning for Recommender Systems."

Wickham, Hadley (RStudio). 2016. "Package ' Rvest'."

Wu, Lili, Sam Shah, Sean Choi, Mitul Tiwari, and Christian Posse. 2014. "The browsemaps: Collaborative filtering at LinkedIn." *CEUR Workshop Proceedings* 1271

(RSWeb).

Xiao, Bo, and Izak Benbasat. 2007. "E-Commerce Product Recommendation agents: Use, Characteristics, and Impact." *MIS Quarterly* 31 (1): 137–209.

Zhang, Xingxing, and Mirella Lapata. 2014. "Chinese Poetry Generation with Recurrent Neural Networks." *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP'14)*, 670–80.

Zhou, Renjie, Samamon Khemmarat, and Lixin Gao. 2010. "The Impact of YouTube Recommendation System on Video Views." *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, 404–10.

Zhuang, Zhongfang. 2013. "The 'Big Data' Ecosystem at LinkedIn." *SIGMOD*, 1125–34.

Ziegler, Cai-Nicolas C.N., Sean M. S.M. McNee, Joseph a. J.a. Konstan, and Georg Lausen. 2005. "Improving recommendation lists through topic diversification." *Proceedings of the 14th International Conference on World Wide Web WWW 05*, no. January: 22.