Exploiting business knowledge in a recommender system for handset devices

André Silva

Master's thesis presented to the Faculty of Sciences of the University of Porto in Computer Science

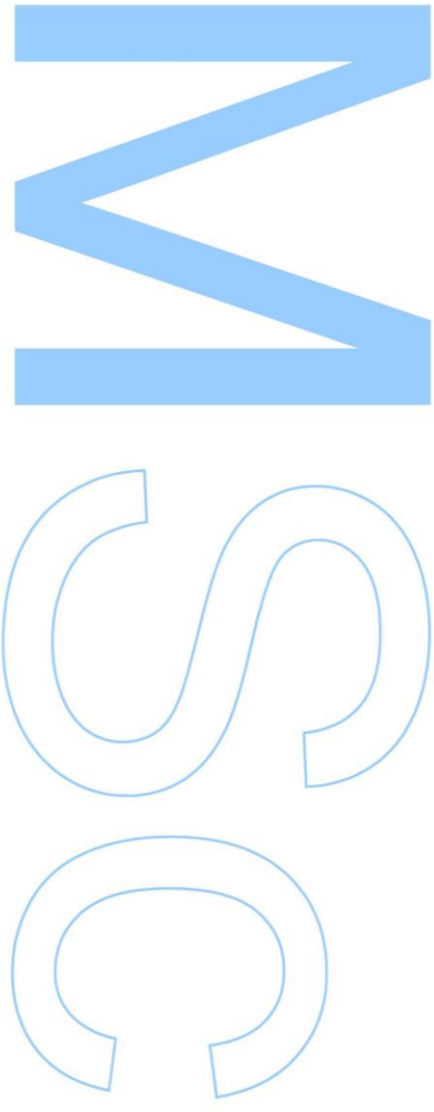
2019

Exploiting business knowledge in a recommender system for handset devices

André Silva
Computer Science: Data Mining and Data Processing
Department of Computer Science
2019

Supervisor
João Vinagre, PhD, FCUP

Co-Supervisors
Alípio Jorge, PhD, FCUP
Kelwin Fernandes, PhD



Acknowledgements

First things first, I would like to thank my supervisor João Vinagre, my mentor at the company, Kelwin Fernandes and my co-supervisor Alípio Jorge for their immense support shown during the development of this dissertation. I've learned a lot from all of you and would not have finished this dissertation without all your help!

I would also like to thank Nuno Paiva for the opportunity given and all my colleagues at the Telco with whom i had the pleasure of sharing a great work environment and had the opportunity to learn a lot more about recommender systems and other data sciences fields.

Cheers to all my peers also writing their dissertation at the Telco with whom any discussion always turned out insightful and entertaining and for their patience to hear out my rambles.

Lastly, but not least, my family for always being there for me when I needed them. Without all of you I would not be here and this dissertation would not exist.

Abstract

Following their inception, handset devices were used only as communication tools without much more utility. Customers decided their purchases based on very simple characteristics such as price, color, brand, shape and whether it had some smaller extras like a lantern and simple games. Choices were limited which made the purchasing decision very simple, akin to buying a nice t-shirt, rather than a complex one that would have a strong impact on the buyer's life. Nowadays, the picture has changed significantly and an enormous diversity of phones is available with devices advertising very technical specification such as RAM, CPU frequency, operating system version, triple cameras, notches and many others. Picking the correct device to buy is nowadays more similar to choosing the right laptop, making this a difficult decision for customers.

Recommender systems ease the users' burden on this decision by offering suggestions that he would most likely be interested in. Collaborative filtering is a recommendation technique, where recommendations are made by looking at the history of items the user has liked in the past and choices made by users with similar tastes whereas with content-based techniques, suggestions are generated taking only into account the user and the item characteristics. Therefore, in an attempt to match people with the right handset that would best satisfy their needs, we propose the development of a recommender system capable of suggesting handsets to customers of a Telco, using the aforementioned recommendation techniques.

In this dissertation we develop a solution for the problem of handset recommendation that takes into account the user's current handset. For this solution, a recommender systems approach, using collaborative filtering, content-based and hybrid techniques was developed and applied to real data provided by one of the top three players in telecom business in Portugal. We compare these approaches to existing baselines used at the Telco and show that our method outperforms the existing solutions by a large margin.

Keywords: Recommender systems, Content-based, Data science, Handset recommendation

Resumo

Nos seus primórdios, os telemóveis eram usados apenas como ferramentas de comunicação sem muita utilidade adicional. Os clientes decidiam o que comprar tendo como base características muito simples, como o preço, cor, marca, forma e se tinha alguns extras menores, como lanterna e jogos simples. As escolhas eram limitadas, o que tornava a decisão de compra muito simples, semelhante à compra de uma t-shirt bonita, em vez de uma decisão complexa que teria um forte impacto na vida dos seus utilizadores. Hoje em dia, a imagem mudou significativamente e uma enorme diversidade de telemóveis está disponível com fabricantes a anunciar especificações muito técnicas, como RAM, frequência de CPU, versão do sistema operacional, câmaras triplas, entalhes e muitos outros. A escolha sobre o telemóvel a comprar é, hoje em dia, mais parecido com a escolha de um computador portátil, tornando esta uma decisão difícil para os clientes.

Os sistemas de recomendação facilitam a vida dos utilizadores por oferecerem sugestões em que ele provavelmente esteja intessado. Collaborative filtering é uma técnica de recomendação, onde os itens são recomendados por base o histórico de itens que o usuário gostou ou escolhas feitas por usuários com gostos semelhantes. No caso de content-based, outra técnica de recomendação, as sugestões são geradas tendo em conta apenas as características do utilizador e do item. Assim, numa tentativa de combinar pessoas com o telemóvel certo que melhor atenda às suas necessidades, propomos o desenvolvimento de um sistema de recomendação capaz de sugerir aparelhos a clientes de uma empresa de telecomunicações, baseado nas técnicas mencionadas anteriormente.

Nesta dissertação foi desenvolvida uma solução para o problema de recomendação de telemóveis tendo em conta o telemóvel atual do cliente. Para a solução, um sistema de recomendação baseado em técnicas de collaborative filtering, técnicas baseadas em conteúdo e técnicas híbridas que foram então aplicadas a dados reais fornecidos por uma das três principais empresas de telecomunicações em Portugal. Comparamos essas abordagens às abordagens já existentes usadas na empresa de telecomunicações e mostramos que o nosso método supera essas mesmas soluções.

Palavras-chave: Sistemas recomendação, Content-based, Data science, Recomendação telemóveis

Contents

List of Tables

List of Figures

Acronyms

ARHR Average Reciprocal Hit-Rank

AUC Area Under ROC Curve

Bagging Bootstrap Aggregating

BPR Bayesian personalized ranking

CPU Central Processing Unit

DCC Departamento de Ciência de Computadores

FCUP Faculdade de Ciências da Universidade do Porto

GB Gigabytes

GBT Gradient Boosting Trees

GPRS General Packet Radio Services

HR Hit Rate

IMEI International Mobile Equipment Identity

LTE Long-Term Evolution

mAh Milliampere hour

MAP Mean Average Precision

MB Megabytes

MRR Mean Reciprocal Rank

NDCG Normalized Discounted Cumulative Gain

RAM Random Access Memory

ROC Receiver Operating Characteristic

SMS Short Message Service

TAC Type Allocation Code

Telco Telecom Company

U.S. United States

UMTS Universal Mobile Telecommunications Service

XGBoost Extreme Gradient Boosting

Chapter 1

Introduction

In the past, handsets were simple and limited to making or receiving calls and text messages. Nowadays, handset devices can be almost as powerful as full-size computers – laptops or desktops – enabling a multitude of new use cases, like browsing the web, watching videos and even some impressive uses like fast image processing and machine learning. Handset devices also come in a large variety of sizes, brands, colors and materials. Brands update their product line frequently, with top companies such as Apple and Samsung updating them yearly. Many customers want to update their handset to either get updated specifications, to get new interesting features like facial recognition, wireless charging or even swivel cameras, get the newest design or simply to replace their old and worn out device due to it having a "cracked" screen, a drained battery or being obsolete and unable to handle recent apps that are technologically more demanding. All these reasons explain the relative short upgrade cycle of handsets with data suggesting that, in the U.S., the average upgrade cycle is only thirty two months in 2017 and twenty five in 2016 [NPD Connected Intelligence, 2018].

These very frequent updates from customers present a great opportunity for Telcos to present them with personalized service packages. A direct consequence of being precise when recommending handset upgrades is the direct increase in sales, resulting in increased revenue for the Telco and increased satisfaction for the user due to having an accurate and personalized recommendation presented to them. These upgrades scenarios also provide a great opportunity to get more value from the customer by extending existing service subscriptions or subscription upgrades.

The main goal of recommender systems is to provide the most likely items the user would be interested in. These recommenders provide suggestions on items the user probably did not even know about but would be interested in, resulting in even bigger customer satisfaction due to the discovery of a novel item for him. Recommender systems can potentially generate a significant increase in sales [Fleder and Hosanagar, 2007], with every small improvement in the system being a possibility for more profit. With this in mind, applying recommendation algorithms to the problem of handset device upgrade in a Telco is an interesting approach capable of having impact on the customer lifetime value.

This project focuses on the development of a recommendation system capable of learning a model that predicts the best handset offer to individual customers. The algorithm should be able to take into account not only the individual user preferences, but also previous purchases and constraints imposed by the business logic (for example, to only recommend same or higher quality devices).

1.1 Motivation

Given the short product life cycle of handsets, Telco customers typically upgrade their handsets frequently. These make excellent opportunities to present customers with personalized service packages making it crucial to be precise when recommending handset upgrades. Even by ignoring the aforementioned, useful recommendations will translate not only in more satisfied customers but will also bring more income to the company.

The existing solutions used by the Telco are based on popularity (non-personalized) and have a clear potential for improvement. For example, these methods lack support for new devices (cold-start problem, explored in subsection 2.5) and do not take into account the user profile or the specifications of the previous phone. They are also not able to deal with business rules. Given that they are based on popularity, the least popular items will be ignored (long tail effect, detailed in Section 2.6). Having this room for improvement reveals the relevance of this research topic for the business.

As a scientific challenge, the problem of handset recommendation is one that, from a scientific standpoint, has not still been thoroughly approached. Moreover, there is potential applicability to similar problems, with low interactions number, such as recommendation for tablets, laptops, cars and possibly services such as insurance packages.

1.2 Problem definition and objectives

The aim of the project is to develop a recommendation system that, given an individual customer and his current handset, is capable of predicting a short list of devices that he would most likely buy next, sorted by the likelihood of the individual customer swapping to them. This problem is different from more classic recommendation problems as we need to also take into account the handset the user currently has and adjust the recommendations accordingly. The recommender should also be able to deal with individual customer preferences, large purchase histories, low number of interactions per user, as well as constraints imposed by the business model – for example, to only recommend same of higher quality devices. The scope of this work is limited to the recommendation system, and not on predicting whether a user will in fact swap his device. This task has already been accomplished by a separate and already implemented model in the company.

The objectives are the following:

- To gain further insight into the problem, by carrying out empirical studies based on real-world data provided by the Telco.
- To design, implement and evaluate a recommender system capable of predicting a better handset offer for individual customers than the already existing baselines.
- To take advantage of the existing individual customer and handset data.

1.3 Contributions

Two types of contributions have resulted from this work, scientific contributions and technological innovation for the industry (Telco).

The scientific contributions from the developed work are the following:

- Review of the state-of-the-art in the field of recommender systems including challenges and ways of evaluating their performance, applicable to the problem at hand.
- Proposal of a content-based and an hybrid recommender system using customer's data and handset device specifications.
- Case study and solution to a problem that, to the best of our knowledge, has still not been approached by the scientific literature on the area of recommender systems in handset recommendation.

The contributions for the Telco are as follows:

- A recommender model with more accurate recommendations and better explainability than the already existing models.
- Extra insights on the data result of data exploration studies realized.

1.4 Organization

After this introductory chapter, the dissertation is divided in three more chapters.

In Chapter 2 the different main types of recommendation approaches are detailed along with some models based on these same approaches. Then, an exploration into how evaluation of recommender systems can be performed and with which metrics, some of the challenges associated with the recommendation task and lastly the long-tail effect on the recommendation problem.

Chapter 3 starts with describing the used methodology, exploratory data analysis on the available data followed by detailing the already existing solutions which will be used as baselines. The developed solution itself is then detailed, in three different sections, corresponding to the different approaches tested. Finally a summary on the entire chapter is given.

To finish, Chapter 4 gives a conclusion on the developed work as well as suggestions for future work with the goal of further improving the given solution on this dissertation.

Chapter 2

Recommender systems

Recommender systems are a broad field of research. It can be mainly distinguished in its approaches in regards to the way data is used to produce a model, but also by the predictive task. These tasks can be in the form of rating prediction or item ranking.

Many different approaches have been studied in the literature to tackle the recommendation problem. The most popular and widely implemented ones are collaborative filtering, content-based and hybrid approaches. These distinguish themselves accordingly to whether they use content information on the user and items, the user-item interactions history or both, respectively.

There are also two predictive tasks that can be performed [Steck, 2013], item ranking and rating prediction. In ranking prediction, a small list of size N is generated from all the available items for all users sorted in order of most likelihood of relevance to the user while in rating prediction the goal is to predict the rating each user would give to an item.

2.1 Collaborative filtering recommenders

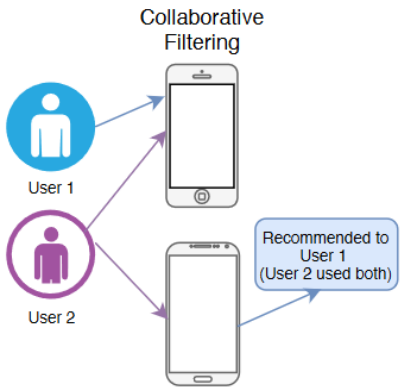


Figure 2.1: Collaborative filtering representation

Collaborative filtering [Schafer et al., 2007] (illustration in figure 2.1) bases its predictions on the historical data of user ratings on specific items, often organized in a user-item interaction matrix, to make recommendations to the users. There are two main types of ratings: implicit and explicit [Zhao et al., 2018], with implicit data being the most commonly available as it typically does not require explicit user intervention. Table 2.1 shows an example of a user-item matrix using explicit ratings while Table 2.2 uses implicit ratings as feedback. Another noteworthy aspect in regard to these matrices is that they are usually very sparse, as users tend to only interact with a very small subset of the items.

	I1	I2	I3	I	I5
U1	3			1	1
U2		2	1	1	
U3	5		4		1
U4	2		4		

Table 2.1: User-item matrix with explicit ratings

	I1	I2	I3	I4	I5
U1	X			X	X
U2		X	X	X	
U3	X		X		X
U4	X		X		

Table 2.2: User-item matrix with implicit ratings

Historically, collaborative filtering approaches were further divided into model-based and memory-based or neighborhood-based. Model-based collaborative filtering learns a model from the data that is then used for making predictions. Neighborhood-based methods preserve all the interactions with recommendations being made based on the neighbourhood of users or items in the rating matrix. Neighborhood-based methods have the benefit of being much simpler albeit have problems with scalability (Subsection 2.5) and sparsity (Subsection 2.5), both very common in recommendation problems.

Neighborhood-based approaches typically provide a good baseline and are simple enough to apply. This approach can be further divided in the two following ways, with both being applicable to the top-N ranking or rating prediction problem.

- User-based collaborative filtering: Exploits the principle that similar users have similar ratings on the same items.
- Item-based collaborative filtering [Sarwar et al., 2001]: Uses the notion that similar items are rated in a similar way by similar users. This approach is usually the most popular as the number of users tend to largely outweigh the number of items and also due to the user profile being typically more dynamic than the items ratings, which allows for a more efficient and less frequent computation.

In both cases, we need to define a similarity metric. In high-dimensional spaces, angular metrics such as the Pearson correlation (2.1) [Sarwar et al., 2001] and cosine distance (2.2) [Linden et al., 2003] are more robust than euclidean metrics as they are not limited in regards to the dimmensionality of the data, where we could have as many dimensions as we have users/items.

$$pearson(v,w) = \frac{\sum_{u \in U}(R_{u,v} - \overline{R}_v)(R_{u,w} - \overline{R}_w)}{\sqrt{\sum_{u \in U}(R_{u,v} - \overline{R}_v)^2}\sqrt{\sum_{u \in U}(R_{u,w} - \overline{R}_w)^2}} \tag{2.1}$$

Where $R_{u,v}$ represents the rating of user u on item v and $\overline{R}_v$ the average rating on the $u-th$ item.

$$cosinesimilarity(v,w) = \cos(\theta) = \frac{v.w}{\|v\|\|w\|} = \frac{\sum_{i=1}^{n} v_i w_i}{\sqrt{\sum_{i=1}^{n} v_i^2}\sqrt{\sum_{i=1}^{n} w_i^2}} \tag{2.2}$$

Other collaborative filtering algorithms

The formerly discussed neighborhood-based algorithms have many advantages such as being simple, easy to understand and providing great interpretability on the results but they do, however, have some disadvantages [Aggarwal, 2016].

One of the main disadvantages is their impracticality with large-scale data, as these algorithms require $\mathcal{O}(N^2)$ time and space complexity. The other is related to sparsity, which causes limited coverage on the trades. This results in increased difficulty in rating items, as the nearest neighbours could, collectively, not have had the relevant items to calculate the distances.

For these reasons, other algorithms will be explored below.

Clustering can also be applied to the recommendation problem [Cuong et al., 2011] in which groups (clusters) are created to represent users with similar interests. These clusters can also be applied to items, resulting in groups of items similar to each other, in the same group. To make a recommendation using this method, we first calculate the clusters, and then we make the predictions. To make a prediction for a specific user on a certain item, we simply need to average the opinions of all the users in the cluster for that item. Clustering approaches typically produce less personalized results (users are seen only as a small part of a bigger cluster) and produce worse results than nearest neighbour algorithms [Breese et al., 1998]. However, they are useful in extremely sparse settings.

Association rules mining [Agrawal et al., 1993] tries to determine sets of items that are very correlated to other sets and thus establishing relationships between these sets, typically referred as transactions, to make recommendations. An example of such a rule would be one such as, in the context of a online store, clients who frequently bought items A and B, also bought C, thus making item C a good recommendation for users buying A and B. To determine how closely correlated two item sets are two measures are used, support and confidence. Support measures how frequent an item set is in the dataset while confidence measures the strength of the rule by determining how often the rule was found to be true. Multiple algorithms for discovering frequent item sets exist with Apriori [Agrawal and Srikant, 1994] and FP-Growth [Han et al., 2004] being examples of them.

Matrix factorization [Lee and Seung, 1999] [Koren et al., 2009] is one of the most commonly used techniques in collaborative filtering. This technique gained popularity with the Netflix prize challenge [Bennett et al., 2007].

Matrix factorization (figure 2.2) is a latent factor model [Everitt, 1984]. Users and items are mapped into the same latent space that is automatically calculated from past user ratings, having then a latent

characterization of both products and items instead of the classical approach of observable variables. The number of latent variables is a user-defined hyperparameter.

This characterization of items and users is typically initiated with random values. The goal is to reduce prediction error iteration by iteration by adjusting these values to converge to the observed ratings. The result of the dot product between the latent user and item features will result in expected rating on that item by that user. Singular value decomposition [Golub and Reinsch, 1970] is commonly used to learn the feature matrices.

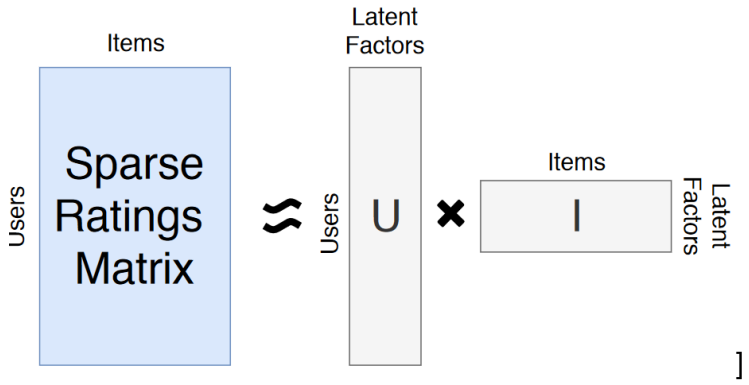


Figure 2.2: Matrix factorization representation

Bayesian personalized ranking [Rendle et al., 2009] is a popular learning to rank method for recommendation. BPR is usually applied to a matrix factorization model where instead of considering the rating given by a user to an item, we consider the pairs of items for each user which tells us that the user prefers one of the items over the other. As per typical in bayesian methods [Bernardo and Smith, 2009], we will need to calculate the posterior probability using the prior probability and a likelihood function. The bayesian method can be seen at (2.3) with H being the hypothesis and E the evidence where $P(H|E)$ is the posterior probability and $P(H)$ is the prior probability.

$$P(H|E) = \frac{P(E|H) \cdot P(H)}{P(E)} \tag{2.3}$$

Deep neural networks [LeCun et al., 2015] has garnered increasing interest in a multitude of different machine learning areas such as natural language processing and image and sound processing. Its definition is related to the usage of neural networks [Zurada, 1992] that are composed by many different layers, where if a graph of the network was drawn, it would be a very deep one.

Deep learning has also been applied to the field of recommendation and, accordingly to the research done by [Zhang et al., 2019]], the benefits to using such models are their capability of modeling the non-linearity in the data, ability to better deal with sequenced inputs, eases the necessity on the need to feature engineering, allows the usage of different data sources such as images, text and sound and offer a higher flexibility in modeling the problem. Some of their drawbacks are associated with poor interpretability of the results, need for larger amounts of data and considerable hyperparameter tuning. Autoencoders, convolutional neural networks and recurrent neural networks [Goodfellow et al., 2016] are

deep learning models typically applied in recommendations.

2.2 Content-based recommenders

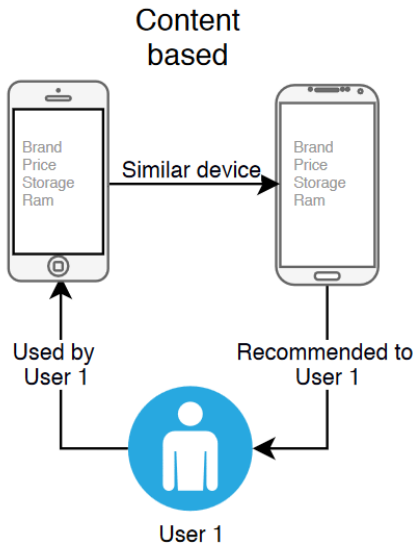


Figure 2.3: Content-based representation

With content-based recommenders [Lops et al., 2011] (illustration in figure 2.3), two main types of data are distinguished. The first, the user profile [Pazzani and Billsus, 2007], characterizes the user in terms of preferences, behaviours and other relevant aspects. This profile can be initially built in many ways and incrementally tuned, with initial information possibly being given by the users during registration on a service, or derived from other possible data sources available on the user. Over time, the user could manually update his profile to show his new interests or the system could learn these, from the interactions the user had with the items.

The second, item information, can come in a multitude of ways but it is mainly distinguished between structured, semi-structured and unstructured. The type of information available is mostly dependent on the recommendation scenario. In the scenario of movie recommendation, for example, we could have structured information on the movies like it's genre, duration and release date or unstructured information such as text.

Due to the nature of the data previously described, content models are not particularly dependent on a strong interaction history by the users on the items, as is the case with collaborative-filtering. For this reason, content-based models are particularly resistant to cold start (Subsection 2.5) but more susceptible to it's suggestions having reduced novelty and serendipity (Subsection 2.5) as only the information on the items and the user profile are looked at resulting in accurate albeit obvious recommendations.

With content-based recommendations, there is the added importance of determining the correct methodology for dealing with the problem, and not just choosing the correct algorithm to use, as with collaborative-filtering recommenders.

Some of the algorithms used in collaborative filtering such as clustering (Subsection 2.1) and deep learning (Subsection 2.1) are also applicable to content recommendation, with the latter being specially good when applied to higher dimensionality data, typically present in such problems.

Decision trees can be applied to content recommendations. Techniques such as bagging, boosting and gradient boosting can use the previously mentioned trees to help reducing the variance or bias [Bauer and Kohavi, 1999] as well as help providing a more robust model. Below, these approaches will be detailed, as well as a library that makes use of gradient boosting.

Decision trees [Quinlan, 1986] follow a tree structure where in each node we have one of the attributes in the data being tested on a certain condition, resulting in a binary split. A tree can have multiple splits, with splits possibly applied to the same attribute multiple times. The last elements of a tree, called leaves, either output a label – in case of classification – or a numerical value – in case of regression [Torgo, 2000]. The modeling task of decision trees involves finding the best splitting criteria, to decide which attributes and conditions to use, as well as when to stop growing the tree. The most common ways of stopping tree growth – and thus avoiding overfitting – is to use either a predetermined tree height or by performing tree pruning. Multiple methods can be used for determining the splits, such as Gini impurity, information gain [Sharma and Kumar, 2016] and variance reduction.

Decision trees are simple, fast and easy to debug. For this reason they are typically used in ensemble techniques, that use them as weak learners – algorithms that provide better predictions than random guessing – to improve the predictive ability of standalone algorithms. Ensemble techniques such as Bagging or Boosting can be applied, where a single aggregate model composed of multiple weak models is learned, significantly elevating the predictive power of these models, thus also providing more accurate recommendations.

Bagging [Breiman, 1996] – or bootstrap aggregating – splits the training data using bootstrap sampling. Multiple versions of the original dataset are created by performing sampling with replacement from the original dataset, until the size of the original dataset is reached. Then models are trained individually in each sample. Global predictions are obtained by aggregating predictions from individual models, typically using majority vote in classification, and the average in regression. An example of bagging being applied to decision trees are random forests [Liaw et al., 2002].

Boosting Boosting [Freund et al., 1999] is an ensemble technique that combines weak learners into strong ones. The term "weak" refers to the weak requirement that base models are only better than random guessing. To achieve this, models are built sequentially by re-weighting data points according to the predictions outcome of the ancestor model. In a classification problem, each example's weight is increased if the predecessor model misclassifies it, and decreased otherwise. This way, new models focus more on the wrong predictions, potentially correcting them.

Gradient boosting Gradient boosting [Friedman, 2000] uses gradient descent to optimize the loss function. Gradient descent is an optimization algorithm that can applied to a wide range of loss functions. Gradient descent measures the local gradient of the loss function for some parameters and adjusts the model parameters – the weights of weak models – in order to minimize it. One of the most important hyperparameters (user-defined) to set is the learning rate, which affects the step length on descent. By

setting too small a step, the algorithm will take too many iterations to find the minima and by setting it too high, we risk passing the minima.

2.3 Hybrid recommenders

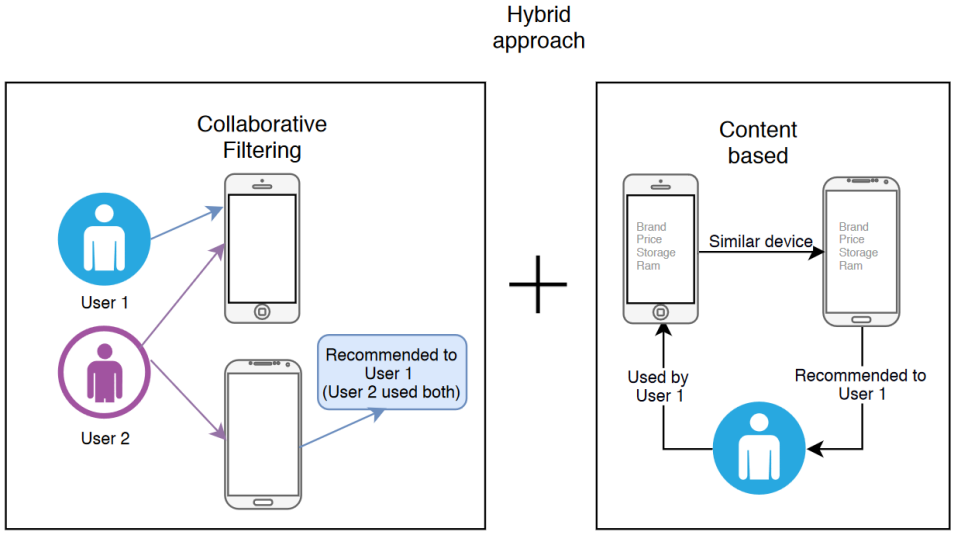


Figure 2.4: Hybrid representation

Hybrid recommender systems [Burke, 2002] (illustration in figure 2.4) combine content-based techniques with collaborative filtering so that each separate approach can compensate for the downfalls of the other. For instance, we are able to take advantage of the whole user-item history as well as make recommendations on items without significant ratings, by using the characteristics (attributes) of the items. These different approaches can be combined in multiple ways [Burke, 2002], as seen in Table 2.3.

2.4 Evaluating recommender systems

Top-N recommendation is a ranking problem where the output is a ranked list of predictions, by user. To ensure that their performance is correctly gauged, metrics need to consider the position of the predictions in regard to the recommendation list. The field of information retrieval shares a lot of similarities with that of ranking recommenders as both approaches output their predictions in the form of a ranked list. For this reason, most of the metrics shown here can be applied to both problems.

The following metrics are used for evaluating the performance of ranking recommender systems:

- Precision
- Recall
- F1

Table 2.3: Hybridization methods [Burke, 2002]

Method	Description
Weighted	The results of each recommender is calculated independently and then combined to make a recommendation.
Switching	The hybrid switches from recommender to recommender depending on the context of the situation.
Mixed	Generating recommendations from each recommender and presenting them all.
Feature Combination	Utilization of different data-sources, both content and collaborative,to achieve the hybrid.
Cascade	A recommender is used to refine the recommendation of another.
Feature augmentation	The results from a recommender is fed to another recommender that uses this data to make his own recommendations.
Meta-level	Uses as input a model learned from a different recommender. Differs from feature augmentation as we are not creating features for the input but instead using the entire model as input.

- Hit rate (HR)
- Average reciprocal hit-rank (ARHR)
- Mean average precision (MAP)
- Normalized discounted cumulative gain (NDCG)
- Mean reciprocal rank (MRR)
- Area under ROC curve (AUC)

For some of these metrics, such as HR, MAP and NDCG, the concept and notation of a cutoff needs to be introduced. In a recommendation problem, the goal is to output a sorted list of recommendations that can vary in number. Some metrics only take into account the first k items of the list, in what is called the list cutoff.

A popular convention for notating the metric and it's respective cutoff is to write the metric acronym followed by an @ and its cutoff value. For example, if we were to use hit rate with a cutoff value of five, the notation would be HR@5.

2.4.1 Precision, recall and F1

Precision, recall and F1 are computed taking into account an unordered set of documents while the others take the position in the list (the rank) into account. With precision (2.4), we are calculating the quality of each recommendation while with recall (2.5) we are calculating the ratio of relevant items guesses. With precision we are answering the question how many selected items are relevant and with recall we are answering how many relevant items were selected. F1 score (2.6) is the harmonic mean of the recall and the precision.

$$precision = \frac{TruePositives}{TruePositives + Falsepositives} \tag{2.4}$$

$$recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{2.5}$$

$$F1 = \frac{2 \times recall \times precision}{recall + precision} \tag{2.6}$$

2.4.2 Hit rate

Hit rate (2.7) is associated with singular choices by the users and gives insight on whether a user chose one of the items on the recommendation list. A value of one (hit) is assigned to that user if the item he chose was in fact on the list of recommendation and 0 (no hit) otherwise. With such a metric, if we made a top-N recommendation list for C clients, with I items equal to the total number of possibly existing items, we would obtain a result of 100% (one) hit rate, as the order in the recommendations list are not relevant as with this metric we are only interested in knowing whether the item was on the recommendation list regardless of position.

$$hitrate@K = \frac{1}{numRecommendations} \sum_{i=1}^{numRecommendations} Hit@K \tag{2.7}$$

2.4.3 Average reciprocal hit rank

Average reciprocal hit-rank is an adaption of the hit rate metric where instead of just counting the hits, we also take into consideration the position on the recommendation list (rank of the hit) for each user. As such, ARHR is a weighted version of the HR metric that also takes into account the position of the hit on the list.

$$ARHR = \frac{1}{numUsers} \sum_{i=1}^{numHits} \frac{1}{positionOnList} \tag{2.8}$$

2.4.4 Mean average precision

Average precision takes into account the position of the items on the list as opposed to precision. Average precision computes the ratio of relevant items in a recommended list of size L for a given user such that if we had a user recommended five items where only the first, third and the last were right, the AP@5 (2.9) would then be:

$$AP@5 = \frac{1}{5}(\frac{1}{1} + \frac{0}{2} + \frac{2}{3} + \frac{0}{4} + \frac{3}{5}) \approx \frac{2.27}{5} = 0.454 \tag{2.9}$$

To find the Mean average precision (2.10) we simply average all of the average precision for all the users.

$$MAP@K = \frac{1}{N}\sum_{n=1}^{N} AP@K \tag{2.10}$$

2.4.5 NDCG

The idea behind discounted cumulative gain (DCG) (2.11), is that items appearing lower on the list have less interest and as such should be penalized, so that their relevance is logarithmically reduced to the rank of the item. To calculate the NDCG (2.12), we first need to find the ideal DCG that corresponds to re-ordering the list so that the higher relevance items appear on the top of the list and as such, the ideal DCG would simply be the DCG of this re-arranged list and NDCG the division of the DCG by the ideal DCG.

$$DCG@k = \sum_{n=1}^{k} \frac{rel_i}{log_2(i+1)} = 1 + \sum_{n=2}^{k} \frac{rel_i}{log_2(i+1)} \tag{2.11}$$

$$NDCG@k = \frac{DCG@k}{DCG@k_ideal} \tag{2.12}$$

2.4.6 Mean reciprocal rank

With mean reciprocal rank (MRR) (2.13) we are only interested in the rank of the first hit we get. As such, MRR simply calculates the reciprocal rank, the inverse of the first hit, for each user, and then averages it over all the N users.

$$MRR = \frac{1}{N}\sum_{n=1}^{N} \frac{1}{rank_i} \tag{2.13}$$

2.4.7 Area under ROC curve

A Receiver Operating Characteristic (ROC) is a curve that is created by plotting the true positive rate (recall) against the false positive rate (fallout (2.14)). By calculating the area under the ROC curve (AUC) we can evaluate the performance of our model as well as compare it against the AUC's from other models

where an ideal recommender would have an AUC of 1 and a always wrong recommender a score of 0. In Figure 2.5 we can see three ROC curve plots, where in red we have the case of a perfect recommender, in green a recommender making random guesses and in blue what we would expect to see from a typical recommender.

$$fallout = \frac{falsepositive}{falsepositive + truenegative} \tag{2.14}$$

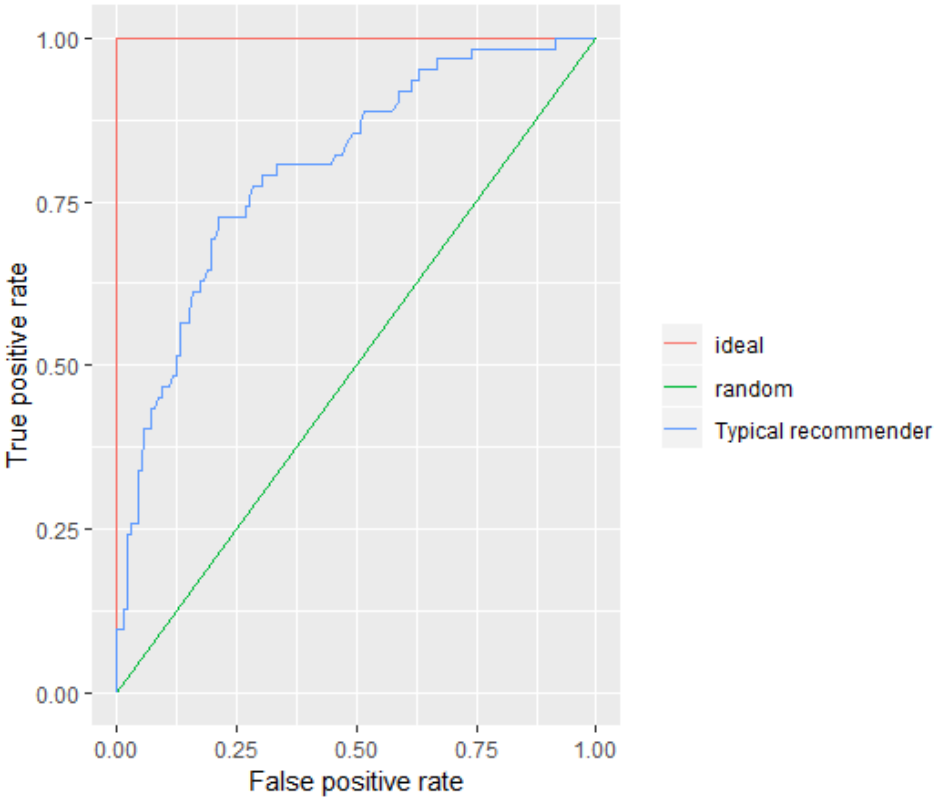


Figure 2.5: ROC curves for various recommendation scenarios

2.5 Beyond accuracy: Recommender systems challenges

The recommendation task is associated with a few challenges [Khusro et al., 2016] with a few of these already been called out previously during other parts of this dissertation.

Therefore, in an effort to better describe these problems the following subsections will detail these challenges, with the long-tail problem being it's own separate Section (2.6), due to it's relevance to the problem at hand.

Cold start [Lam et al., 2008] in recommender systems can happen when either a new item or user joins the platform and there are not enough ratings to accurately make predictions. This challenge goes

hand-in-hand with collaborative filtering systems while content-based tend to suffer less from it due to their reduced focus on historic ratings data. There are some far from perfect solutions such as asking the user to input their preferences on sign up, using available information on the user to find similar users based on basic criteria such as age, region and time of registration or by just using non-personalized predictions. As with new items, possibilities range from grouping this new item to an already existing very similar one or have specialists review the new items beforehand.

Confidence [Shani and Gunawardana, 2011] is the recommender trust in the predictions that it outputs. These confidence values can be presented to user to allow them to gain further insight into the recommendations. A user presented with a low confidence recommendation may be skeptical of it and do further research on the item whilst an item with very high confidence can be accepted by the user without much doubt. These confidence values can also be used by the companies, to make sure the time is opportune when presenting a suggestion (like a bundle), by, for example, only making these suggestions to users whose recommendations exceed a certain threshold (e.g., only present users with a certain campaign if they have >90% confidence in the recommendation).

Diversity [Çano and Morisio, 2017] is easy to understand in practical terms. A music streaming service provides users with music they would probably be interested in. If a user really likes a certain artist, the recommender could only recommend music from that artist or musics very similar to the ones he had listened to. In top-N recommendation, with recommendation lists of increased size, this is especially a problem, as the user should have presented to him items that are not too similar to each other. However, providing a diverse recommendation list, can have a negative impact on accuracy. Diversity is popularly measured by using item-item similarity, with high item-item similarity suggesting a non-diverse recommendation list.

Scalability [Sarwar, 2001] problems tend to arise when the number of users or items increase in a way that makes recommendations either too slow, requiring absurd amounts of memory or compute power. In recommender systems, when dealing with recommendation in movies, music and online stores, it is not uncommon for recommendations to be made on millions of users and items. It is necessary to design recommender systems accordingly to the scale of the problem to minimize these challenges. This types of problems are even further pronounced in online recommender systems, where users expect instant, personalized and accurate predictions. Some solutions to the problem include clustering (Subsection 2.1), dimensionality reduction [Sarwar et al., 2000] and bayesian networks, detailed in Subsection 2.1.

Serendipity and novelty [McNee et al., 2006] has to do with the ability of recommenders suggesting unexpected items that the users themselves would not know they were interested in while novelty is related to recommending less known items. In a music recommendation task, it is reasonable to assume that users will like songs from artists they have liked in the past. However, making such recommendations will not be unexpected for the users whereas a recommendation on a new artist from other people who like both the artists would. This has the benefit of increasing user satisfaction by possibly showing him new items they would love.

Sparsity [Sarwar, 2001] happens due to the low amount of items rated by each user when compared to the total available items on the catalog. Having very sparse data has consequences, especially in the

case of neighbourhood-based algorithms, where computation becomes increasingly more difficult and coverage gets limited, resulting in items that cannot be rated, as previously discussed in subsection 2.1. Two possibilities of countering this effect would be to use dimensionality reduction techniques [Sarwar et al., 2000] and using either content-based or hybrid approaches when possible.

Synonymy another import aspect to take into account with the data used for recommendation is that of synonymy, where apparently different items could in reality be the same and the presence of synonyms in the attributes of either user or item data, for example, multiple song categories exist with some being either a sub or super-type or just having different name for representing the same type. All these effects lead to this synonymy effect with recommenders having difficulty making associations between these different labels resulting in poorer performance, To counteract this effects, it is important to perform exploratory data analysis and do pre-processing on the data before feeding it to the model.

2.6 The long-tail

An important aspect to take into account when dealing with recommender systems is that of the concept of a long-tail [Anderson, 2006], where items with very low demand (long-tail) can collectively sell more than the most popular ones (head) individually. This concept is related to the Pareto principle, also known as the 80-20 rule, which states that 80% of the effects are caused by only 20% of the causes. The relation with recommender systems here is very important, as if we only recommend popular items (head) and ignore the less popular ones (long-tail), we could miss what could collectively be an even larger amount of hits. To have an effective recommender it is then important to not only focus on the head but also on the tail of items.

A representation of such a long tail can be seen on Figure 2.6 where data from the MovieLens 20M dataset [Harper and Konstan, 2015] was used with the red trailing line representing the half split in total ratings.

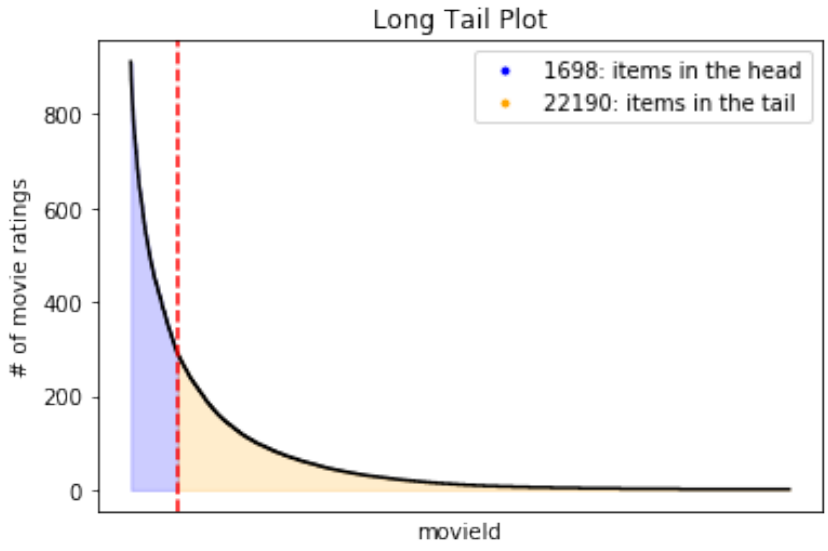


Figure 2.6: Long tail plot of ratings on movielens 20M dataset

Chapter 3

Hybrid handset recommendation

In this chapter, the methodology used for dealing with the recommendation problem is detailed, beginning with a formal and generic methodology made for use in data science projects followed by the methodology taken for this dissertation. Exploratory data analysis is then presented summarizing the available data, their sources and limitations. Solutions are then presented to deal with these limitations.

In the following sections, the various approaches taken to solve the problem of handset recommendation are detailed, starting off by the description of the existing solutions at the Telco which are used as baselines to compare with these new methods. Then, the explanation and results are presented for the collaborative-filtering, content-based and hybrid approaches.

The chapter ends by summarizing the chapter and the results for each of the approaches compared to the baselines.

3.1 Methodology

It is important to have a well-defined methodology, as following a methodology is following a systematic way to solve a problem, which in turn further guarantees the chances of the project being successful.

As such, IBM's foundational methodology for data science [J. B. Rollins, 2015] accurately portraits the steps which any data science project should partake in from inception to deployment. This methodology consists of ten steps detailing the process a data science related project should follow in order to achieve its goal. This methodology steps can be seen in Figure 3.1.

The ten steps will be briefly summarized below:

1. Business Understanding: In this first step, the problem to be solved should be well defined with the business specialists playing the main role here of defining what is wanted, what limitations are imposed and with which time restrictions.
2. Analytic Approach: Specification of the analytic approach to follow to solve the problem defined in the previous step. In this step, it is expected to know which machine-learning technique should be applied and how it should be tested and implemented.
3. Data Requirements: Definition of data content, formats and representations to use accordingly to

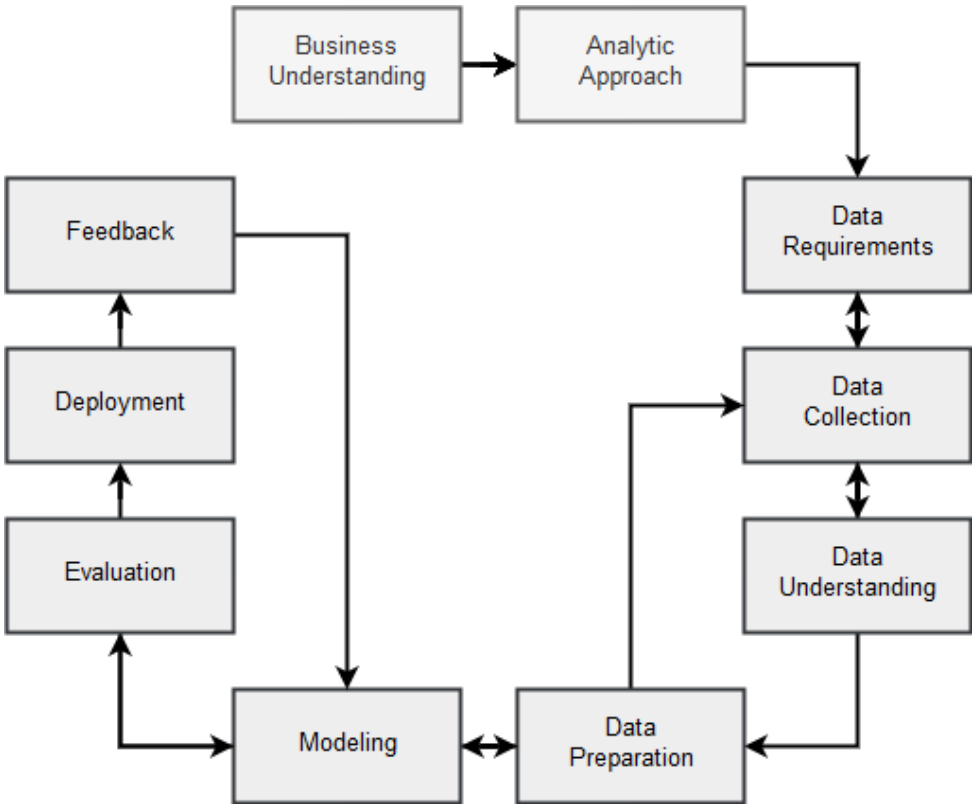


Figure 3.1: IBM's Foundational Methodology for Data Science

what was defined in the previous steps.

4. Data Collection: Identification and gathering of the relevant data sources and checking if the data matches the data requirements previously specified and/or if further data needs to be collected.
5. Data Understanding: Data exploration using descriptive statistics and visualization techniques to evaluate the quality of the data and obtain initial insights about the data.
6. Data Preparation: Data cleaning and feature engineering are the focus in this step making this usually the most time-consuming step in the methodology, but also one of the most important as how well the data preparation job was done will affect the models performance as this is the data which will be used in them.
7. Modeling: Development of predictive or descriptive models using the analytic approach and data set defined in the previous steps. This modeling process frequently provides intermediate insights with these being a great opportunity for refinement of the data and model used. It is also typical to test multiple algorithms and their features to find the better solution.
8. Evaluation: Crucial step in which the developed model is evaluated to check if the business problem was fully addressed, as well as if the quality of the solution is satisfactory using diagnostic measures and other outputs such as tables and graphs.
9. Deployment: With the model properly evaluated, it is then deployed into production or comparable test environment. Typically, it is only deployed in a limited manner until its performance has been further assessed. The deployment also varies in complexity with some deployments being as simple as generating a simple report with suggestions with some others needing to be integrated

in an already existing sophisticated workflow with the possibility of automated scoring using an existing application.

10. Feedback: In this step, the business obtains feedback on the impact of the model on the environment in which it was deployed with the data scientists using this feedback to further improving the model or data used in it.

From these ten steps detailed above, the first eight were accomplished with the deployment step not being performed nor the feedback step due to its dependency.

For the analytic approach to take with the project, it was defined that the focus would first be put on understanding the existing baseline algorithms and their data sources. This not only has the obvious benefit of understanding what has previously been done and why, but also gives a better understanding on the problem to solve as well as it's business ramifications.

During the baseline exploration, limitations with the data were found which required the collection of new data. Both new and old data sources were then analyzed to gain further insights into the data and possibly give clues for how to model the problem and which results to expect from it.

For the modeling approach taken, they can be mainly split among three stages, first with collaborative filtering, with only historical trades data being used, secondly the content-based approach using information on the handsets and their users, and lastly the combination of both approaches models.

3.2 Exploratory data analysis

Available data detailed below has been provided by the Telco for the sake of developing this dissertation. As such, some of the values provided will be presented in relative terms for confidentiality, where instead of presenting the absolute raw values, we divide them by their group maximum. Some of the data obtained has been through web scraping and will be identified as such.

3.2.1 Trade data

Trade data is the most important available data we have available, specially in the case of collaborative filtering. This information is related to the company's customers and their usage of handset devices. We have historical data going back from January, 2002 to present detailing customers and their handset devices history.

With this data, it is possible to know when each customer stops using one device and starts using another, which is considered a replacement. This information is fundamental for modeling the problem.

This data is updated in a monthly manner, with user's handset for that month corresponding to his most used device in that same month. This brings some problems into the table, as we do not have the latest data and it is possible to have temporary transitions, such as the case of a temporary replacement in case the previous handset got damaged and needed repair.

Due to the monthly nature of the dataset and possibility for temporary transitions, a criterion was chosen: a replacement will only be considered if the previous device has a first usage date on the network, for any customer, strictly inferior than that of the traded to device. This criterion is different than the previous

existing one at the company, where a trade was only considered if the customer remained with the same phone for at least three successive months. Only feature phones – simple device with main focus being on making calls –, and smartphones – powerful phones capable of performing many of the functions of a computer – were considered in this data with devices such as Wi-Fi hotspots being discarded, as they fall out of the focus of the dissertation.

Data analysis and exploration presented below uses data ranging from January 2018 to December 2018, unless otherwise specified.

Before application of the new criteria our trade data order of magnitude can be seen in Table 3.1 [1].

Table 3.1: Data order of magnitude for trade information (no criteria applied)

Group	Order of magnitude
Users	10^5
Handsets	10^3
Trades	10^5

After application of the new criteria our relative cardinality loss on trade information is shown in Table 3.2.

These results show that approximately 3% of the replacements are lost by using this criterion. This loss is acceptable for two reasons: enough trades exist, after its application, for training a good model and the feedback given from these trades is not desirable from a business standpoint.

This criterion is then a better option as it also takes into consideration the most recent trades without having the additional two months delay from the previously existing criterion, relevant to reducing the cold-start problem.

Regarding the devices being sold at the Telco as of 28/05/2019, less than one hundred devices are currently in catalogue available for sale.

Table 3.2: Data cardinality for trade information (after criteria applied)

Group	Relative data loss
Users	$\approx 2\%$
Handsets	0%
Trades	$\approx 3\%$

As can be seen in Table 3.3, approximately 74% of users who had at least one trade made in the considered time range (2018 range) only did so once, with approximately 18% having made two and around 6% having made three trades. We also have some rarer cases (0.10%) where clients traded their phone very frequently (seven or more).

In Figure 3.2, we show the trades over time from 2013 to 2018 in a relative scale. A very large increase in

[1] Absolute values not presented for the sake of confidentiality

Table 3.3: Percentage of users by number of trades made

Number of trades per user	Percentage of users
1	73.65%
2	17.80%
3	5.86%
4	1.83%
5	0.56%
6	0.20%
7 (or more)	0.10%

upgrades can be observed in 2014 when compared to the previous year, as well as steady but slower increments until 2018, where we see a small decrease in upgrades in comparison to 2017, which the conclusions from [NPD Connected Intelligence, 2018] also suggest. Upgrades tend to increase steadily from January to December, with a big drop after December, possibly related to the end of the festive season where people tend to upgrade more.

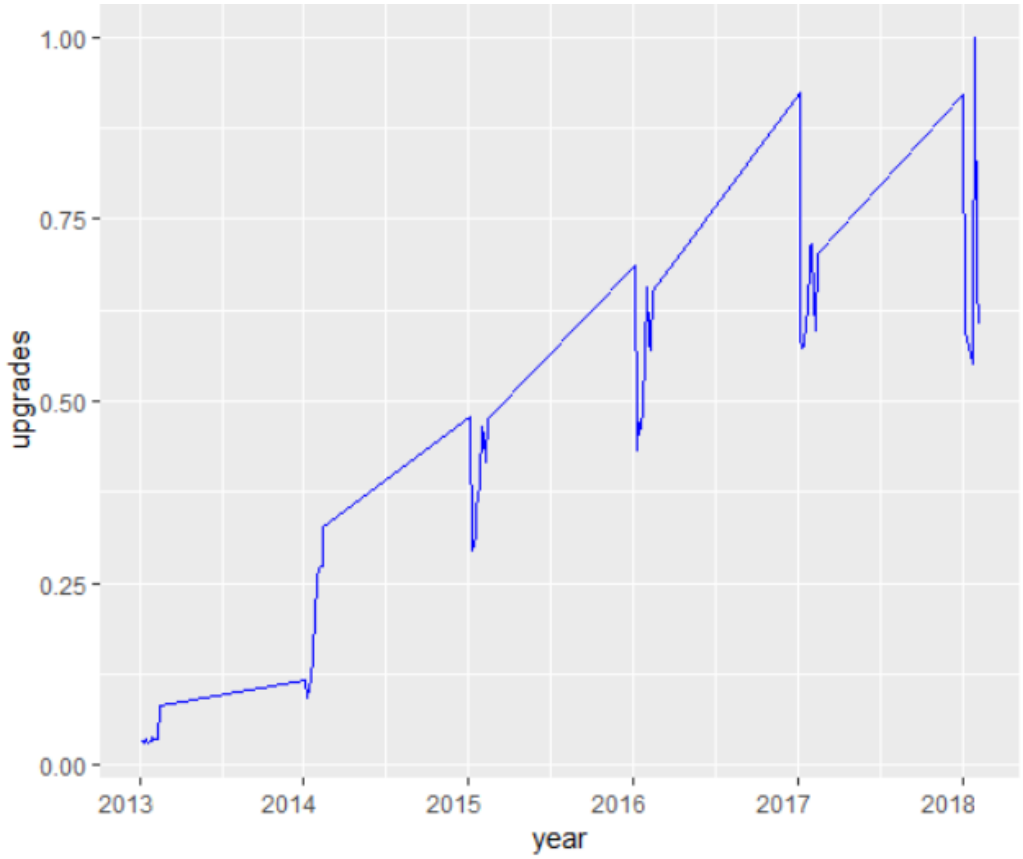


Figure 3.2: Trades over time, relative scale

The long tail plot for the considered data can be seen in Figure 3.3. The red line depicts the fifty percent split for the head and tail of the data. In our case, we have a pronounced long tail which is evident when compared to the previous tail plot shown in Figure 2.6. With only 81 devices in the head we would cover 50% of total trades which contrasts deeply with the reality on the tail, where we would need 5150 handsets to cover the same amount of trades as the head.

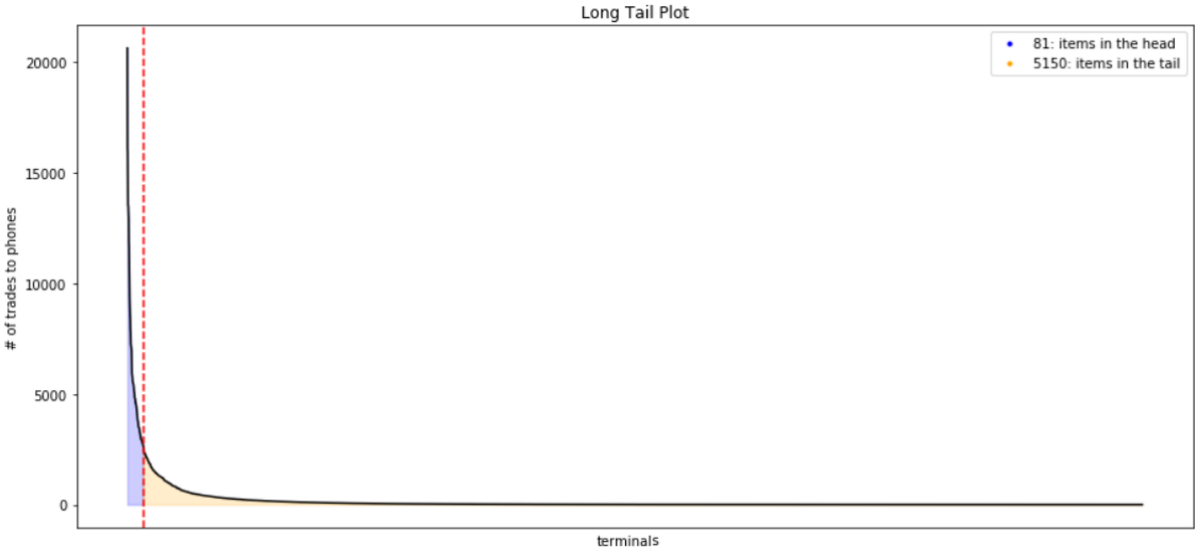


Figure 3.3: Long tail plot for trade information data

In Figure 3.4 the effort needed to have complete coverage on all trades is shown. In order for a trade to be covered, both the previous and new handset need to be available. We can see that to cover 50% of trades we would only need data on 290 handsets, however, to cover 75% of trades, we would need more than double the amount required at 50% for a total of 632 handsets – $\approx$ 118% increase in effort for a gain in coverage of only half what was previously obtained. These differences in effort get even more extreme the further we go down the goal of reaching full coverage. For example, in order to reach 99% coverage, 2980 devices would be needed and for a increase of only 0.9% in coverage which translates in an increased effort of approximately 74%.

3.2.2 User data

At the Telco, a strong effort is made to protect user information and as such, identifiable information such as names, age and location are not available. In total, 26 different variables are described below and split into five different categories. The user identifier field has been anonymized in order to protect the users' identities.

Mobile traffic usage:

- data_dl: Amount, in bytes, of total data downloaded each month by the user.
- data_up: Amount, in bytes, of total data uploaded each month by the user.
- data_vol: Total amount, in bytes, the user has consumed each month via downloads or uploads.

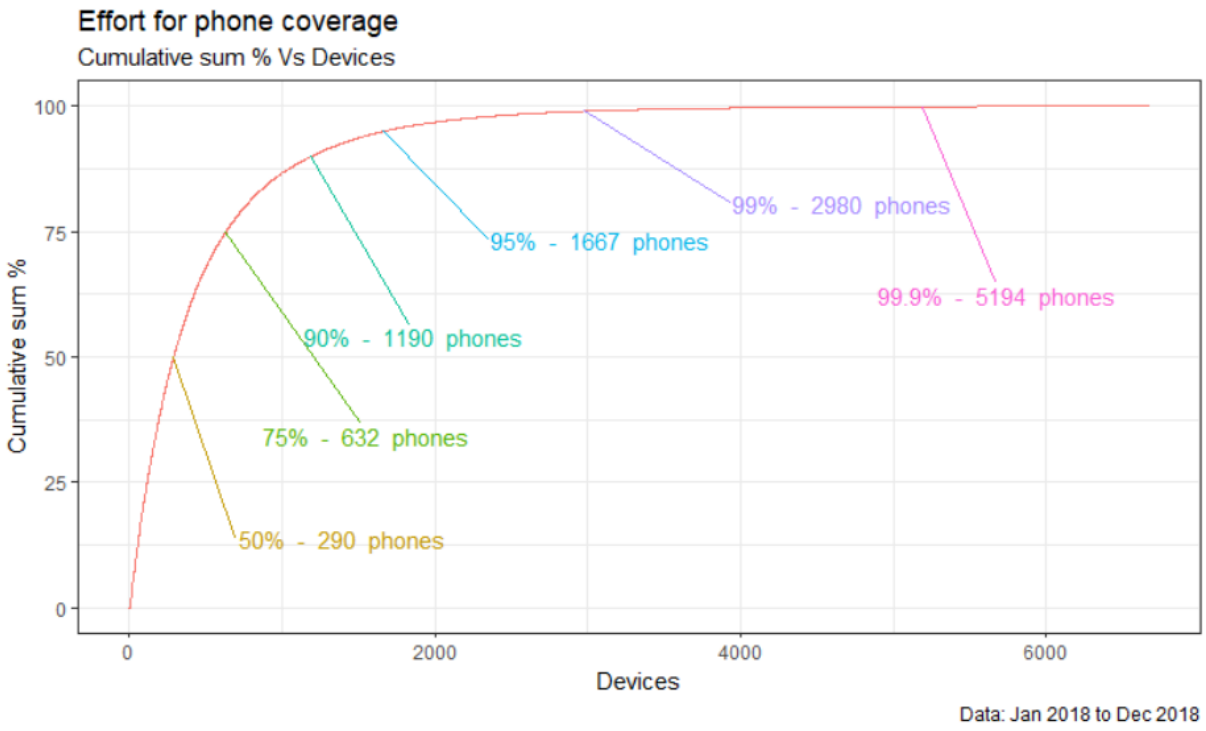


Figure 3.4: Effort for phone coverage considering source and destination phone

- data_gprs: Total amount, in bytes, the user has consumed via GPRS (general packet radio services) standard, each month via downloads or uploads.
- data_umts: Total amount, in bytes, the user has consumed via UMTS (universal mobile telecommunications service) standard, each month via downloads or uploads.
- data_lte: Total amount, in bytes, the user has consumed via LTE (long-term evolution) standard, each month via downloads or uploads.

Airtime and SMS utilization:

- airtime_in: Number of minutes a client has been on call, from incoming calls, each month.
- airtime_out: Number of minutes a client has been on call, from outgoing calls, each month.
- airtime_inter: Number of minutes a client has been on call, from international calls, each month.
- airt_landline: Number of minutes a client has been on call, from landline calls, each month.
- sms_total: Number of SMS (short message service) made by the client, for a specific month.

User's mobile plan:

- rate_plan_id: Identifier giving us information on what mobile plan the client has, in the form of a numeric identifier.
- contract_antiquity: How old the contract for the mobile plan is, in months.
- first_contract_date: Date of first contract made for the mobile plan.

- mobile_cards_corp: Number of corporate mobile cards the client has.
- mobile_cards_pme: Number of small and medium business mobile cards the client has.
- bundle_mobile_net_mb: Mobile data limit, in number of megabytes, available each month to the user.

User landline services:

- price_pack: Price of the pack of the client's home service.
- end_pf_day: For how long much longer, in days, the user has a loyalty contract to the Telco.
- bundle_net_speed: Home wired internet connection speed of the user, in megabits.
- bundle_services_additional: Number of additional services the client has.

User profile data:

- client_antiquity: How long, in months, the user has been a client.
- marketing_flag: Whether the client has activated the no marketing flag.
- channel_visits_web_3m: Number of visits the client has made via web channels, in the past three months.
- channel_visits_total_3m: Number of visits the client has made via any channel, in the past three months.
- user_identifier: Anonymized user identifier.

More detailed characterization of the data will not be done for confidentiality reasons due to the sensitive nature of the data.

3.2.3 Handset data

At the Telco there are two main different ways of identifying a specific brand and model handset: Using the Type Allocation Code (TAC) or the terminal identifier. The TAC corresponds to the initial eight digit portion of the fifteen digit International Mobile Equipment Identity (IMEI) while the terminal identifier is an internal reference to aggregate multiple TACs into a single identifier. The need for grouping multiple TACs comes from its unstable nature and propensity to change over time, meaning that if we were to use the TAC to identify the handsets and, for example, a system upgrade triggered a TAC change, we would see it at as new device. Table 3.4 shows an example for a specific device where for just one model, we have fifteen different TAC variations but only one terminal identifier.

For the existing baseline models, the TAC was being used but due to the reasons mentioned the terminal identifier will instead be used as the way to identify different handsets.

There are two different sources on handset data available at the Telco, both having some limitations. The first, called info_terminals, has only very generic information while the second, called terminal_characteristics, whilst very rich on the variety of attributes available for some devices, is unfortunately too deficient in terms of completeness of the data as will be further explored below.

Table 3.4: TAC to terminal identifier comparison

TERMINAL_ID	TAC	EQUIPMENT_NAME
4608	35155809	GALAXY S8+ SM-G955F
4608	35155909	GALAXY S8+ SM-G955F
4608	35156009	GALAXY S8+ SM-G955F
4608	35156109	GALAXY S8+ SM-G955F
4608	35435808	GALAXY S8+ SM-G955F
4608	35767308	GALAXY S8+ SM-G955F
4608	35781908	GALAXY S8+ SM-G955F
4608	35782008	GALAXY S8+ SM-G955F
4608	35782108	GALAXY S8+ SM-G955F
4608	35782208	GALAXY S8+ SM-G955F
4608	35879208	GALAXY S8+ SM-G955F
4608	35912208	GALAXY S8+ SM-G955F
4608	35912308	GALAXY S8+ SM-G955F
4608	35912408	GALAXY S8+ SM-G955F
4608	35912508	GALAXY S8+ SM-G955F

Another separate source will be used to have information on the prices of handset devices which is represented, for trades from January 2018 to December 2018, in Table 3.5. Pricing data covers 65% of the devices.

Table 3.5: Ratio of null values on pricing (items with no price information) for source and destination handset

	Source handset	Destination handset
Ratio of null values	0.3914664	0.3549961

Exploring the information on info_terminals:

We extract the following data from info_terminals:

- Brand
- Model
- Type
- Release date
- Supported network bands
- Wi-Fi and bluetooth capabilities

- Marketing type

In Figure 3.5 the bar chart illustrates the upgrades by brand in 2018. Samsung and Huawei dominated the upgrades by the Telco customers with the next top-seller being Apple who did not reach half the users of Samsung. We can also see that by summing all phones from all other brands that are not in the top nine, we still do not reach the numbers of Huawei, the second most popular brand. From this analysis, we can reasonably assume that brand plays a relevant role in user choices.

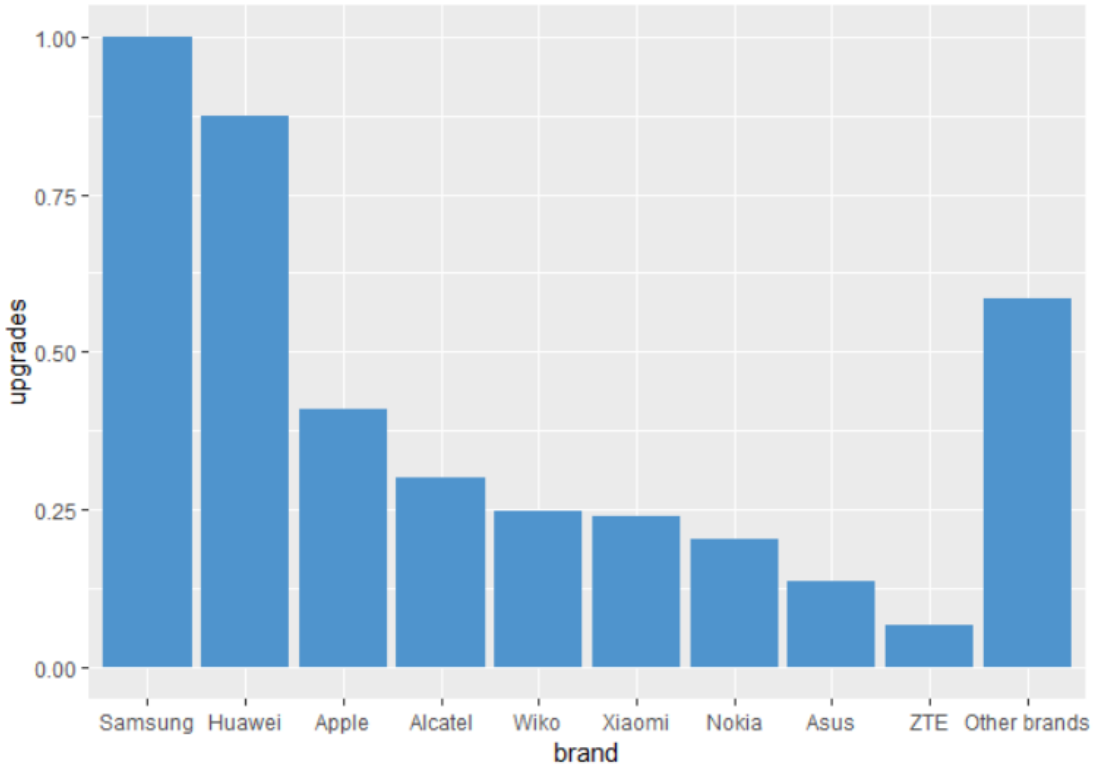


Figure 3.5: Upgrades by brand for 2018, relative scale

Regarding the existing types of device, there are fourteen different types with an absolute majority of handsets falling within the category of either phone or smartphone as summarized in Table 3.6. The top 5 types are represented in relative terms to the existing maximum. For this reason, only phones and smartphones will be considered for the model.

Table 3.6: Relative frequency of the different types of devices

TYPE	Relative frequency
SMARTPHONE	1.0000000
PHONE	0.0866669
TABLET	0.0014139
WIRELESS GATEWAY	0.0001855
PDA	0.0001154

Since the release date of the devices is associated with the TAC, multiple release dates end up existing for the same device (terminal id). Even when taking this aspect into consideration, the values are still not consistent with the real and expected release dates, as the examples in Table 3.7 show. Given this inconsistency, instead of using this particular field for the phone release date, we use the date corresponding to the first time any client had that device on the network as a proxy of the release date.

Table 3.7: Comparison between supposed equipment date and real release date

Handset	Equipment date	Real release date
Iphone 6 (A1586)	May, 2014	September, 2014
Samsung Note 8	February, 2017	August, 2017
Moto G (3rd Generation)	October, 2014	July, 2015

In relation to the marketing type of handset upgrades, the vast majority are labeled as "Superphone 4G", with less than a fourth of the frequency coming the "smartphone" and basic types, as plotted in Figure 3.6.

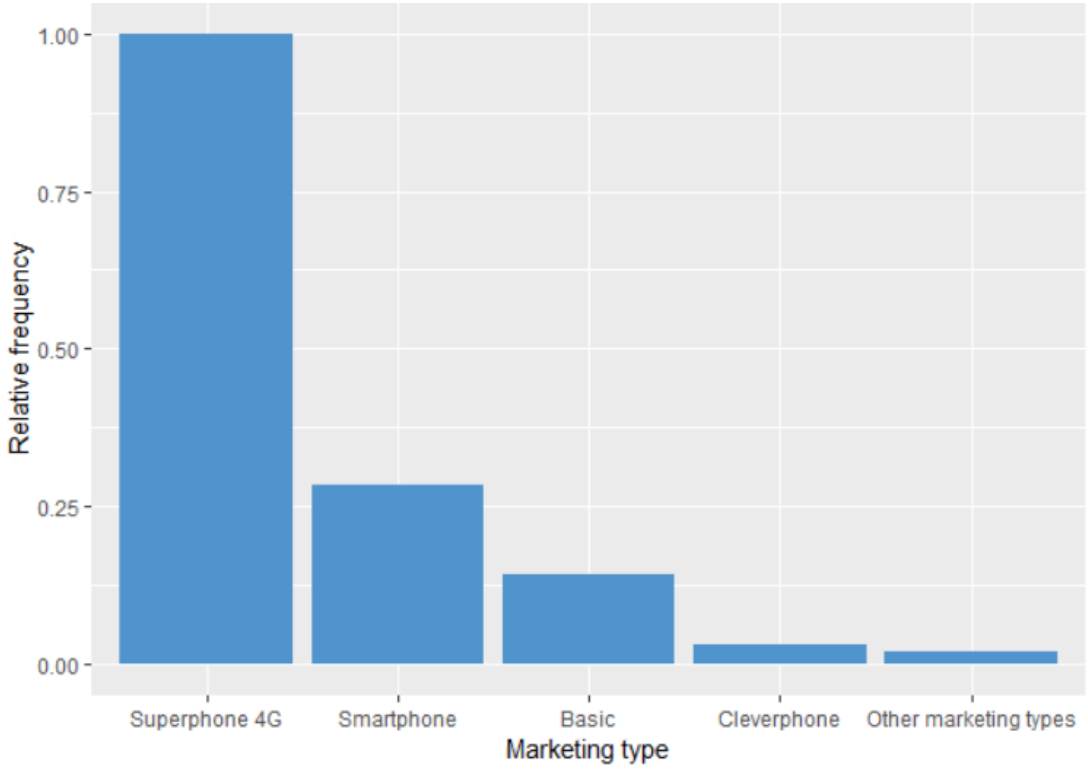


Figure 3.6: Relative frequency of marketing type labels for handsets

Brand, model, support network bands and WiFi and Bluetooth capabilities are a given in the vast majority of handset devices. Since these features have negligible discriminative power, they are not used.

Information available on terminal_characteristics:

In total, there are 358 different features that describe handsets using the data on terminal characteristics. Of those 358, the following were manually identified as being the most important using the Telco's domain knowledge:

- Back and Front camera resolution
- Back and Front camera video resolution
- Battery capacity
- Storage
- RAM
- Screen resolution
- Size in inches
- Weight in grams

These features would be a potentially valuable asset for the content-based handset recommendations if they were complete. As such some of these will be explored below to test for completeness.

In Table 3.8 we can see that there is a very high amount of missing values with the best case being 76% for the size and worst being more than 99% in the case of RAM. These values are too high to be useful for any use case. A possible explanation would be that older devices have more missing values, since they could be more poorly described.

Table 3.8: Relative frequency for some important characteristics of null or NA values

	Characteristics				
	RAM	Storage	Battery	Size	Screen resolution
Relative frequency of null values	0.992028	0.808492	0.979666	0.761018	0.961304

To check if devices released recently have more complete data, in Table 3.9, we show the same relative frequency of missing values but instead considering only devices released since 2016. Surprisingly, these results are considerably worse than what was seen before with the the full catalogue, with the best case now being around 94% compared to the previous worst case of 76%.

Table 3.9: Relative frequency for some important characteristics of null or NA values (handsets released since 2016)

	Characteristics of handsets released since 2016				
	RAM	Storage	Battery	Size	Screen resolution
Relative frequency of null values	0.976318	0.945690	0.962729	0.938112	0.962741

Finally we consider the top devices for each month in 2018, and then remove the devices repeated in between months. We can see that our results have in fact improved a lot with our best case now being the size with only 16% missing values and and the worst case being RAM with a relative frequency for missing values of 68%. Although this is an improvement regarding the proportion of missing values, we are only looking at a very small part of the entire catalogue, which naturally reduces scope.

Table 3.10: Relative frequency for some important characteristics of null or NA values (top 10 handsets of 2018)

	Characteristics of top 10 devices in each month of 2018				
	RAM	Storage	Battery	Size	Screen resolution
Relative frequency of null values or NA	0.580645	0.451612	0.290322	0.1612903	0.258064

Aditional data sources for handset information:

To counter the limitations of the previous datasets, web-scrapping [Malik and Rizvi, 2011] was performed on free and public resources available on the web with information being gathered on some of the most popular devices. We collected data on 285 different devices, which corresponds to 51% of the total trades made from January 2018 to December 2018, considering that a trade is only covered if data is available on both the user's previous and new handset.

As a result, the following features were made available: AnTuTu[2] score, which is an application used to benchmark a handset's performance, screen aspect ratio, battery capacity in mAh, CPU core count and frequency, height, width and thickness, weight, number of SIM cards, operating system, RAM, rear and front camera resolutions, screen resolution and density, storage capacity, percentage of usable surface and whether it has an audio jack, removable battery, SD slot, fast charging or a fingerprint sensor.

The relative frequency for the storage, RAM and SIM card attributes are listed in Table 3.11. We can see that most of the devices have between 4 to 32 GB of storage and that the most typical values for RAM are either 512 MB, 1GB, 2GB or 3GB. In regards to the numbers of SIM cards available on the handset, 57% of them are dual-SIM while 43% are single-SIM.

In Figure 3.7 and Figure 3.8 it is shown the density plot of the screen and camera resolution respectively.

Operating system can vary between devices as can its version. For this reason, this attribute was changed to show the versions as an incremental number, for every operating system. For example, if we had in our dataset only version 2.0 and 4.1 of android, they would be converted to 1 and 2 respectively. Figure 3.9 shows the result of this processing and by analyzing it, we can say that Android is the most fragmented system due to having the most different versions and it is also seen that more recent versions tend to have more presence than older versions.

In Figure 3.10 we show all device features that can either be present or absent, such as the the presence of an audio jack, removable battery, SD slot, fast charging and fingerprint sensor.

[2]https://www.antutu.com/en/

Table 3.11: Relative frequency for storage, RAM and number of sim cards

Storage (GB)		RAM (GB)		Number of sims	
0.278	1.40	0.278	1.40	1	0.43
1	0.35	0.347	0.35	2	0.57
4	22.11	0.512	17.19		
8	28.07	0.768	4.21		
16	30.18	1	32.28		
32	12.63	1.5	5.61		
64	3.51	2	18.95		
128	1.75	3	12.98		
		4	4.91		
		6	2.11		

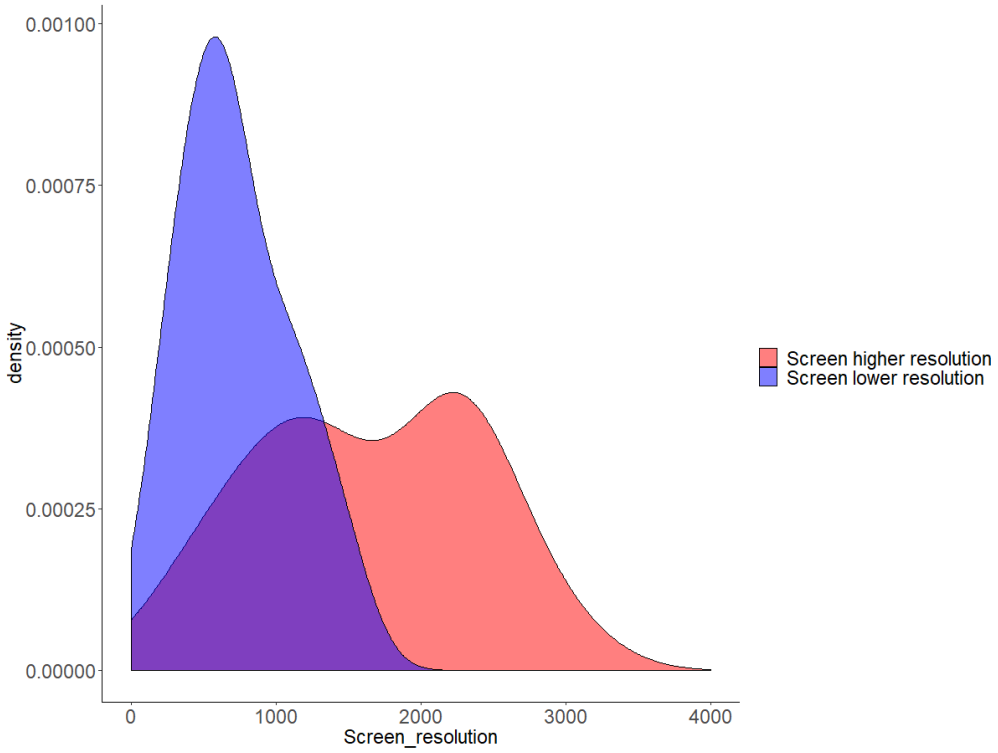


Figure 3.7: Screen lower and higher resolution density plot

3.3 Baseline algorithms

There are two distinct models used at the Telco with different goals in regards to handset recommendation: the first is applied to predict the probability of a user being in a situation of handset upgrade while the other is related to determining which device to recommend to the users. The latter is not dependent on the first with recommendations being possible for every user. Only the second model, the item recommendation model, will be detailed here.

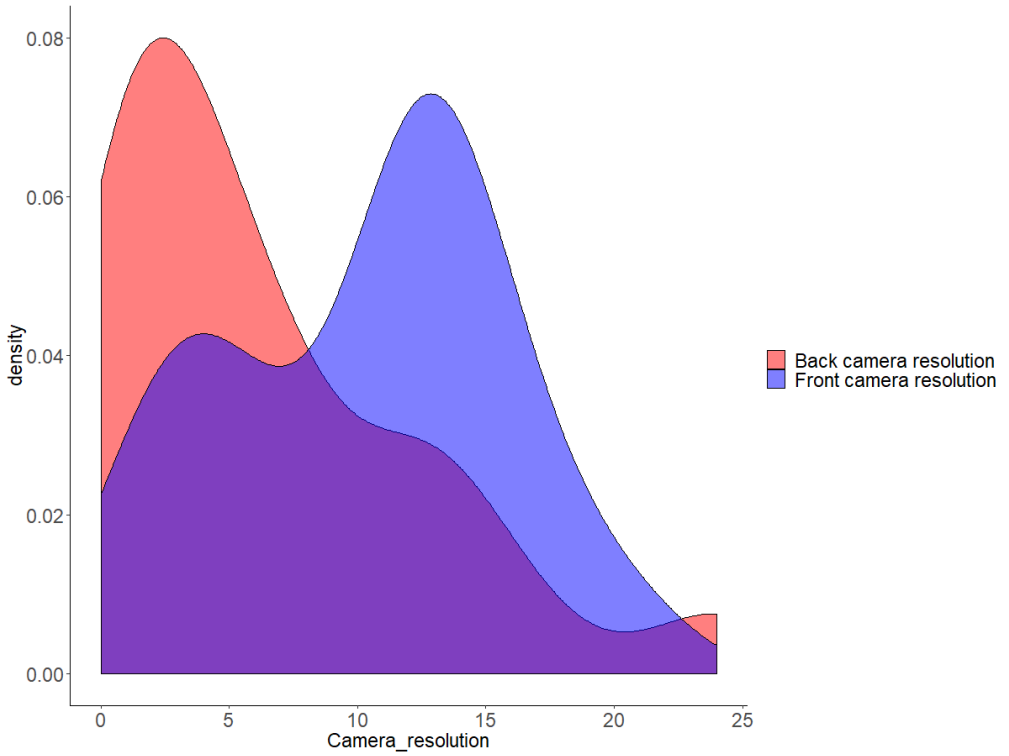


Figure 3.8: Front and back camera resolutions density plot

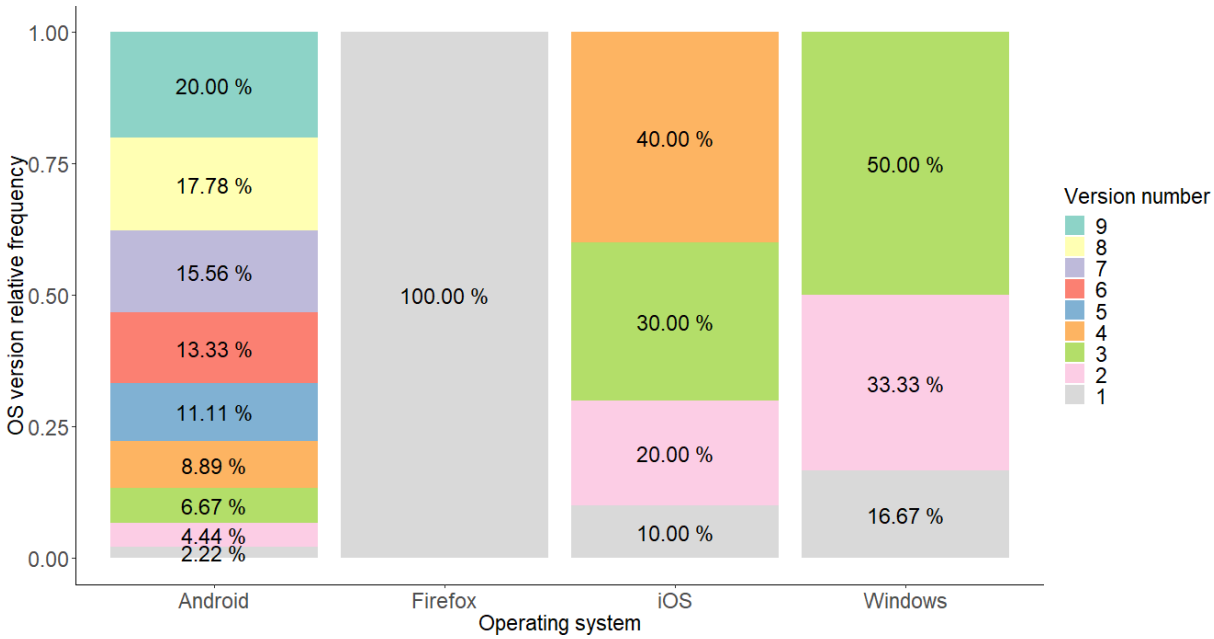


Figure 3.9: Relative frequency of version by operating system

There are three existing probabilistic models currently being used at the Telco, with the last one being an ensemble of the first two. All three methods will be detailed below, but only the first two will serve as baseline for future methods. The reason for the latter not being considered is that it is an ensemble method and there are very small improvements in using such method, as can be seen on the results available in Table 3.12, which will be shown below.

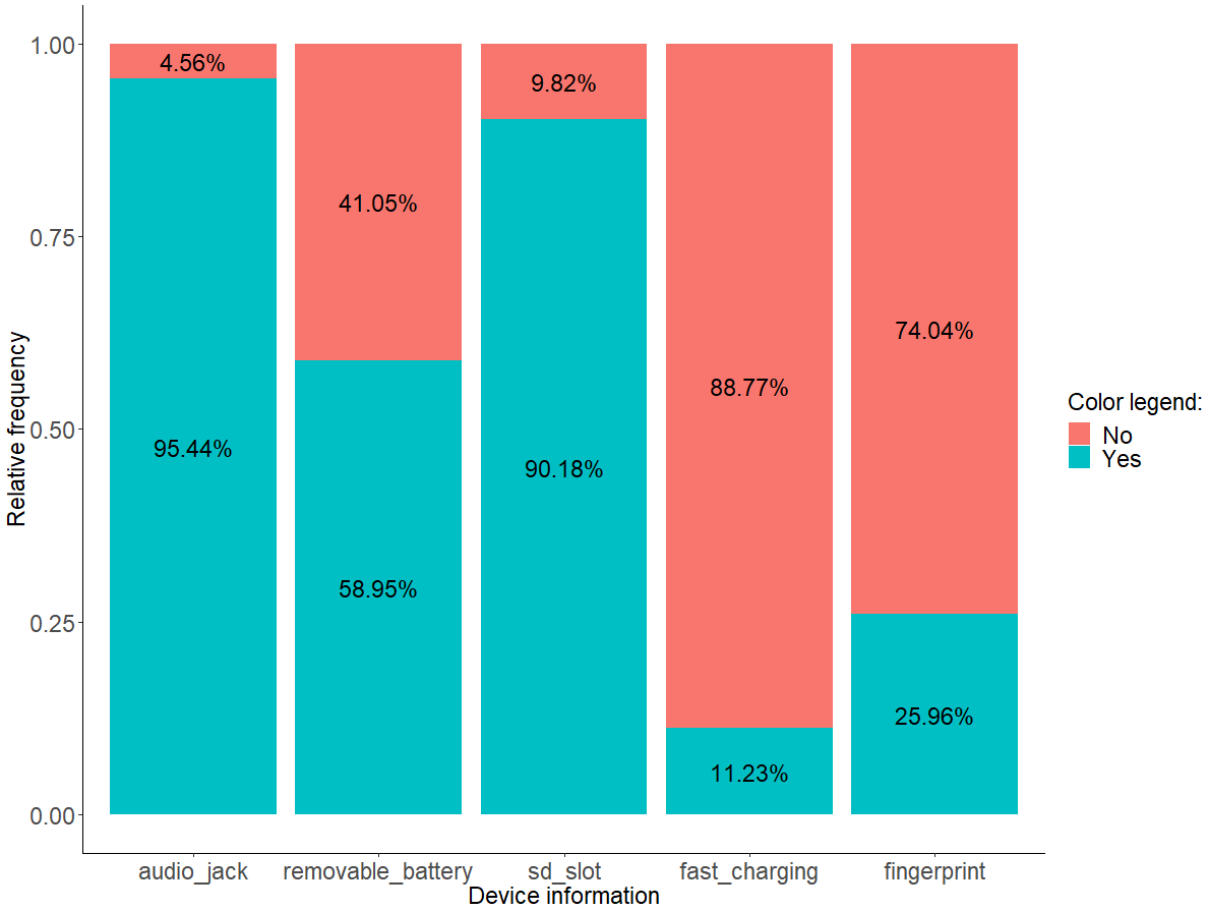


Figure 3.10: Relative frequency by characteristic

All the solutions that will serve as baseline are based solely on the handsets popularity with rating information being of the unary kind where a rating of one is given to a user-item pair if the user had used that device in the past and zero otherwise. The available models are then the following:

1. Global Probability
2. Specific Probability
3. Weighted Probability

The global probability is the probability of a random user swapping to handset B given any other handset. This probability simply corresponds to the probability of B:

$$p_{global} = P(B) \tag{3.1}$$

The specific probability is the probability of a random user swapping to handset B given he currently has handset A. This probability is defined as:

$$p_{specific} = P(B|A) = \frac{P(B \cap A)}{P(A)} \tag{3.2}$$

The weighted probability is the weighted combination of the two previous probabilities. For this probability the user needs to define the minimum number of trades, U, between the pair origin handset and destiny handset. This probability is calculated in the following manner, where M represents the number of trades with a specific handset as origin:

$$p_{weighted} = wod \times p_{global} + (1 - wod) \times p_{specific} \tag{3.3}$$

where:

$$wod = \frac{(T - M)}{T} = 1 - \frac{M}{T}$$
$$T = max(U, M)$$

At the Telco, a criterion called the definitive trade criterion exists, where, in order for a given trade to be considered a real trade, the user had to keep the same phone for the month in which the first trade had occurred and another two extra ones. This results in the data having an extra delay of two months but would correct cases where for example, a trade only occurred due to temporary replacement of the previous handset. The baseline models used both these criteria and considered trades using the TAC instead of the terminal id. The main differences between the TAC and the terminal id is that the TAC can change over time for the exact same device while the terminal id groups these ids into a unique handset identifier.

Existing evaluation metrics were available for these models which used the historic data of customers and their phones from the months of January 2018 to May 2018 as training data and June 2018 as testing data. The results obtained can be found in the Table 3.12.

Table 3.12: Results using probabilistic model

	$Pspecific$	$Pglobal$	$Pweighted$
HR@1 (Hit rate at 1)	0.061	0.038	0.061
HR@5 (Hit rate at 5)	0.180	0.130	0.184

3.4 Collaborative-filtering approach

With the collaborative filtering approach, the focus was put on testing a large variety of different algorithms while minimizing the number of comparisons between different libraries. The reason for using a reduced number of different libraries is to avoid differences in implementations and evaluations as, with the same evaluation metric and algorithm, it is possible to reach orders of magnitudes of difference for an equal experiment, as explored in [Said and Bellogín, 2014].

Two approaches were taken:

- Implementing the algorithms from scratch, to have better control on our experiments and use whatever metric we would be interested in. We call this our custom implementation.

- Using the Librec library [Guo et al., 2015], that supports a large amount of state-of-the-art CF algorithms and evaluation metrics.

In the following sub-sections, these approaches will be explored using the trades from January 2018 to May 2018 as train and June 2018 as test. These months were chosen as, due to the amount of algorithms tested and the available computational power, and also due to the baseline (Section 3.3) testing already done by the company using the same months. The definitive trade criterion considered for this testing was to only consider trades to devices more recent or released as the same time. This criterion is different from the previously existing one, as justified in Subsection 3.2.1.

3.4.1 Collaborative filtering custom implementation

Since both the global and specific probability methods were already implemented, the methods chosen to be implemented were the following:

- Item-item using cosine similarity
- Item-item using cosine similarity, adjusted with time
- Unnormed cosine
- Rarity

User-based methods were not used due to the high number of users. With item-item, the goal is to find similarities between pairs of items using a similarity measure, which in this case was the cosine similarity (3.4).

$$cosinesimilarity(A,B) = \cos(\theta) = \frac{A.B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{3.4}$$

As the cosine similarity is not directional, the similarity from phone B to phone A would be the same as A to B. It is important to make sure we have an asymmetric similarity measure as we are only interested in recommending upgrades, and not in finding similar items or downgrades. To give directionality to our similarities, time was added in a way such that the positive similarities would correspond to similarities from a older to a newer device and negative similarities for a newer to older device. This choice makes the assumption that clients trade their handsets for more recent ones. This assumption goes accordingly to the domain knowledge the Telco has on the issue.

Since with cosine similarity the values vary from 0 to 1, it is important to note that our new similarity can no longer be considered a cosine similarity but will be called *cosine time adjusted* (3.5) for simplicity sake.

$$cosine\_time\_adjusted(A,B) = \begin{cases} cosinesimilarity(A,B) & \text{A older than B} \\ -cosinesimilarity(A,B) & \text{B older than A} \end{cases} \tag{3.5}$$

Rarity (3.6) was tested to verify if the results would end up being the same as with the specific probability (3.2).

$$\begin{aligned} rarity(A,B) &= \frac{P(B|A)}{P(B|\neg A)} = \frac{P(B \cap A)}{P(A)} \div \frac{P(B \cap \neg A)}{1 - P(A)} \\ &= \frac{(1 - P(A))}{P(A)} \frac{P(B \cap A)}{P(B \cap \neg A)} \Rightarrow (constant) \frac{P(B \cap A)}{P(B \cap \neg A)} \end{aligned} \tag{3.6}$$

The unnormed cosine (3.7) corresponds to the typical cosine similarity (3.4) but without any normalization. The goal of this test was to check if the normalization used with the cosine was responsible for the lackluster results (table 3.13) with typical cosine similarity.

$$unnormedcosinesimilarity(A,B) = A.B = \sum_{i=1}^{n} A_i B_i \tag{3.7}$$

User-based CF has not been implemented due to low amount of trades made by each user and due to the large amount of users compared to items, which would not only make it much more resource intensive, but also give worse results. This was verified using both the librec and recommenderlab [Hahsler, 2015] libraries.

Table 3.13 shows the HR@1 and HR@5 for both the existing popularity methods and the methods described above. The expected probability model still holds the best results closely followed by the specific probability and rarity and then by the unnormed cosine. The item-item performance is lower than expected even with the time adjustment but the unnormed cosine yields better results. The rarity method has equal results to the specific probability.

From these results, we see some evidence suggesting that a bias towards popular items exists, supposition that will be further explored in the next two following chapters.

Table 3.13: Collaborative filtering custom implementation - results

Model	HR@1	HR@5
Most popular (global)	0.039	0.128
Most popular (specific)	0.067	0.196
Most popular (expected)	0.067	0.198
Item-item	0.030	0.090
Item-item (time-adjusted)	0.039	0.122
Rarity	0.067	0.196
Unnormed cosine	0.055	0.167
Unnormed cosine (time-adjusted)	0.067	0.195

3.4.2 Collaborative filtering using librec

While, despite multiple versions of librec being available, version 2.0 was chosen due to it being the one which most of the available documentation was made for and the one librec official website suggests. Multiple libraries were explored, but none was found that could match the number and variety of algorithms available as well as recommendation metrics.

All learning-to-rank algorithms available in librec 2.0 were tested, with simple hyperparameter tuning being done empirically. The following were the methods used, their simple description according to the librec library and their respective hyperparameters, with the most popular and random guess methods not shown as they have no hyperparameters to tune.

- aobpr [Rendle and Freudenthaler, 2014], an adaptation of bayesian personalized ranking with adaptive oversampling.
- aspectmodelranking [Hofmann and Puzicha, 1999], probabilistic latent space model for collaborative filtering.
- bpoissmf [Gopalan et al., 2015], hierarchial Poisson matrix factorization.
- bpr [Rendle et al., 2009], bayesian personalized ranking.
- climf [Shi et al., 2012], collaborative less-is-more filtering where the mean reciprocal rank is optimized.
- eals [He et al., 2016], an efficient implementation of the alternating least squares algorithm.
- fismauc Kabbur et al. [2013], factorization item similarity model with focus on data sparsity.
- itembigram Wallach [2006], model that incorporates n-gram statisticss and latent topic variables.
- itemknn, k-nearest neighbours model.
- lda [Blei et al., 2003], latent dirichlet allocation for implicit feedback, where users are documents and items are words.
- listrankmf [Shi et al., 2010], list-wise matrix factorization.
- plsa [Hofmann, 2004], a latent semantic model.
- rankals [Takács and Tikk, 2012], a latent factor model based on alternating least squares.
- ranksgd [Jahrer and Töscher, 2011], a latent factor model based on stochastic gradient descent.
- slim [Ning and Karypis, 2011], sparse linear method for top-N recommendation.

By analyzing the Figure 3.11, containing the average precision at cutoff 10 for the tested algorithms, we can see that none of the results are satisfying with listrankmf, lda, mostpopular and fismauc having the better results and bpoissmf, climf and eals having the worst. Although, there is a clear difference in performance between the bottom and top scorers, the results are still low.

In Figure 3.12 we now have the area under the curve represented for the tested algorithms. Again, fismauc, lda, mostpopular and listrankmf are top scorers with the bottom scorers also being bpoissmf, climf and eals. None of the methods passed 0.6 for the AUC with the bottom scorers getting about 0.5 AUC (random).

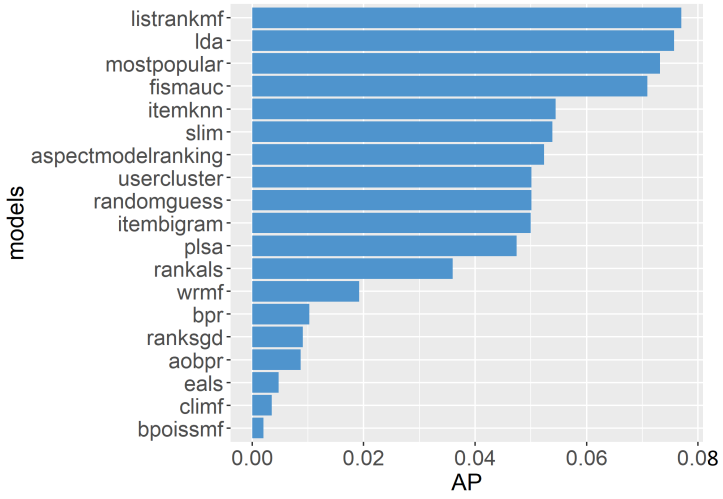


Figure 3.11: Average Precision at 10

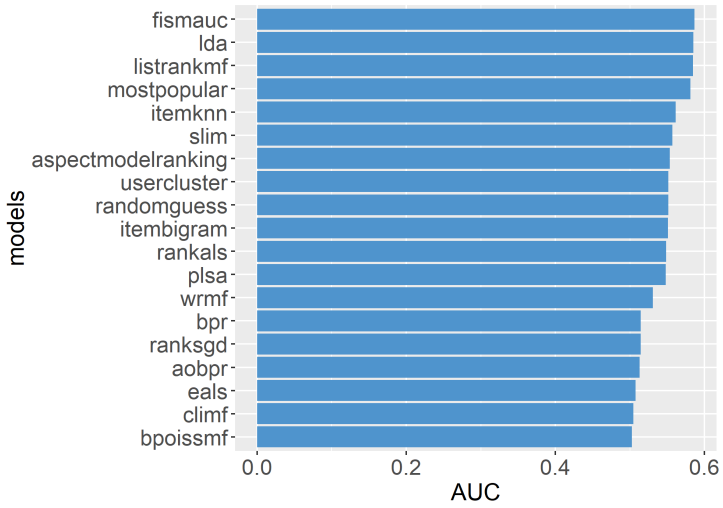


Figure 3.12: Area under ROC curve at 10

The normalized discounted cumulative gain (NDCG) for the algorithms is shown in Figure 3.13 and here again the top performing algorithms are listrankmf, lda, fismauc and mostpopular with bpoissmf, climf and eals being the worse performing one. The worst performing algorithm, bpoissmf, had an NDCG very close to 0 and best performing had a NDCG of close to 0.1.

All the results obtained using the precision at 10 metric, as seen on 3.14, have their precision under 0.02 with the top scorers being fismauc, lda, listrankmf and mostpopular and the least scorers bpoissmf, climf and eals.

Recall results are similar, as can be observed in 3.15, with the better algorithms being fismauc, lda, listrankmf and mostpopular and the worse ones being bpoissmf, climf and eals. The top scorer algorithm saw a recall of 0.17 and bottom scorer algorithm a recall of less than 0.01. There is a very clear difference between the bottom and top scorers.

Finally, for the reciprocal rank at 10, the top scorers were listrankmf, lda, mostpopular and fismauc with

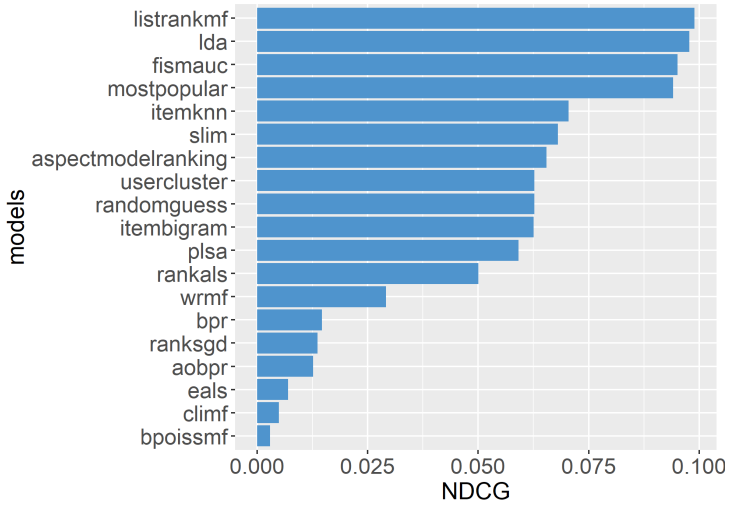


Figure 3.13: Normalized discounted cumulative gain at 10

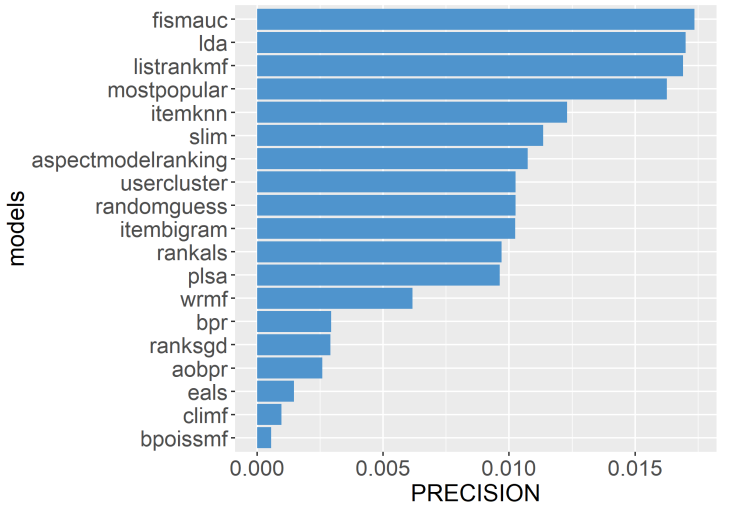


Figure 3.14: Precision at 10

the bottom scorers being bpoissmf, climf and eals. The results varied from close to 0 to around 0.08, which is a substantial difference.

The predictions of one of the better performers in our test, listrankmf, had its performance calculated, and the results were approximately 5% worse in terms of HR@1 and HR@5, making the probability model unbeaten for now.

Drawing conclusions from the results obtained above, we still have not found an algorithm capable of beating the existing popularity methods, with all tested algorithms having similarly equal bad performances, further suggesting that the data is highly biased data towards the popular items.

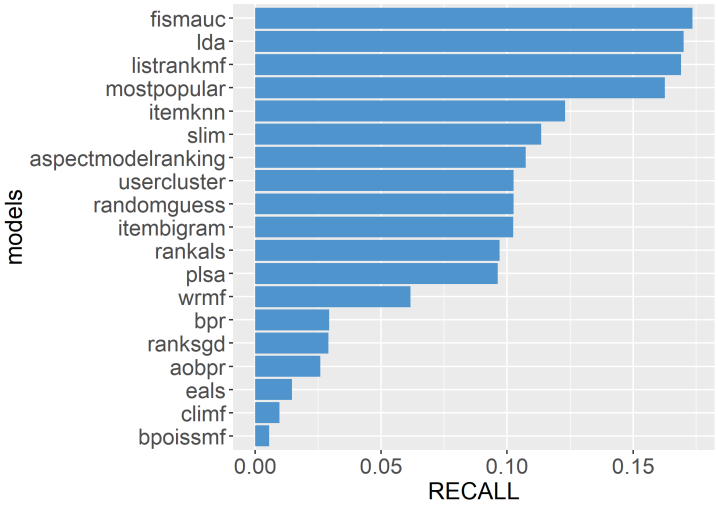


Figure 3.15: Recall at 10

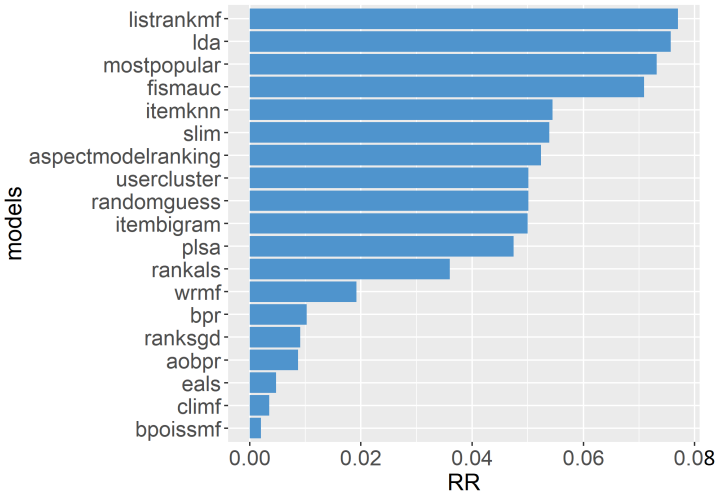


Figure 3.16: Reciprocal rank at 10

3.4.3 Collaborative filtering conclusion

As explored in the sections above, both with the custom implementations 3.4.1 and with the librec library 3.4.2, it becomes obvious that by just using CF data, substantial gains on the existing probability models will not be gained. Even though hyperparameter tuning was not fully explored, the improvements gained would not be high enough to justify the added effort and, with the high variety of CF algorithms tested, the data was extensively explored enough to give us enough of a guarantee on this.

We believe the reason for such poor performance with the tested methods to be due to the extreme sparsity by user and item. For the problem of handset recommendation, we typically only have a very low amount of interaction per user, mostly one, as can be seen in table 3.3, while these methods typically require more to be viable.

One approach that was not explored but that would very likely improve the results would be the combination

of multiple of the existing weak models to try and achieve better results as a whole group than individually, with techniques such as stacking being a possibility of tackling the problem. The reason for not testing these types of approaches was due to the CF space being limited in its nature as it only looks at the trade history for a source of information. If we had more data than simply which user traded to which item, this approach would become more viable, which is what will be explored in the following chapters.

3.5 Content-based approach

A model based on Gradient Boosting with Decision Trees was used for the content-based approach. We describe the algorithm in Subsection 2.2. The modeling approach is described in Subsection 3.5.1. An exploratory study is performed to define criteria for the model. After the method and its criteria have been defined, results will then be provided in Subsection 3.5.2 for a variety of tests using data from 2018 and comparing it with the baselines described in Section 3.3. To finish, Subsection 3.5.3 will serve as a conclusion on the content-based approach.

Since for this approach we will be dependent on having data on the handset devices, we will further filter our dataset to only consider trades involving handsets with available data.

3.5.1 Modeling approach

In the content-based approach, we wish to predict the probability of any user switching from its current handset A to every other handset. For each handset B other than the user's, the desired outcome is the probability of a trade from A to B to occur. (3.8).

$$f :: user \times current\ handset \times target\ handset \Rightarrow purchase\ (yes/no) \tag{3.8}$$

After obtaining the probabilities for the users and target handsets of our choosing, we can then sort the probabilities by user, and obtain a ranked list per user, which would equate to the user's recommendation list.

In Table 3.14 a dummy example on how the approach would work is presented for the users John and Mary, with brands being referred to by either X or Y and models by the numbers following the brand. John has a X 6 and Mary has a Y 8, and we are trying to predict a recommendation list containing three devices: X 7, Y 10 and X 11. By application of the algorithm, we get the probabilities on both users switching to each one of the three target handsets. After sorting these probabilities, we can see that, John, who had an X 6, is more likely to get an X 7 (0.7 probability compared to 0.6 and 0.2) while Mary, who had a Y 8, is morel likely to trade to a Y 10 (0.8 probability compared to 0.4 and 0.3).

To calculate the probabilities mentioned before, a gradient boosting tree algorithm (Subsection 2.2) with the xgboost library was used along with the historical trade information described in Subsection 3.2.1, that serves as a source of positive feedback. Another solid choice for an algorithm would be to use deep neural networks (Subsection 2.1) with similar works [Volkovs et al., 2017] also showing a preference on the GBT algorithm, due to being significantly easier to train.

Table 3.14: Dummy example detailing modeling approach taken for the content-based model

User	Source handset	Target handset	Probability	Recommendations
John	X 6	X 7	0.7	X 7
John	X 6	Y 10	0.2	X 11
John	X 6	X 11	0.6	Y 10
Mary	Y 8	X 7	0.3	Y 10
Mary	Y 8	Y 10	0.8	X 11
Mary	Y 8	X 11	0.4	X 7

XGBoost XGBoost [Chen and Guestrin, 2016] or Extreme Gradient Boosting is a software library based on Gradient Boosting that uses decision trees as base learner. It is optimized for speed and implemented with parallel code, and has been successfully used in many competitions such as the Higgs Boson competition on kaggle [Adam-Bourdarios et al., 2014]. XGBoost can deal by default with missing values without need for manual data imputation, provides L1 and L2 regularization [Xu et al., 2010] and computes second-order gradients of the loss function which helps better locate the minima of the function.

XGBoost has a wide range of hyperparameters that need to be understood to get good performing and fast models and the most commonly used are as follows:

- η: Step size shrinkage used in update to prevents overfitting. Also refered to as learning rate.
- max_depth: Maximum height or depth value for a tree.
- subsample: Subsample ratio to be used on the training data to prevent overfitting.
- colsample_by*: Subsample ratio of columns to be considered. Can be applied to tree construction (colsample_bytree), at each new depth level (colsample_bylevel) or at each split (colsample_bynode).
- γ: Minimum loss reduction needed for further partition on a leaf node of a tree.
- α: L1 regularization term.
- λ: L2 regularization term.

However, an important detail is still missing for our approach to work. Since we only have positive feedback information, the model will always output positive predictions in regards to whether the user will trade or not to that device, as the data only contains unary information.

To address this problem of only having data on positive instances – the actual trades –, we use negative sampling, where, for each positive and real trade, we artificially add one or more negative trades – a synthetic trade. To generate a negative example, we replicate the user and source handset information remaining from the positive one, but switch the destination handset to another different one. Figure 3.17 shows an example of this procedure, where a certain user switched from the blue to the purple handset and a dummy trade was added with negative value for switching from blue to orange. Regarding the choice on the sampling criterion, the questions are how many dummy trades should be added and how we sample the "negative" device.

The following tests were made regarding the sampling of devices for dummy trades:

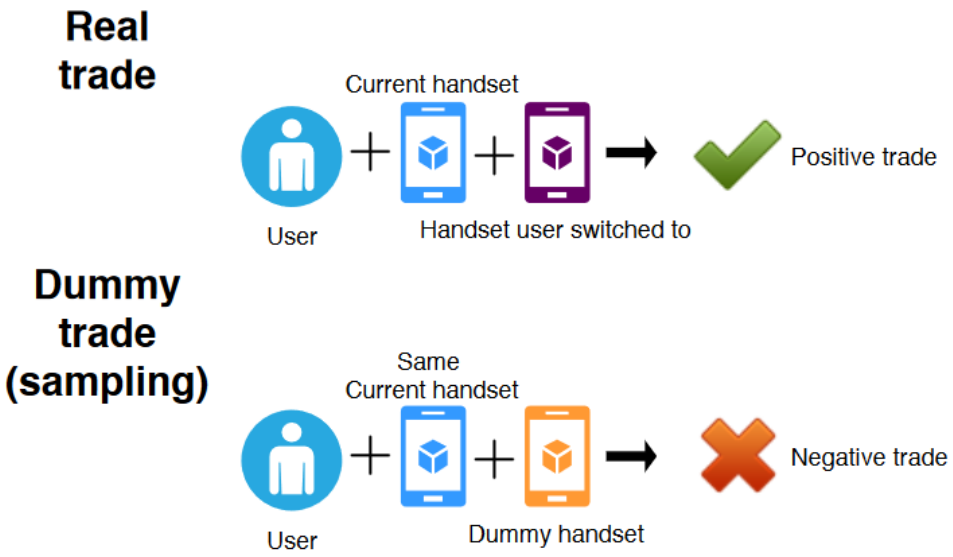


Figure 3.17: Real and dummy trade

- Random sampling: Picking a handset at random.
  - Using only one dummy;
  - Using two dummies;
  - Using five dummies;
- Most popular sampling: Picking a handset taking into account the popularity of devices:
  - Using only one dummy;
  - Using two dummies;
  - Using five dummies;
- Least popular sampling: Same as the most popular sampling but for the least popular.
  - Using only one dummy;
  - Using two dummies;
  - Using five dummies;

The results obtained are as can be seen in Figure 3.18 and 3.19, showing the scores for the HR@1 and HR@5 respectively. The blue in the graph corresponds to the random sampling method while the green corresponds to the most popular and red to the least. Five runs were executed for each sampling approach since we now have randomness in our data with straight lines representing one run, dashes representing two and dotted representing five runs.

As can be seen regarding the HR@1 metric, the worst performing criteria is the most popular, followed by the least popular and lastly by the random. In regards to the number of dummies used, five dummies scored consistently worse than one and two dummies, with the best performer, the random, having better performance with only one dummy.

In the case of HR@5, the gap between the most popular and the others widen, with the random and least popular being very even now. In regards to the number of dummies to be used, in the cases of the

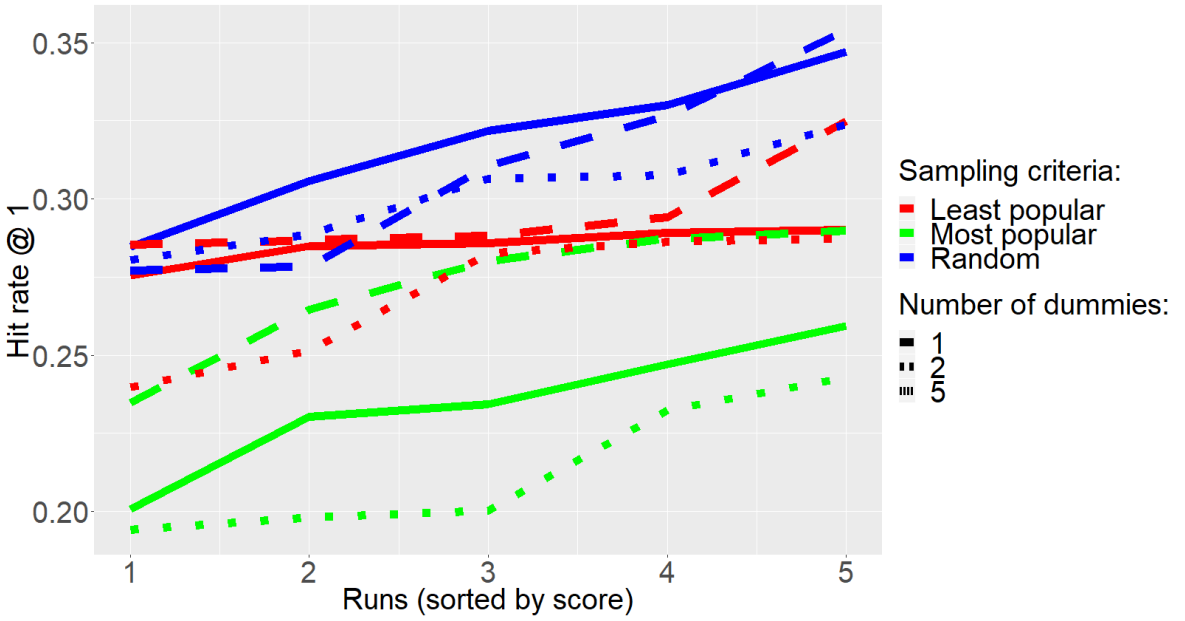


Figure 3.18: Hit rate @ 1 results by sampling type and dummy numbers

random and least popular criteria, the better number of dummies is either one or two, with a slight lead for two.

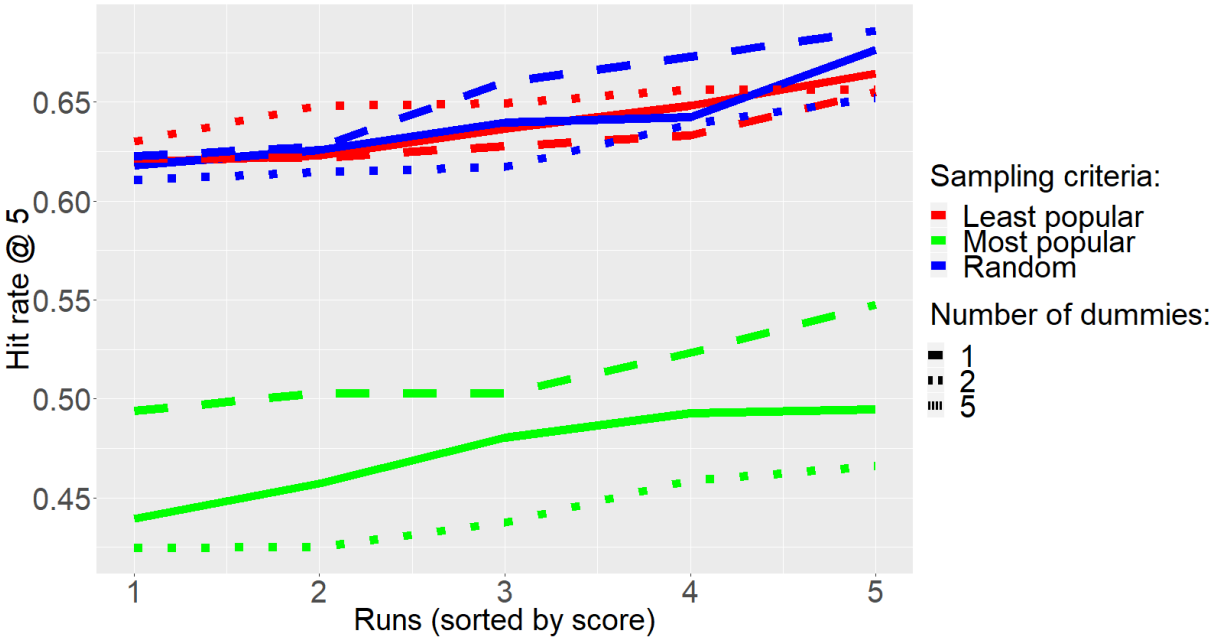


Figure 3.19: Hit rate @ 5 results by sampling type and dummy numbers

Due to the results detailed before, the following criteria will be opted in regards to the sampling: Use one dummy via random sampling. As such we now have a perfectly balanced dataset, where for each positive trade we have a negative one.

Decision trees are not able to encode interactions in a compact way. Here, we introduce similarity and upgrade information. In addition to the current and proposed handset features to be used as a data source, the addition of deltas will be made, both in relative and absolute terms from the proposed to the origin handset to better capture upgrade information such as, for example, in the case of a handset as source with one gigabyte of ram and another destination one with three gigabyte, we would have an absolute and relative delta of two and three respectively.

3.5.2 Results

Before experiments with the content-based model and the comparisons with the existing baselines, a grid-search was performed in order to tune the hyperparameters used by XGBoost. In Table 3.15, we show the values tested for the hyperparameters, and the best value found for each.

Table 3.15: Grid search hyperparameters chosen and best values

Hyperparameter	Values considered	Best value
max_depth	3, 5, 8, 10	5
eta	0.01, 0.1, 1	0.1
lambda	0.01, 0.1, 1	1
colsample_by_node	0.5, 0.75, 1	0.5

Regarding the data used in this set of experiments, only trades in 2018, starting from a known to a known handset device are considered. At least one, and up to eleven months being used as train, while for testing, only one month being used, the following different testing approaches will be performed:

- Growing window using shifting test month - Test where the training size will increase over time and the test month shifts. First test uses January and February as training and test respectively with the latter test using all the months from January to November as train and December as test.
- Growing window using a fixed test month - Test where for all runs the test month is fixed as December. First test will use November as train and the latter November to January as train.
- Sliding window - Test using one month for training and testing. First test will use January as train and February as test. Last test will use November as train and December as test.

Growing window using shifting test month

We use a growing window where the test month is shifting. Firstly, training is performed with data from the month of January, and testing with data from February. Then, we train again with January and February, and use March to test. The test will end up with December being used as test, and all months from January to November being used for training.

In Figure 3.20 we plot the HR@1 metric for the baseline models, as well as the average of five runs of the content-based model. The shadowed region illustrates the difference between the best and worst runs. As can be seen, the content-based algorithm always provides substantially better results, never lower

than twice the HR@1 of any of the baselines. It can also be seen, that the content-based model is less stable, contrary to the baselines which provide very similar results across the timeline.

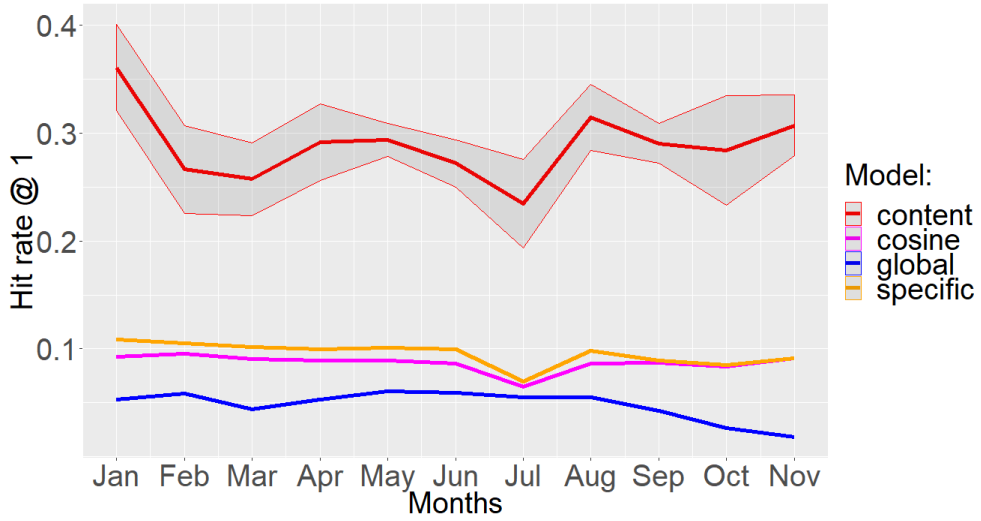


Figure 3.20: Growing window with shifting test month - HR@1 for all models

Figure 3.21 shows the HR@5 for the same experiment. Again, the content-based model clearly outperforms the baselines.

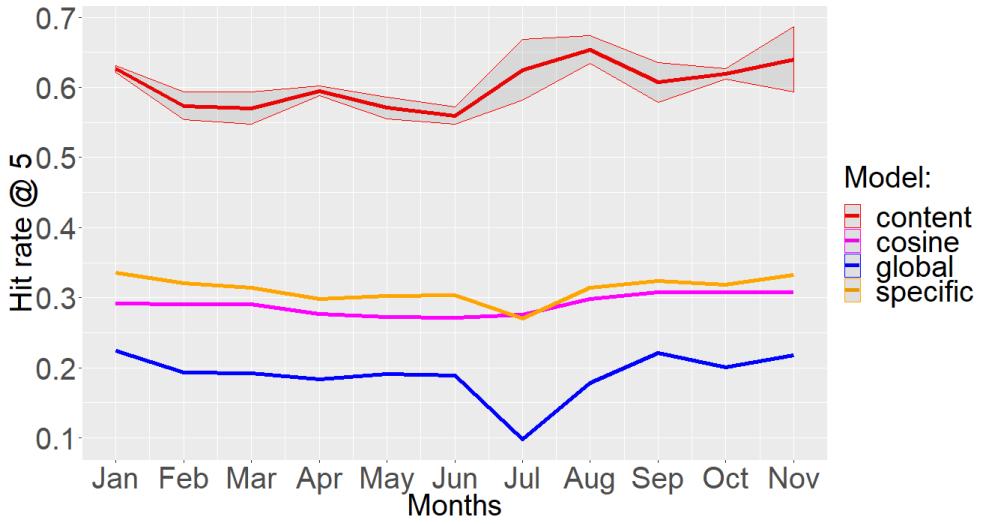


Figure 3.21: Growing window with shifting test month - HR@5 for all models

Table 3.16 summarizes the previous analysis by showing the average result for all the months considered. By considering the averages, the best performing baseline in terms of HR@1 was the specific probability with 0.1. The content-based model achieves a HR@1 of 0.29. With HR@5 the results are similar5, with the content model yielding 0.60 while the specific probability model only reaches 0.31.

Table 3.16: Growing window with shifting test month: Average HR@1 and HR@5 for all months

Model	HR@1	HR@5
Content-based 5 run average	0.29	0.60
Global probability	0.05	0.19
Specific probability	0.10	0.31
Cosine	0.09	0.29

Growing window using a fixed test month

For this test, we use a growing window with the test month fixed in December. This way, the only thing varying is the quantity of months used for training, which will start off with only November being used, to November and October, finishing with all months being used from January to November for training.

In Figure 3.22, the HR@1 for the CB model is noticeably higher than the baseline approaches. We can also notice that the baseline models tend to lose performance the more months are used for train, especially in the case of the Global Probability model, while the CB model provides relatively similar results across the board.

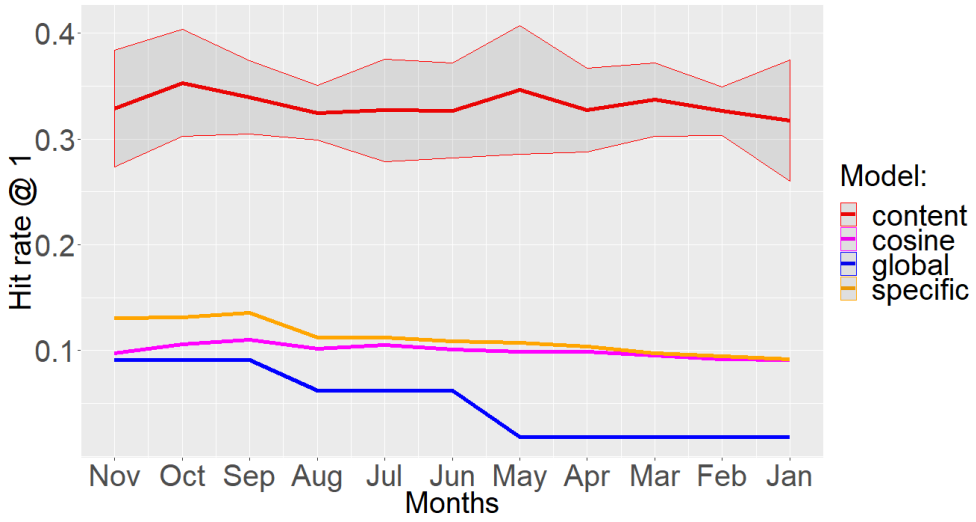


Figure 3.22: Growing window with fixed test month - HR@1 for all models

In the case of HR@5, the CB model presents more varying results across sampling results specially when using fewer months. Nevertheless, it still beats the results from the baselines, that again, tend to degrade as more data is used, especially in the case of the Global Probability model.

Table 3.17 shows the average results across all models for the test using a growing window with fixed test month as December. The HR@1 for the CB model is at least three times better than the best baseline model (Specific Probability) and gets an average HR@5 score of 0.64 compared to the best baseline score of 0.38.

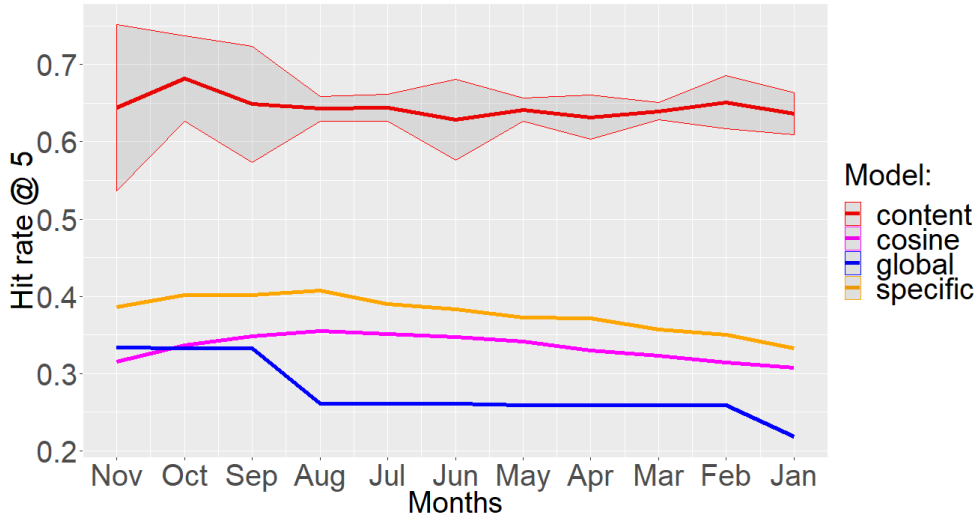


Figure 3.23: Growing window with fixed test month - HR@5 for all models

Table 3.17: Growing window with fixed test month: Average HR@1 and HR@5 for all months

Model	HR@1	HR@5
Content-based 5 run average	0.33	0.64
Global probability	0.05	0.28
Specific probability	0.11	0.38
Cosine	0.10	0.33

Sliding window

The sliding window test uses always one month for both train and test, sliding successively, so that we start off with January being used as train and February as test, and end up with December being used as test and November as train.

Looking at Figure 3.24, where the HR@1 for all the models is displayed, we can see that, again, the CB approach outperforms all other baselines. CB model has a large decrease in performance with data in July, while this decrease was much smaller for the baselines. Again, the baselines provide much more stable results when compared to the CB model. This is especially noticeable in the months of February to March, and very heavily across different months, with HR@1 values ranging from 0.15 to 0.4.

In the case of the HR@5, shown in Figure 3.25, the scenario is somewhat reversed, with the CB model providing very stable results across different sampling runs and for all months, while the baselines had a steep decrease in the months of July.

Summarizing the previous analysis for the sliding window results, Table 3.18 shows the averages HR@1 and HR@5 for all months for all models. In average, we can see that the CB model provides an improvement of three in regards to the best scoring baseline model (Specific Probability) and an improvement of close to two in terms of HR@5.

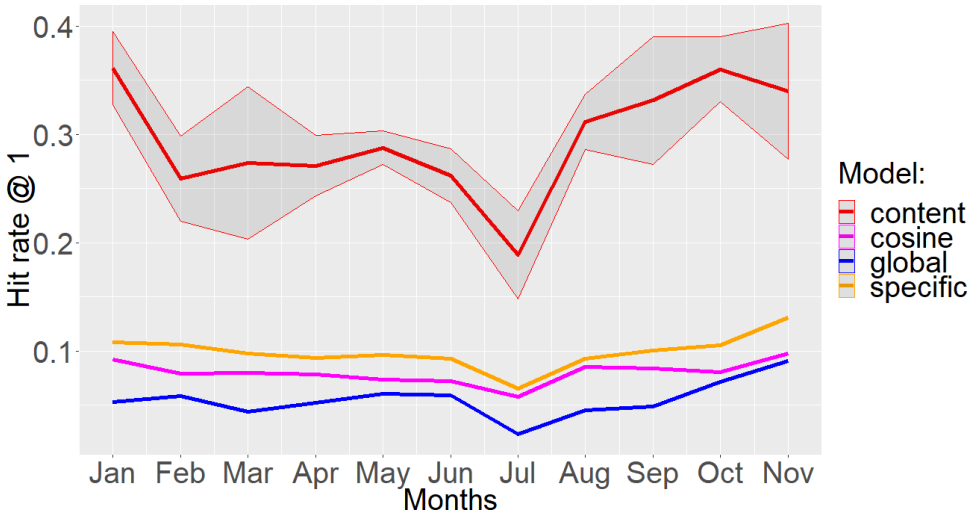


Figure 3.24: Sliding window - HR@1 for all models

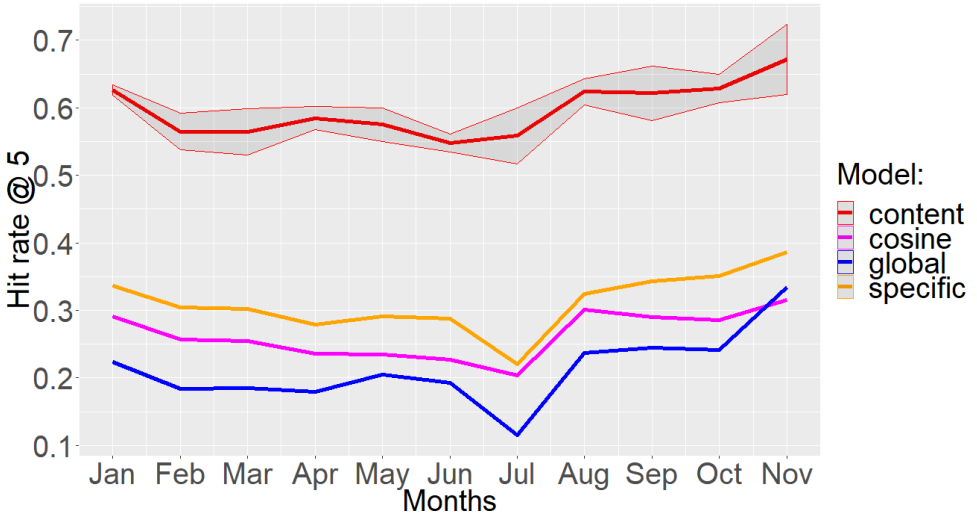


Figure 3.25: Sliding window - HR@5 for all models

Explainability

One of the advantages of using XGBoost that is of particular interest for the specified use case is the ability to provides the weight of contributions of each feature of the dataset to the prediction. This, helps us understand why a particular suggestion was made for a user and possibly understand the users'

Table 3.18: Sliding window: Average HR@1 and HR@5 for all months

Model	HR@1	HR@5
Content-based 5 run average	0.30	0.60
Global probability	0.06	0.21
Specific probability	0.10	0.31
Cosine	0.08	0.26

preferences.

In Table 3.19 we list the feature importance for a model trained using the entirety of the data from 2018 as training, except for December that served for testing. The Gain measures the feature contribution to the model compared to all others, the Cover measures the number of observations covered by this feature and Frequency represents the relative number of times the feature is used as a decision for the splits. For this specific model, we can see that the price is a very important feature, with some other contributions also appearing in the top ten, such as the release date of the destination handset, the amount of mobile LTE data used by the customer, the absolute AnTuTu score for the destination phone and its relative delta score in between handsets.

Table 3.19: Feature importance for CB model

Feature	Gain	Cover	Frequency
source_phone_price	0.31	0.18	0.27
dest_release_date_int	0.27	0.11	0.06
dest_max_price	0.07	0.03	0.05
dest_antutu_score	0.04	0.04	0.02
dest_mean_price	0.04	0.03	0.04
dest_approximate_price	0.03	0.04	0.01
dest_min_price	0.02	0.04	0.06
user_data_lte	0.02	0.02	0.04
delta_rel_antutu	0.02	0.02	0.01
dest_antutu_score	0.02	0.03	0.01

Another possible analysis is to use the predicted probability of trading to any handset and relate it with some feature of our choosing.

In Figure 3.26 we can see that the probability of trading devices is higher for positive deltas, as one would expect, and that it peaks when a absolute difference in ram between the two handsets of 2 to 3 Gigabytes exists.

Regarding the storage's deltas, represented in Figure 3.27, the peak appears when a difference of 25 to 100 Gigabytes in storage exists between the devices.

In Figure 3.28, it is shown that upgrades most often occur when a delta in AnTuTu scores of 50 to around 125 occurs.

3.5.3 Conclusion

The content-based approach performs clearly better than baselines, with the added advantage of also providing better explainability. This model can also be easily extended by adding new features on handsets and users. Additionally, we can use the analyses previously shown to support the vendor during the rules process.

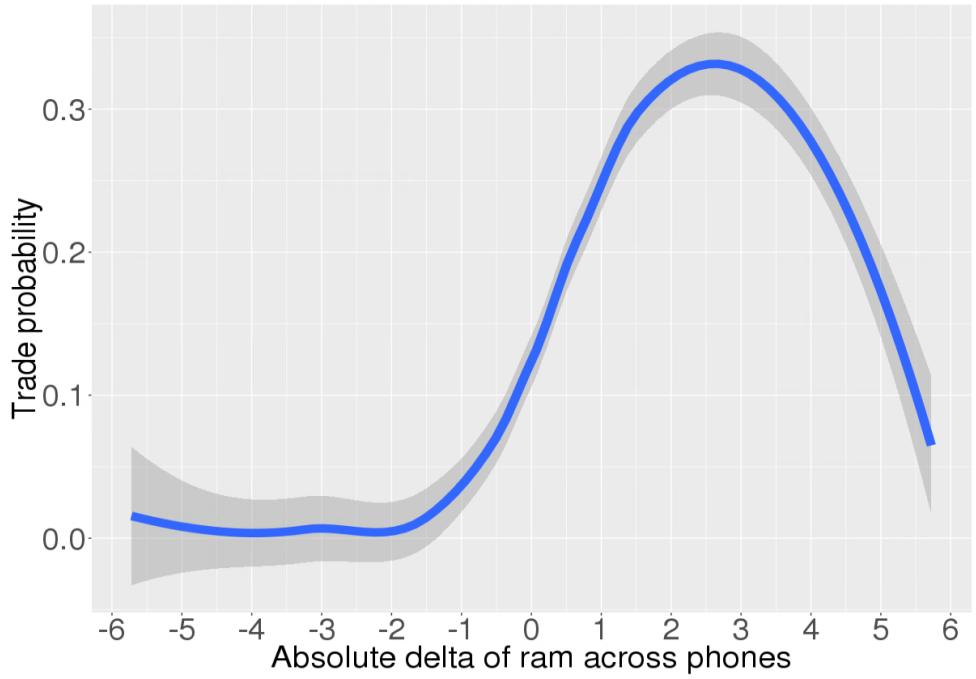


Figure 3.26: Trade probability by ram difference across handsets

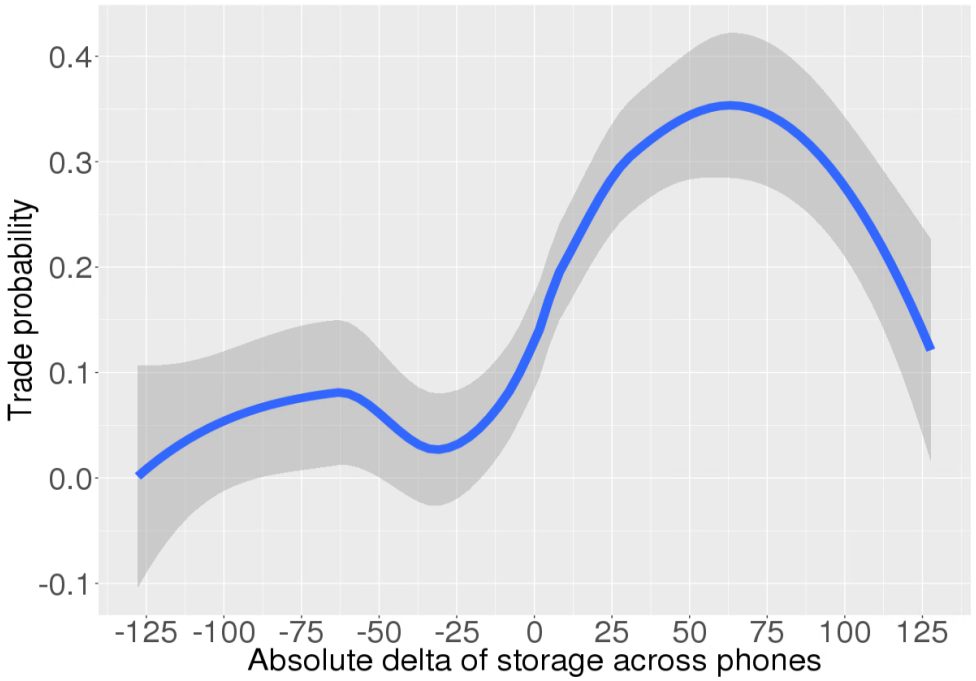


Figure 3.27: Trade probability by storage difference across handsets

3.6 Hybrid approach

To verify the potential to improvement the content based model by integrating other approaches in a hybrid system, a feature augmentation model was used, where the predictions from the baseline models were added as features to the content-based model. To test this approach, a similar methodology is taken

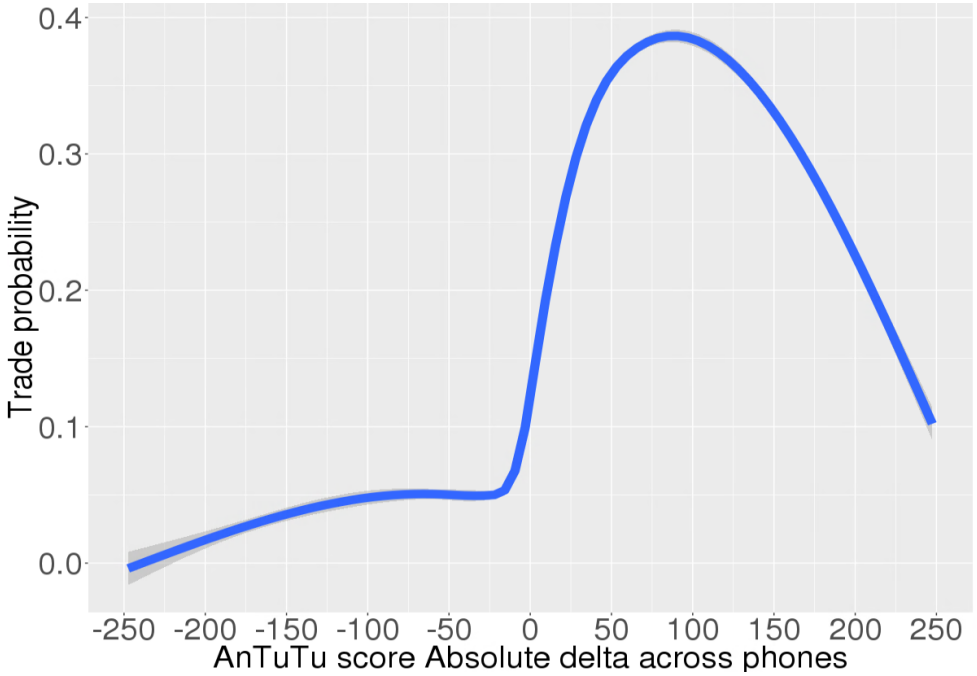


Figure 3.28: Trade probability by AnTuTu score difference across handsets

to that of the CB results analysis, with its respective windows. For these tests, all models had sampling added to them, so the results shown are the five run average of them.

3.6.1 Growing window using shifting test month

For these tests, only the CB model and the model with all the predictions from the baselines will be considered. As such, and as can be seen in Figure 3.29, the raw CB model always outperforms the hybrid model, with a very similar result curve for both, in regards to the HR@1 metric.

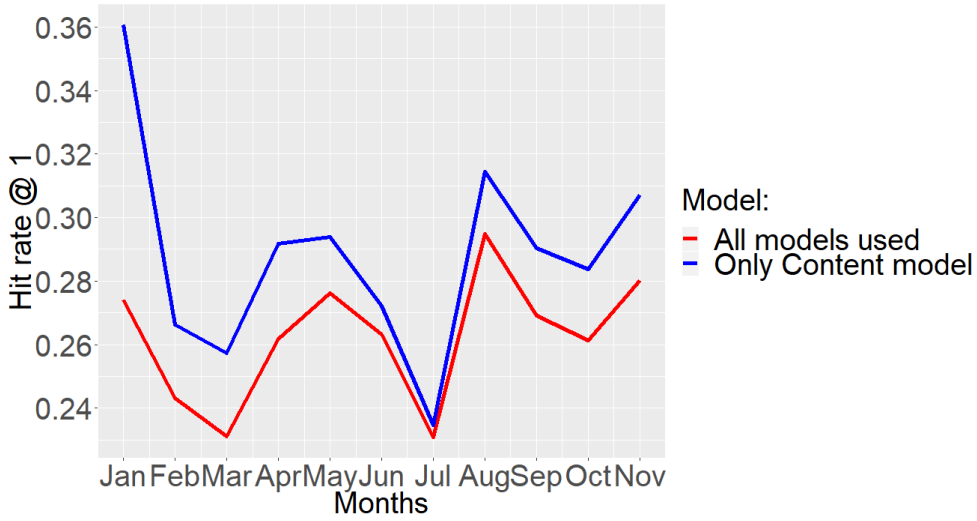


Figure 3.29: Growing window with shifting test month - HR@1 for all models

In regards to the performance when the HR@5 is considered, Figure 3.30 shows a different curve behaviour from what was seen at Figure 3.29, with the curves not matching anymore. Overall, the CB model outperformed the full hybrid model, except for the latter months, where the full hybrid model has a small edge over the raw CB model.

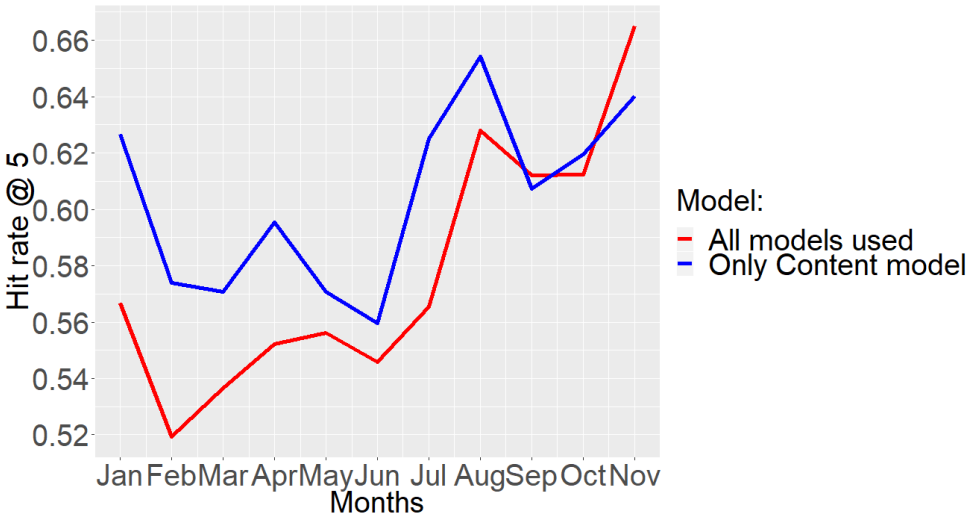


Figure 3.30: Growing window with shifting test month - HR@5 for all models

In Table 3.20 the all month average is shown for both models. In regards to the HR@1, the CB model edges with a gain of 0.03 while with HR@5, the lead falls to 0.02.

Table 3.20: Growing window with shifting test month: Average HR@1 and HR@5 for all months

Model	HR@1	HR@5
All models used	0.26	0.58
Content-based only	0.29	0.60

3.6.2 Sliding window

In Figure 3.31, it is shown the HR@1 performance for the raw CB model, the CB model combined with each of the baselines and the CB model using all the baselines. As can be seen, the best performer model were the raw CB and the CB with global probability added into it. Overall, the curves for all models are very similar.

The HR@5 for the sliding window is shown in Figure 3.32. As can be seen, the raw content model is clearly the best performer, with the content plus global being the second best. The curves for all models are very similar with the exception of the raw CB model, as it does not have the strong peaks present in the other models.

In Table 3.18 we show the average for all the models considered for testing. In regards to the HR@1

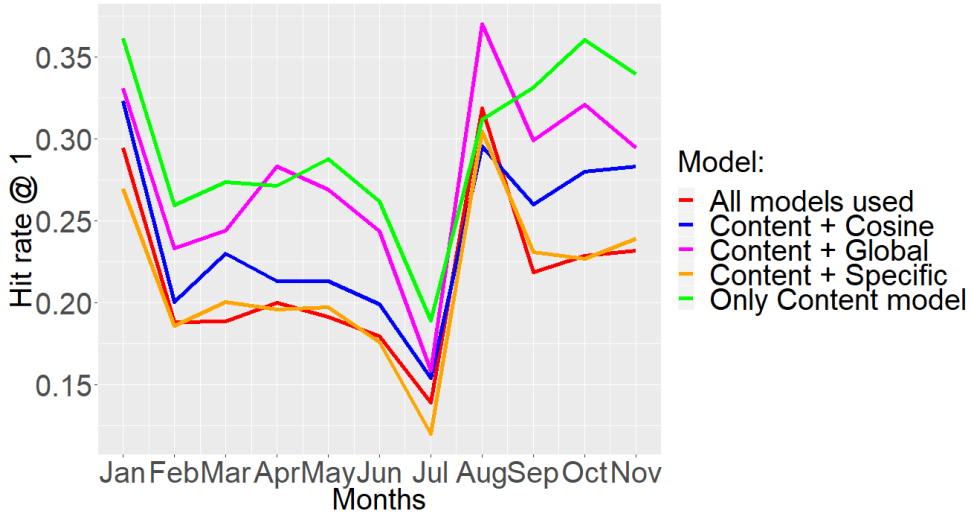


Figure 3.31: Sliding window - HR@1 for all models

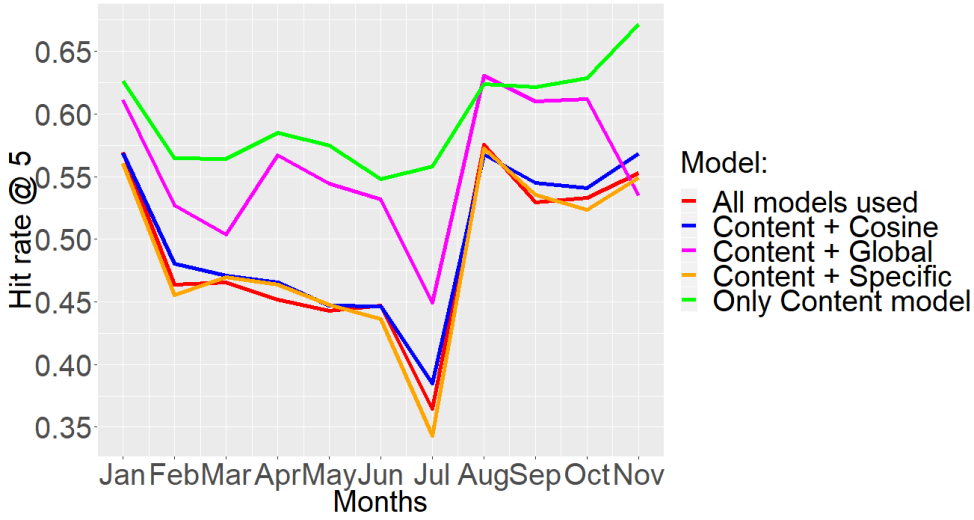


Figure 3.32: Sliding window - HR@5 for all models

metric, the best performing model was the raw CB model with a 0.02 gain compared to the CB model with Global Probability model added. Results are similar with HR@5, but the lead grows from 0.02 to 0.04.

3.6.3 Growing window using a fixed test month

For the growing window with the test month fixed as December, the HR@1 results for all models can be seen at Figure 3.33. For the first months, the raw CB model has a clear lead. Then, for the following months, the results are very similar for all models, except for the CB with cosine added, which performed worse than the others.

As for the HR@5, the story is similar in regard to the first months. then, in this case, the worst performer models are the raw content and the content with the cosine added. This result is surprising as the CB raw model was performing slightly better in all testing done before, which has not happened here. Still, if the raw CB model was trained using only two months, the performance would be equivalent to the other

Table 3.21: Sliding window: Average HR@1 and HR@5 for all months

Model	HR@1	HR@5
All models used	0.22	0.49
Content + Cosine	0.24	0.50
Content + Global	0.28	0.56
Content + Specific	0.21	0.49
Content-based only	0.30	0.60

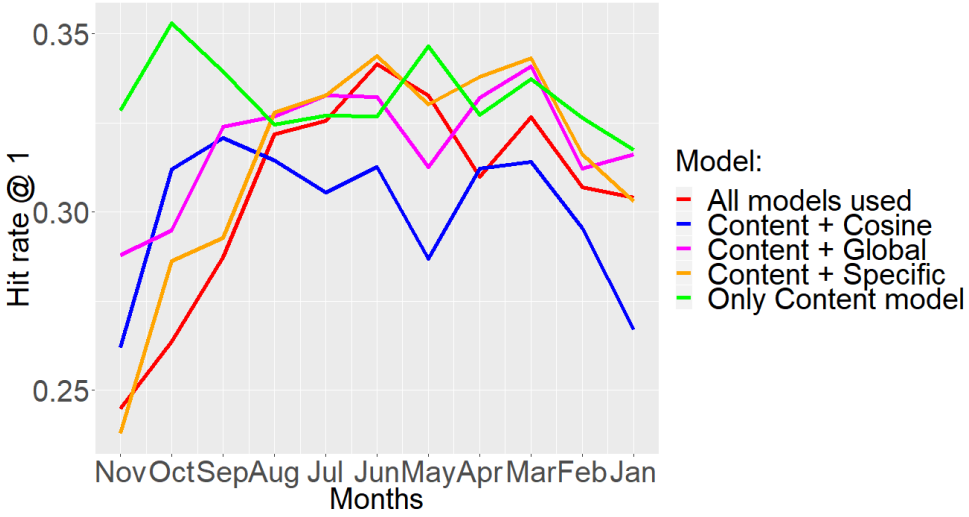


Figure 3.33: Growing window with fixed test month - HR@1 for all models

better performing model.

Resuming the results, in Table 3.17, the better performer model in regards to the HR@1 was the raw CB model with a gain of 0.02 compared to the second best model, while with HR@5, the best performer model was the content plus specific model beating the raw CB model by 0.02.

Table 3.22: Growing window with fixed test month: Average HR@1 and HR@5 for all months

Model	HR@1	HR@5
All models used	0.31	0.65
Content + Cosine	0.30	0.63
Content + Global	0.32	0.65
Content + Specific	0.31	0.66
Content-based only	0.33	0.64

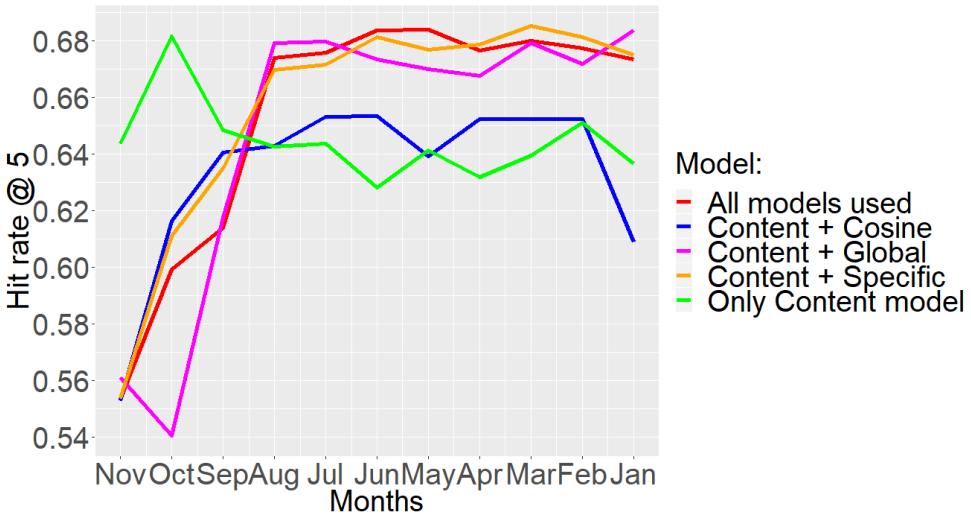


Figure 3.34: Growing window with fixed test month - HR@5 for all models

3.6.4 Hybrid method conclusion

Overall, the results from using this hybrid approach do not outperform the standalone CB model. The content model beats the other models in all cases except one. This is not to say that hybrid approaches would not beat the content-based model, but that this particular implementation of feature augmentation for hybridization was lackluster, thus suggesting the experimentation of other hybridization approaches.

3.7 Summary

In this section, the methodology and baseline algorithms have been shown and explained with an exploratory data analysis done on all data sources available to use. Then the three different approaches were tested: Collaborative filtering approaches provided mediocre results that were not capable of beating the baselines, the content-based approach had very satisfactory performance clearly beating the baselines and the hybrid approach, when compared to the content-based approach, did not perform as expected, providing very similar or a bit worse than the ones using only a CB model.

Chapter 4

Conclusions and future work

In this chapter, the conclusions will be detailed along with current limitations for the solution provided and future work to be done to further improve what explored in this dissertation.

4.1 Conclusions

In this dissertation, multiple approaches have been taken to try and tackle the problem of handset recommendation. Of the approaches taken, collaborative filtering provided poor results, content provided very substantial improvements over the baselines and hybrid methods showed no to very small negative improvements compared to that of the content models.

The solution proposed also provides good explainability on the obtained result and can be extended easily in the future, provided that the handset and user information is available.

4.2 Limitations

The need for accurate information on handset devices is something that is very important to the content-based model, but for the results presented was limited as information was only available for a subset of devices (285) and the trades were limited to those that occurred from a known to an also known device. In the future, a better source of information with full handset information for almost all devices would further heighten the proposed model.

Another limitation is that the model has not been tested in a production environment so further questions could arise from it's results.

4.3 Future work

Some future work has been identified that could result in performance benefits. These are bettering the criteria used for what is considered or not a definitive trade and exploring further hybrid approaches such

as a weighted hybrid systems. In Subsection 4.3.1 we compare the early results obtained from pointwise to pairwise approaches.

4.3.1 Pointwise and pairwise approach

Another consideration to take into account is the number of trades to be considered by the the loss function. Pointwise (element wise) approaches consider only one element, pairwise considers pairs of trades and listwise considers a whole list of trades that would then be ordered. In our case, due to how the model has been defined, the listwise approach would not make sense as the case in where it would make sense, making lists of trades per user, is simply too small to be of use.

To get some preliminary results, grid-search was performed for the pairwise approach, using the same grid used for the pointwise approach, with it's results shown in Table 4.1.

Table 4.1: Pairwise: Grid search hyperparameters chosen and best values

Hyperparameter	Values considered	Best value
max_depth	3, 5, 8, 10	3
eta	0.01, 0.1, 1	1
lambda	0.01, 0.1, 1	0.01
colsample_by_node	0.5, 0.75, 1	0.5

As can be see from the results in Figure 4.1, pointwise approaches show promise by scoring consistently higher than with pointwise ones.Due to positive feedback with early results, this topic would warrant some further research and work put into it.

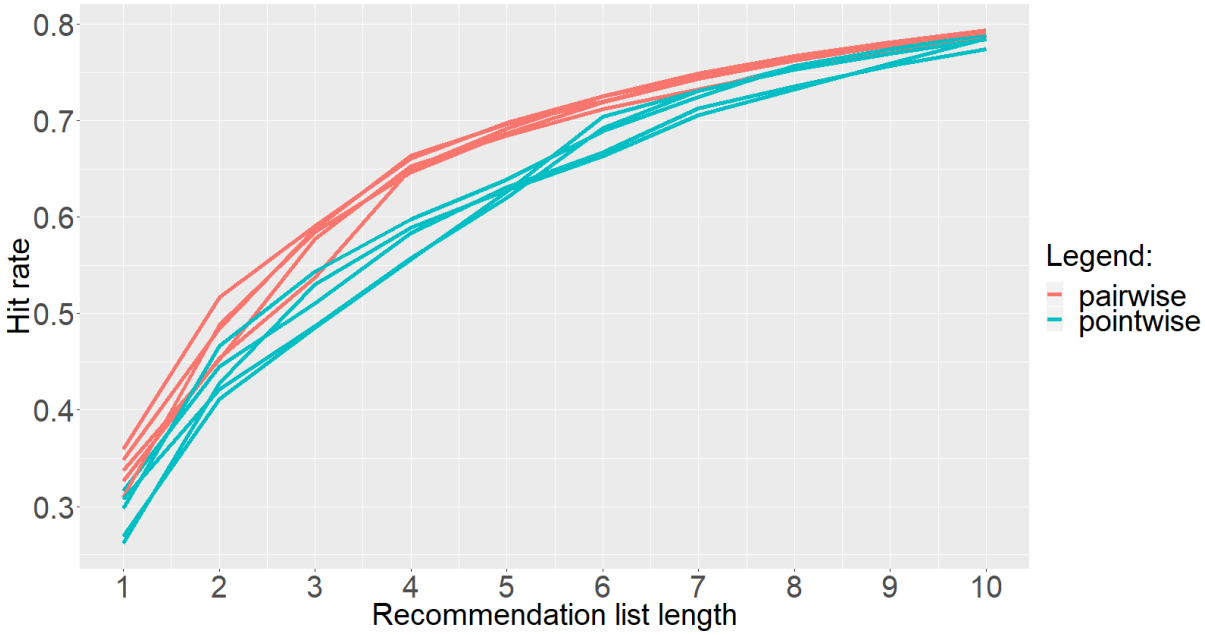


Figure 4.1: Preliminary results on pointwise and pairwise approaches

Bibliography

Claire Adam-Bourdarios, Glen Cowan, Cécile Germain, Isabelle Guyon, Balázs Kégl, and David Rousseau. The higgs boson machine learning challenge. In *Proceedings of the 2014 International Conference on High-Energy Physics and Machine Learning - Volume 42*, HEPML'14, pages 19–55. JMLR.org, 2014.

Charu C. Aggarwal. *Neighborhood-Based Collaborative Filtering*, pages 29–70. Springer International Publishing, Cham, 2016. ISBN: 978-3-319-29659-3.

Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc. ISBN: 1-55860-153-8.

Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22(2):207–216, June 1993. ISSN: 0163-5808. doi:10.1145/170036.170072.

Chris Anderson. *The Long Tail: Why the Future of Business Is Selling Less of More*. Hyperion, 2006. ISBN: 1401302378.

Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine learning*, 36(1-2):105–139, 1999.

James Bennett, Stan Lanning, and Netflix Netflix. The netflix prize. In *In KDD Cup and Workshop in conjunction with KDD*, 2007.

José M Bernardo and Adrian FM Smith. *Bayesian theory*, volume 405. John Wiley & Sons, 2009.

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3: 993–1022, March 2003. ISSN: 1532-4435.

John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, pages 43–52, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc. ISBN: 1-55860-555-X.

Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, Aug 1996. ISSN: 1573-0565. doi:10.1007/BF00058655.

Robin Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, Nov 2002. ISSN: 1573-1391. doi:10.1023/A:1021240730564.

Erion Çano and Maurizio Morisio. Hybrid recommender systems: A systematic literature review. *Intelligent Data Analysis*, 21(6):1487–1524, 2017.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

pham Cuong, Yiwei Cao, Ralf Klamma, and Matthias Jarke. A clustering approach for collaborative filtering recommendation using social network analysis. *Journal of Universal Computer Science (j-jucs)*, 17:583–604, 01 2011.

Brian Everitt. *An introduction to latent variable models*. London ; New York : Chapman and Hall, 1984. ISBN: 0412253100. Includes index.

Daniel M. Fleder and Kartik Hosanagar. Recommender systems and their impact on sales diversity. In *Proceedings of the 8th ACM Conference on Electronic Commerce*, EC '07, pages 192–199, New York, NY, USA, 2007. ACM. ISBN: 978-1-59593-653-0. doi:10.1145/1250910.1250939.

Yoav Freund, Robert Schapire, and Naoki Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

Jerome Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29, 11 2000. doi:10.1214/aos/1013203451.

Gene H Golub and Christian Reinsch. Singular value decomposition and least squares solutions. *Numerische mathematik*, 14(5):403–420, 1970.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Prem Gopalan, Jake M. Hofman, and David M. Blei. Scalable recommendation with hierarchical poisson factorization. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*, UAI'15, pages 326–335, Arlington, Virginia, United States, 2015. AUAI Press. ISBN: 978-0-9966431-0-8.

Guibing Guo, Jie Zhang, Zhu Sun, and Neil Yorke-Smith. Librec: A java library for recommender systems. In *UMAP Workshops*, 2015.

Michael Hahsler. recommenderlab: A framework for developing and testing recommendation algorithms, 2015.

Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Min. Knowl. Discov.*, 8(1):53–87, January 2004. ISSN: 1384-5810. doi:10.1023/B:DAMI.0000005258.31418.83.

F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19, December 2015. ISSN: 2160-6455. doi:10.1145/2827872.

Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. Fast matrix factorization for online recommendation with implicit feedback. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, pages 549–558, New York, NY, USA, 2016. ACM. ISBN: 978-1-4503-4069-4. doi:10.1145/2911451.2911489.

Thomas Hofmann. Latent semantic models for collaborative filtering. *ACM Trans. Inf. Syst.*, 22(1):89–115, January 2004. ISSN: 1046-8188. doi:10.1145/963770.963774.

Thomas Hofmann and Jan Puzicha. Latent class models for collaborative filtering. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, IJCAI '99, pages 688–693, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN: 1-55860-613-0.

J. B. Rollins. Foundational Methodology for Data Science, 2015.

Michael Jahrer and Andreas Töscher. Collaborative filtering ensemble for ranking. In *Proceedings of the 2011 International Conference on KDD Cup 2011-Volume 18*, pages 153–167. JMLR. org, 2011.

Santosh Kabbur, Xia Ning, and George Karypis. Fism: Factored item similarity models for top-n recommender systems. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 659–667, New York, NY, USA, 2013. ACM. ISBN: 978-1-4503-2174-7. doi:10.1145/2487575.2487589.

Shah Khusro, Zafar Ali, and Irfan Ullah. Recommender systems: Issues, challenges, and research opportunities. In Kuinam J. Kim and Nikolai Joukov, editors, *Information Science and Applications (ICISA) 2016*, pages 1179–1189, Singapore, 2016. Springer Singapore. ISBN: 978-981-10-0557-2.

Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, August 2009. ISSN: 0018-9162. doi:10.1109/MC.2009.263.

Xuan Nhat Lam, Thuc Vu, Trong Duc Le, and Anh Duc Duong. Addressing cold-start problem in recommendation systems. In *Proceedings of the 2Nd International Conference on Ubiquitous Information Management and Communication*, ICUIMC '08, pages 208–211, New York, NY, USA, 2008. ACM. ISBN: 978-1-59593-993-7. doi:10.1145/1352793.1352837.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

Daniel D Lee and H Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788, 1999.

Andy Liaw, Matthew Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, January 2003. ISSN: 1089-7801. doi:10.1109/MIC.2003.1167344.

Pasquale Lops, Marco De Gemmis, and Giovanni Semeraro. Content-based recommender systems: State of the art and trends. In *Recommender systems handbook*, pages 73–105. Springer, 2011.

Sanjay Kumar Malik and SAM Rizvi. Information extraction using web usage mining, web scrapping and semantic annotation. In *2011 International Conference on Computational Intelligence and Communication Networks*, pages 465–469. IEEE, 2011.

Sean M. McNee, John Riedl, and Joseph A. Konstan. Being accurate is not enough: How accuracy metrics have hurt recommender systems. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '06, pages 1097–1101, New York, NY, USA, 2006. ACM. ISBN: 1-59593-298-4. doi:10.1145/1125451.1125659.

Xia Ning and George Karypis. Slim: Sparse linear methods for top-n recommender systems. In *Proceedings of the 2011 IEEE 11th International Conference on Data Mining*, ICDM '11, pages 497–506, Washington, DC, USA, 2011. IEEE Computer Society. ISBN: 978-0-7695-4408-3. doi:10.1109/ICDM.2011.134.

NPD Connected Intelligence. The Average Upgrade Cycle of a Smartphone in the U.S. is 32 Months, According to NPD Connected Intelligence, 2018.

Michael J. Pazzani and Daniel Billsus. Content-based recommendation systems. In *The Adaptive Web*, 2007.

J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, Mar 1986. ISSN: 1573-0565. doi:10.1007/BF00116251.

Steffen Rendle and Christoph Freudenthaler. Improving pairwise learning for item recommendation from implicit feedback. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, WSDM '14, pages 273–282, New York, NY, USA, 2014. ACM. ISBN: 978-1-4503-2351-2. doi:10.1145/2556195.2556248.

Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, pages 452–461, Arlington, Virginia, United States, 2009. AUAI Press. ISBN: 978-0-9749039-5-8.

Alan Said and Alejandro Bellogín. Comparative recommender system evaluation: Benchmarking recommendation frameworks. In *Proceedings of the 8th ACM Conference on Recommender Systems*, RecSys '14, pages 129–136, New York, NY, USA, 2014. ACM. ISBN: 978-1-4503-2668-1. doi:10.1145/2645710.2645746.

Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Application of dimensionality reduction in recommender system-a case study. Technical report, Minnesota Univ Minneapolis Dept of Computer Science, 2000.

Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, pages 285–295, New York, NY, USA, 2001. ACM. ISBN: 1-58113-348-0. doi:10.1145/371920.372071.

Badrul Munir Sarwar. Sparsity, scalability, and distribution in recommender systems. In *Sparsity, Scalability, and Distribution in Recommender Systems*, Minneapolis, MN, USA, 2001. University of Minnesota. ISBN: 0-493-04207-5. AAI9994525.

J Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. Collaborative filtering recommender systems. In *The adaptive web*, pages 291–324. Springer, 2007.

Guy Shani and Asela Gunawardana. *Evaluating Recommendation Systems*, volume 12, pages 257–297. Springer, 01 2011.

Himani Sharma and Sunil Kumar. A survey on decision tree algorithms of classification in data mining. *International Journal of Science and Research (IJSR)*, 5, 04 2016.

Yue Shi, Martha Larson, and Alan Hanjalic. List-wise learning to rank with matrix factorization for collaborative filtering. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 269–272, New York, NY, USA, 2010. ACM. ISBN: 978-1-60558-906-0. doi:10.1145/1864708.1864764.

Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, Nuria Oliver, and Alan Hanjalic. Climf: learning to maximize reciprocal rank with collaborative less-is-more filtering. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 139–146. ACM, 2012.

Harald Steck. Evaluation of recommendations: Rating-prediction and ranking. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 213–220, New York, NY, USA, 2013. ACM. ISBN: 978-1-4503-2409-0. doi:10.1145/2507157.2507160.

Gábor Takács and Domonkos Tikk. Alternating least squares for personalized ranking. In *Proceedings of the Sixth ACM Conference on Recommender Systems*, RecSys '12, pages 83–90, New York, NY, USA, 2012. ACM. ISBN: 978-1-4503-1270-7. doi:10.1145/2365952.2365972.

Luís Torgo. Inductive learning of tree-based regression models. *Ai Communications*, 13, 04 2000.

Maksims Volkovs, Guang Wei Yu, and Tomi Poutanen. Content-based neighbor models for cold start in recommender systems. In *Proceedings of the Recommender Systems Challenge 2017*, RecSys Challenge '17, pages 7:1–7:6, New York, NY, USA, 2017. ACM. ISBN: 978-1-4503-5391-5. doi:10.1145/3124791.3124792.

Hanna M. Wallach. Topic modeling: Beyond bag-of-words. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 977–984, New York, NY, USA, 2006. ACM. ISBN: 1-59593-383-2. doi:10.1145/1143844.1143967.

Zongben Xu, Hai Zhang, Yao Wang, XiangYu Chang, and Yong Liang. L 1/2 regularization. *Science China Information Sciences*, 53(6):1159–1169, 2010.

Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Comput. Surv.*, 52(1):5:1–5:38, February 2019. ISSN: 0360-0300. doi:10.1145/3285029.

Qian Zhao, F. Maxwell Harper, Gediminas Adomavicius, and Joseph A. Konstan. Explicit or implicit feedback? engagement or satisfaction?: A field experiment on machine-learning-based recommender systems. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, SAC '18, pages 1331–1340, New York, NY, USA, 2018. ACM. ISBN: 978-1-4503-5191-1. doi:10.1145/3167132.3167275.

Jacek M Zurada. *Introduction to artificial neural systems*, volume 8. West publishing company St. Paul, 1992.

Appendix A

Librec: hyperparameters used

The following are the parameters used on the collaborative filtering algorithms tested by using the librec library:

- aobpr:
  - rec.item.distribution.parameter = 500
  - rec.iterator.learnrate=0.01
  - rec.iterator.learnrate.maximum=0.01
  - rec.iterator.maximum=100
  - rec.user.regularization=0.01
  - rec.item.regularization=0.01
  - rec.factor.number=10
  - rec.learnrate.decay=1.0
- aspectmodelranking:
  - rec.iterator.maximum=20
  - rec.pgm.burnin=10
  - rec.pgm.samplelag=10
  - rec.topic.number=10
- bpoissmf:
  - rec.iterator.maximum=100
  - rec.factor.number=100
  - rec.a=0.3
  - rec.a.prime=0.3
  - rec.b.prime=1.0
  - rec.c=0.3
  - rec.c.prime=0.3
  - rec.d.prime=1.0

- bpr:
  - rec.iterator.learnrate=0.01
  - rec.iterator.learnrate.maximum=0.01
  - rec.iterator.maximum=50
  - rec.user.regularization=0.01
  - rec.item.regularization=0.01
  - rec.factor.number=100
  - rec.learnRate.decay=0.1
- climf:
  - rec.iterator.learnrate=0.001
  - rec.iterator.learnrate.maximum=0.01
  - rec.iterator.maximum=100
  - rec.user.regularization=0.01
  - rec.item.regularization=0.01
  - rec.factor.number=10
  - rec.learnrate.decay=1.0
- eals:
  - rec.iterator.maximum=20
  - rec.user.regularization=0.01
  - rec.item.regularization=0.01
  - rec.factor.number=200
- fismauc:
  - rec.iterator.learnrate=1.05
  - rec.iterator.learnrate.maximum=1.05
  - rec.iterator.maximum=10
  - rec.user.regularization=0.01
  - rec.item.regularization=0.01
  - rec.factor.number=10
  - rec.learnrate.decay=1.0
  - rec.fismauc.rho=0.7
  - rec.fismauc.alpha=1.5
- itembigram:
  - rec.iterator.maximum=100
  - rec.topic.number=30
  - rec.user.dirichlet.prior=0.01
  - rec.topic.dirichlet.prior=0.01

  - rec.pgm.burnin=10
  - rec.pgm.samplelag=10
- itemknn:
  - rec.similarity.class=cos
  - rec.neighbors.knn.number=50
  - rec.similarity.shrinkage=10
- lda:
  - rec.iterator.maximum=1000
  - rec.topic.number = 50
  - rec.user.dirichlet.prior=0.01
  - rec.topic.dirichlet.prior=0.01
  - rec.pgm.burnin=100
  - rec.pgm.samplelag=10
- listrankmf:
  - data.splitter.ratio=user
  - rec.iterator.learnrate=1.05
  - rec.iterator.learnrate.maximum=100
  - rec.iterator.maximum=30
  - rec.user.regularization=0.05
  - rec.item.regularization=0.05
  - rec.factor.number=20
  - rec.learnrate.decay=1.0
- plsa:
  - rec.iteration.learnrate=0.01
  - rec.iterator.maximum=100
  - rec.topic.number = 10
- rankals:
  - rrec.iterator.learnrate=0.01
  - rec.iterator.learnrate.maximum=0.01
  - rec.iterator.maximum=100
  - rec.user.regularization=0.01
  - rec.item.regularization=0.01
  - rec.factor.number=10
  - rec.learnrate.decay=1.0
- ranksgd:

  - rec.iterator.learnrate=0.01
  - rec.iterator.learnrate.maximum=0.01
  - rec.iterator.maximum=30
  - rec.user.regularization=0.01
  - rec.item.regularization=0.01
  - rec.factor.number=10
  - rec.learnrate.decay=1.0
- slim:
  - rec.similarity.class=cos
  - rec.recommender.similarities=item
  - rec.iterator.maximum=15
  - rec.similarity.shrinkage=10
  - rec.neighbors.knn.number=50
  - rec.recommender.earlystop=false
  - rec.slim.regularization.l1=0.02
  - rec.slim.regularization.l2=10