Politecnico di Milano
School of Industrial and Information Engineering
Master of Science in Computer Science and Engineering
Dipartimento di Elettronica, Informazione e Bioingegneria

Practical Quantum Computing: a Collaborative-Driven Quantum Feature Selection Approach for the Cold-Start Problem in Recommender Systems

Supervisor: Prof. Paolo Cremonesi
Co-supervisor: Dott. Maurizio Ferrari Dacrema

Master Thesis by:
Riccardo Nembrini, 912887

Academic Year 2019-2020

Abstract

Recommender systems are tools aimed at recommending items to the users of a platform, trying to predict their preferences. The popularity of this tools has been increasing with the continuous growth of the Web, and they have been extensively implemented on platforms that sell contents to their users, such as Netflix or Amazon.

In order to make recommendations, a recommender system relies on two main types of information, content and collaborative. The first refers to the features characterizing the items on the platform, such as the actors and the release year of a movie, or the author and the genre of a book. Collaborative information, instead, consists of the past interactions between users and items and it is known in the recommender systems literature that methods based on this type of information generally perform better than content-based ones.

However, it is not always possible to rely on collaborative information, for example when a new item is added to the platform. This situation is called cold-start scenario and in such cases only content-based and collaborative with side information methods are effective, since content information is available, as opposed to collaborative one.

Different techniques have been proposed to cope with the cold-start problem by optimizing content-based approaches. For example, feature weighting techniques, that weight item features based on various criteria. In particular, recent machine learning approaches have obtained promising results by estimating weights on the base of collaborative information.

Further developing this concept, in this thesis we propose a new feature selection model able to embed collaborative information into a content-based model. In particular, in order to efficiently tackle the given problem, we apply quantum annealing, one of the current paradigms of quantum computing. Quantum annealing has been acquiring great industrial interest in recent years due to its ability of efficiently solving practical NP-hard optimization problems.

Sommario

I sistemi di raccomandazione sono degli strumenti utilizzati per raccomandare degli oggetti agli utenti di una piattaforma, cercando di predire le loro preferenze. La popolarità di questi strumenti è aumentata costantemente con la crescita del Web, ed essi sono stati integrati sempre più in piattaforme che vendono contenuti ai loro utenti, come Netflix o Amazon.

Per proporre raccomandazioni, questi strumenti si affidano a due principali tipi di informazione, quella contenutistica e quella collaborativa. La prima si riferisce a quelle caratteristiche che rappresentano gli oggetti di una piattaforma, come gli attori e l'anno di uscita di un film, o l'autore e il genere di un libro. L'informazione collaborativa, invece, consiste nelle interazioni passate tra utenti e oggetti ed è noto, nella letteratura dei sistemi di raccomandazione, che i metodi basati su questi tipo di informazione ottengano risultati generalmente migliori di quelli basati sui contenuti.

Tuttavia, non è sempre possibile fare affidamento sull'informazione collaborativa, per esempio quando un nuovo oggetto viene aggiunto alla piattaforma. Questa situazione è chiamata scenario cold-start e, in questi casi, solo i metodi basati sul contenuto e quelli collaborativi con informazione laterale sono efficaci, considerato che l'informazione contenutistica è l'unica disponibile.

Diverse tecniche sono state proposte per trattare il problema cold-start ottimizzando gli approcci basati sul contenuto. Per esempio, esistono tecniche che ponderano le caratteristiche degli oggetti secondo vari criteri. In particolare, recenti approcci basati sul machine learning hanno ottenuto risultati promettenti, stimando i pesi delle caratteristiche sulla base dell'informazione collaborativa.

Sviluppando ulteriormente questo concetto, in questa tesi proponiamo un nuovo metodo di selezione delle caratteristiche in grado di incorporare l'informazione collaborativa in un modello contenutistico. Nello specifico, per affrontare efficientemente il problema ottenuto, utilizziamo il quantum annealing, uno degli attuali paradigmi di quantum computing. Il quantum annealing ha acquisito sempre più interesse industriale negli ultimi anni, grazie alla sua capacità di risolvere efficientemente problemi pratici di ottimizzazione classificati come NP-difficili.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recommender systems are tools used to recommend items to users on a platform. With the continuous growth of the Web, they became more and more relevant for platforms that sell products or contents to their customers, such as Amazon or Netflix. Indeed, recommender systems are able to recommend to the users some contents that they may not have bought or watched without any suggestion. In order to make recommendations, recommender systems try to predict the user preferences relying mainly on two different types of information, content and collaborative.

Content information consists of all the characteristics that an item has, also called features. For example, a movie has its actors, a release date or a production company, while a book has an author, a publisher and so on. Features can be editorial, listed and verified by an official source, or user created, such as tags. Content information is exploited by content-based filtering methods, which tend to recommend to the users the most similar items, in terms of features, to the ones they interacted with. Thus, if a user already watched *Iron Man* and *Iron Man 2*, a content-based recommender system would probably recommend them movies such as *Iron Man 3* or *The Avengers*. However, this could often result in poor recommendations, for example if a user that recently bought a new smartphone would be recommended another one just because they have similar features.

Collaborative information, instead, consists of the past users interactions with the items on the platform. These interactions can be explicit, which means that the user rated the item on a certain scale, or implicit, which means that the user either had any interaction with the item or none at all. Recommender systems exploiting this kind of information are called collaborative filtering methods. They try to predict the user preferences based on the past interactions of all the users with all the items on the platform. Collaborative filtering methods usually make better recommendations than content-based ones. Recalling the same example as before, a user that already watch a movie like *Iron Man* could now be recommended different movies, in terms of features, that other users that watched *Iron Man* liked.

However, it is not always possible to make use of collaborative information. For example, when a new item is added to the platform, no user has ever interacted with it, thus making it impossible to recommend the item with collaborative filtering methods. This is called cold-start scenario and the newly added items are called cold items. The only way to tackle this recommendation problem is by making use of content information, for example using a content-based filtering technique, or a collaborative filtering method with side information, which enriches the collaborative model with content information.

It should be noted that pure content information is often not enough to cope with the cold-start problem. As we already stated, content-based methods are in general outperformed by collaborative ones. Therefore, many solutions have been proposed in recommender systems literature to improve content-based models. One of them is feature weighting, which aims at giving different weights to the various item features, attempting to describe a real user's interest in the available features. Different feature weighting techniques have been adopted in recommender systems, for example techniques derived from the information retrieval field, such as TF-IDF. Some methods proposed in recent years have obtained promising results by embedding collaborative information into a content-based models via machine learning techniques.

Further developing this concept, we propose a new feature selection method aimed at selecting the item features that best incorporate collaborative information, in order to improve a content-based model. We called this method *Collaborative-driven Quantum Feature Selection*, where the term quantum in the name comes from the way we solve the selection problem. We model the feature selection procedure as a quadratic unconstrained binary optimization problem. In order to efficiently tackle this problem, we make use of quantum annealing, one of the current paradigms of quantum computing.

Quantum annealing is a technique used to search for solutions of NP-hard optimization problems. Thanks to the continuous research and development of physical quantum annealers, devices that implement this technique making use of actual quantum mechanics phenomena, quantum annealing has increasingly gained great industrial interest.

In this thesis we present some fundamentals of recommender systems and quantum annealing, on top of which we consequently build the mathematical model of our technique. Then, we report the detailed results of our experiments, discussing how collaborative and content information influences the quantum feature selection, and how the latter is effective in improving a content-based model in a cold-start scenario.

Chapter 2

State of the art

Recommender Systems are a particular kind of information filtering systems used to predict how much a user would like an item. This topic gained increasing importance with the continuous development of the Web, especially as a medium for business, e-commerce and media consumption [2].

In this chapter, we are going to see how recommender systems are structured, what are some of the models used to make recommendations and how they work. Then, we are going to discuss the so-called *cold-start problem* and some *feature weighting techniques.* At the end of the chapter, a description of *quantum annealing* will be proposed, explaining, in particular, which kind of problems it can solve.

2.1 Recommender Systems Structure

Depending on the type of problem that it has to solve, a recommender system can have different structures and use different models. However, the objective is always the same: recommend items to users, users to items or even users to other users (for example in a social network). In this work, we will only consider the first case and, through this chapter, we will see different scenarios and models used to predict the preferences of users towards items.

The data we are going to deal with are the users' interactions with items and the items (or users) characteristics, also called features. Usually, this information are collected in convenient data structures used to build the models:

- URM: the User Rating Matrix is the matrix that contains interactions of each user with the items on the platform. These interactions can be of two types. Implicit interactions represent the fact that users interacted with items in any particular way, for example watching a movie or buying some product. This information is translated as boolean values (0 or 1) in the URM. Explicit interactions are, instead, real values, often given by the users themselves to the items, such as ratings. If we call U the set of all users and I the set of all items, the URM, which we will call, from now on, R, is a $|U| \times |I|$ matrix. In particular, we will use the notation R_{ui} to represent the interaction between user u and item i, which is the value at row u and column i of R.

- ICM: the Item Content Matrix is the matrix that contains information about items' features. The features of an item can be very heterogeneous, for example, to describe a movie we could have actors, production year, concepts extracted from the plot and so on. Thus, the representation that is used in the ICM is a binary one: 1 if an item has some feature and 0 otherwise. Calling F the set of all the item features, the ICM is a $|I| \times |F|$ boolean matrix. Similarly to the URM, we will use the notation ICM_{if} to refer to the value at row i and column f of the ICM, which tells if item i has feature f.
- UCM: analogously to the ICM, the User Content Matrix is the matrix that contains information about users' features. If we call G the set of all the user features, the UCM is a $|U| \times |G|$ boolean matrix. We use the notation UCM_{ug} to indicate the value in row u and column g, which tells if user u has feature g.

In general, in order to make recommendations, we can use different sources of information. For example, let's consider a multimedia platform like Netflix. After watching a TV series, a user could receive recommendations based on what other users saw after that same series. This kind of information is used by *collaborative filtering* methods. Otherwise, the user could also be suggested to watch new TV series that have common characteristics with the one that they have just finished, such as the same actors or the same genre. This information is used in *content-based filtering* models. Collaborative filtering and content-based filtering are two of the most used categories of recommender systems.

2.2 Collaborative Filtering

Collaborative filtering methods exploit the assumption that observed ratings (the ones belonging to matrix R) are often highly correlated across users and items [2]. For example, let's consider two users that rated the movies they watched very similarly. When we want to recommend a new movie to one of them, we could rely on the ratings given by the other one, since their tastes are probably very similar as well.

As it is said in [52], "collaborative filtering is considered to be the most popular and widely implemented technique in RS".

There are different approaches to collaborative filtering. Here, three categories used throughout this work are presented: *neighborhood-based*, *matrix factorization* and *graph-based.*

2.2.1 Neighborhood-based

Neighborhood-based methods are relatively simple to implement and explain, since they predict the missing ratings from the given ratings on the basis of the neighborhoods of users or items. The missing ratings can be directly computed in-memory from the stored ratings or from a previously created model. Depending on the neighborhood we are using, we are dealing with user-based or item-based methods.

User-based

When making a recommendation to some user, the missing ratings are determined with respect to the ratings given by the users that are more similar to the target user. This can be done by weighting the ratings given by each user v with the similarity between them and the target user u. Thus, the rating $\hat{R}_{ui}$, given by user u to item i, can be estimated as:

$$\hat{R}_{ui} = \frac{\sum_{v \in U} R_{vi} \cdot sim_U(u, v)}{\sum_{v \in U} |sim_U(u, v)|}$$

where $sim_U(u, v)$ is a similarity function defined between two users, which quantifies how much the ratings they gave are similar.

Item-based

Instead, item-based methods estimate the missing ratings of a user by considering the ratings they gave to the most similar items to the ones they interacted with. These methods compute the predicted rating given by user u to item i as a sum of the interactions between user u and each item j, weighted by the similarity of each item j to the target item i:

$$\hat{R}_{ui} = \frac{\sum_{j \in I} R_{uj} \cdot sim_I(i, j)}{\sum_{j \in I} |sim_I(i, j)|}$$

where $sim_I(i, j)$ is a similarity function defined between two items, which quantifies how much the ratings they obtained are similar.

Model-based Neighborhood Methods

The model-based variant of these methods requires the computation of a model called *similarity matrix* and denoted by S. Since in this work we only deal with the item-based variant, we can consider S as an $|I| \times |I|$

matrix. Every element S_{ij} of this matrix is the similarity between item i and item j, computed by the function $sim_I(i,j)$. The user rating matrix can then be estimated from the similarity matrix as:

$$\hat{R} = R \cdot S$$

While this approach is straightforward, many variants have been proposed in the literature. In particular, when computing the similarity matrix, some *hyperparameters* can be adjusted in order to obtain different recommendations. Hyperparameters are those kind of parameters that a system needs to build the model. In the case of neighborhood methods, they are the following ones:

- *Neighborhood size*: determines how many neighbors are considered for prediction. For each column i of S, only the k most similar items to item i are kept, while the similarity values of the rest are set to 0. From now on, this parameter will be called *top-k*.
- *Similarity measure*: determines the measure to use when computing the similarity between two items i and j with the function $sim_I(i,j)$. Many similarity measures have been proposed in the literature, such as *Cosine* [55], *Jaccard* [51], *Asymmetric Cosine* [3], *Dice-Sørensen* [19] and *Tversky* [61] similarities. In particular, in this work we extensively use cosine similarity, which is computed as

$$sim_I(i,j) = cos(\vec{i},\vec{j}) = \frac{\vec{i} \cdot \vec{j}}{||\vec{i}||_2 \cdot ||\vec{j}||_2} \tag{2.1}$$

  and reduces to Jaccard similarity when the item vectors are boolean.
- *Shrinkage*: a *shrink term* is introduced into the similarity function in order to lower the similarity of items that have only few interactions [6]. Cosine similarity with shrinkage can be computed as:

$$sim_I(i,j) = \frac{\vec{i} \cdot \vec{j}}{||\vec{i}||_2 \cdot ||\vec{j}||_2 + c}$$

  where c is the shrink term.
- *Feature weighting*: feature weighting can be used to weight ratings as proposed in [63]. Possible weighting techniques are TF-IDF and BM25.
- *Normalization*: this parameter determines if the similarity should be normalized or not. For example, Cosine similarity in equation (2.1) presents normalization at the denominator, while it would reduce to a *dot product* without normalization.

The best values for these hyperparameters can be learned through optimization procedures.

2.2.2 Matrix Factorization

Matrix factorization (MF) is a family of latent factor models widely used in recommender systems. Latent factor models aim to describe given information in a different dimension space than the original one. In particular, they achieve this by using dimensionality reduction techniques.

Thus, the key concept is to decompose the interactions matrix R in more matrices with lower dimensionality, introducing the latent factors. We can factorize R in two matrices U and V which correspond respectively to the projection of users and items into the latent factors space:

$$R = U \cdot V^T$$

where U is a $|U| \times |L|$ matrix, while V is a $|I| \times |L|$ matrix, with L being the set of latent factors of the factorization.

Since we are interested in estimating the missing ratings, in practice we search for an approximated factorization:

$$\widetilde{R} = U \cdot V^T$$

This way, it is possible to predict the rating a user u would give to an item i as the element at row u and column i of matrix $\widetilde{R}$:

$$\widetilde{R}_{ui} = U_u \cdot V_i^T$$

Various factorization methods have been proposed through the literature, such as SVD (Singular Value Decomposition) [37], AsymmetricSVD [36] or iALS (implicit Alternating Least Squares) [30]. The one we are going to present and use in our work is the most basic, yet effective, implementation of an SVD technique, called *PureSVD*.

PureSVD

SVD is a well-established matrix factorization technique, which can be used to factorize the URM. The factorization of a generic matrix M of size $m \times n$ with SVD is the following:

$$M = U \cdot \Sigma \cdot V^T$$

where U is a $m \times m$ orthonormal matrix, which means that $U^T U = I$, V is a $n \times n$ orthonormal matrix, and Σ is a $m \times n$ diagonal matrix containing the $\min(m, n)$ singular values of M. A non-negative real number σ is a singular value for M if and only if there exist two vectors, $\vec{u}$ of length m and $\vec{v}$ of lenght n such that:

$$M\vec{v} = \sigma\vec{u}$$
$$M^T\vec{u} = \sigma\vec{v}$$

It is always possible to build the decomposition in such a way that the singular values in Σ are in a descending order, which gives a unique decomposition of M. Moreover, the number $r \leq \min(m, n)$ of non-null singular values of M corresponds to the rank of M.

The problem with conventional SVD is that it is undefined when knowledge about the matrix is incomplete, which is the case with missing ratings in recommender systems. Thus, different solutions were proposed to apply it to this field, such as imputation of missing ratings [56] or exclusive modeling of the observed ratings [36], using regularization to avoid overfitting.

PureSVD, instead, aims to factorize the URM by considering all the missing values as zeros, without any imputation or regularization [13]. After choosing the number of latent factors ($|L|$) as a parameter of the method, the user rating matrix R is then estimated by the following factorization:

$$\hat{R} = U \cdot \Sigma \cdot V^T$$

where U is a $|U| \times |L|$ orthonormal matrix, V is a $|I| \times |L|$ orthonormal matrix, and Σ is a $|L| \times |L|$ diagonal matrix containing the first $|L|$ singular values, sorted in a descending order.

Let's now define the matrix $P = U \cdot \Sigma$. Since V is an orthonormal matrix, we can rewrite P as:

$$\begin{aligned} P &= U \cdot \Sigma \cdot V^T V \\ &= R \cdot V \end{aligned}$$

This means that the missing ratings can now be estimated as:

$$\begin{aligned} \hat{R} &= U \cdot \Sigma \cdot V^T \\ &= P \cdot V^T \\ &= R \cdot V \cdot V^T \end{aligned}$$

Since V is a $|I| \times |L|$ matrix, $V \cdot V^T$ is a $|I| \times |I|$ matrix. Thus, $V \cdot V^T$ can be seen as an item-item similarity matrix S and the user rating matrix can be estimated as:

$$\hat{R} = R \cdot S$$

Similarly to neighborhood-based methods, seen in Section 2.2.1, we can add a top-k parameter in this method too, in order to keep, for each item, only the similarities of the k most similar items deriving from $V \cdot V^T$.

2.2.3 Graph-Based

In graph-based methods, data is represented as a bipartite graph, whose nodes are users and items and whose edges encode interactions between users and items [52]. Edges can be weighted and the graph can be undirected, if

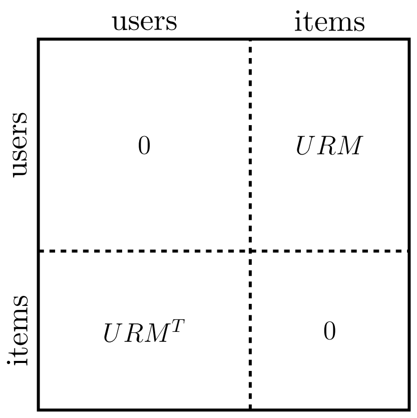


Figure 2.1: Adjacency matrix built from the URM.

going from a user to an item is no different than vice versa, or directed, with two different edges for the two directions, if they need different weights.

Before presenting graph-based recommendation algorithms, let's introduce the mathematical concepts used by them. The first is the adjacency matrix A, which is a matricial representation of the graph. Each node of the graph, being it a user or an item, has both a corresponding row and column in the matrix. So, the element at row i and column j in the matrix is:

$$A_{ij} = \begin{cases} 1 & \text{if exists an edge from } i \text{ to } j \\ 0 & \text{otherwise} \end{cases}$$

If the graph is undirected, the adjacency matrix would be symmetric, otherwise we cannot make this assumption. Moreover, if the edges of the graph have some weights, we need to include this information into A, by simply replacing the boolean values with the corresponding weights w_{ij}:

$$A'_{ij} = \begin{cases} w_{ij} & \text{if exists an edge from } i \text{ to } j \\ 0 & \text{otherwise} \end{cases}$$

Since the graph is built from the interactions, we can conveniently build a symmetric adjacency matrix directly from the URM and its transposed (see Figure 2.1). Each non-zero value of the URM is indeed the value of the weight between a user node and an item node.

Then, let's introduce the concept of *random walk*, which is the basis of the graph-based algorithms we are going to present. A random walk on a graph can be seen as a *Markov chain*, starting from a node and randomly moving to one of its neighbors at each time step [49]. A Markov chain is a stochastic process satisfying the Markov property, which tells that an action on a particular time step only depends on the current state, rather than on previous actions and states. A Markov chain is characterized by the so-called transition matrix, a square matrix of non-negative values representing the probability of reaching each node from every other node. The transition matrix P can be obtained by normalizing, row by row, the adjacency matrix A':

$$P_{ij} = \frac{A'_{ij}}{\sum_{k \in V} A'_{ik}}$$

where V is the set of all nodes and $i, j \in V$.

$\mathbf{P}^3_\alpha$

P^3_α is an algorithm based on the transition matrix P [12]. The idea is that the score of the recommendation of an item to a user is given by the probability of reaching the node corresponding to the item from the user node.

The model used by the algorithm is literally described by its name. First of all, we know that elevating the matrix P to a positive integer n we obtain a matrix P^n whose values are the probabilities of reaching the node referenced by the column, starting from the node referenced by the row, after n random walk steps. Since the objective is to recommend an item to a user and the graph is bipartite, n should be an odd number, so that P^n represents the probability of reaching an item node from a user node. In particular, the chosen n for this is algorithm is 3, since it has been shown that it outperforms other values [12], thus giving P^3. Moreover, to improve the original model performances, a real valued parameter α has been introduced. P_α is thus derived from the transition matrix P by elevating each positive element to the power of α:

$$P_{\alpha\ ij} = \left(\frac{A'_{ij}}{\sum_{k \in V} A'_{ik}} \right)^\alpha$$

The optimal value of the parameter α can be determined from a parameter optimization phase. Moreover, similarly to neighborhood-based methods (Section 2.2.1), we can add a top-k parameter in order to keep, for each item, only the similarities of the k most similar items deriving from P^3_α. So, the complete model is computed by elevating the P_α matrix to the power of 3, obtaining the P^3_α matrix, consequently used to make recommendations. Each recommendation of item i to user u has a score p^3_{ui} corresponding to the element at row u and column i of matrix P^3_α:

$$P^3_{\alpha\ ui} = \sum_{j \in V} \sum_{v \in V} P_{\alpha\ uj} \cdot P_{\alpha\ jv} \cdot P_{\alpha\ vi}$$

$\mathbf{RP}^3_\beta$

RP^3_β is an extension of the standard P^3_α algorithm proposed to solve the bias that P^3_α has towards popular items [49]. In order to cope with this problem, a re-ranking procedure based on item popularity is introduced. The RP^3_β score is obtained by re-weighting the original score p^3_{ui} given by P^3_α for user u and item i:

$$\widetilde{p}^3_{ui} = \frac{p^3_{ui}}{d^\beta_i}$$

where d^β_i is the number of incoming edges into the node corresponding to item i in the graph, also called indegree, elevated to the parameter of the model, β. This parameter is a positive real number used to regulate the influence of the indegrees in the re-ranking: higher values correspond to a higher popularity penalty. The value of β can be determined through an optimization phase. Moreover, it has to be noted that, if we put $\beta = 0$, the results of RP^3_β would coincide with the ones of P^3, since $d^0_i = 1$.

Also in this method it is possible to add a top-k parameter in order to keep, for each item, only the similarities of the k most similar items deriving from the model.

2.3 Content-based Filtering

Content-based recommender systems try to recommend items similar to the ones the target user has liked in the past [39]. Content-based filtering leverages a different type of information with respect to collaborative filtering. As the name says, these methods rely on contents to make predictions. In particular, when making recommendations for a target user u, they only need the interactions of the target users and the information about the item descriptions. They ignore other user interactions, not making use of collaborative information.

For example, if we consider a movie recommendation system, a user that watched movies like *Iron Man* or *Captain America* would very likely be suggested to watch *The Avengers*, since they share common actors, production members and the same genre. Instead, with a collaborative approach, the system could have also estimated high affinity with other popular movies like *Star Wars* or *Avatar*.

Since content-based methods rely only on item features, the first thing to do when building a content-based recommender system is to extract discriminative features from the data. Given that most of the content information is contained in describing texts, a common approach to the feature extraction problem consists in extracting representative keywords from the textual descriptions. Adopting a binary representation and combining these keywords with editorial features, such as actors, directors and so on, the result is the previously described data structure, the *item content matrix*.

The recommendation for a target user u is then computed on the basis of their previous ratings and on the neighborhoods of the rated items. In particular, the neighbors are defined by the similarities between items computed using item features. This is a substantial difference from collaborative filtering, where we use previous interactions between users and items.

Thus, the rating $\hat{R}_{ui}$ that user u would give to item i can be estimated as:

$$\hat{R}_{ui} = \frac{\sum_{j \in I} R_{uj} \cdot sim(i,j)}{\sum_{j \in I} |sim(i,j)|}$$

Of course, since this is a neighborhood-based method, all the parameters presented in Section 2.2.1 for collaborative filtering are used with content-based filtering as well.

2.4 The Cold-Start Problem

Collaborative filtering allows to achieve good results, in general. However, there are some cases in which collaborative information is not available, thus making collaborative filtering ineffective. Let's consider, for example, a movie platform such as Netflix. When a new movie or TV series is released, there is no collaborative information available for that item, since no one has ever interacted with it. Thus, a collaborative method would never recommend this new item to any user, since there is no information to base the recommendation on. This is called the *cold-start problem*. When appearing for the first time in the system, an item has no interactions and it is referred to as a *cold item*.

Various methods have been proposed in the literature to cope with this problem. One of them consists in exclusively using content-based filtering in order to recommend all items, since content information is available, instead. However, usually content-based methods are substantially outperformed when collaborative information is available. This is due to the available item features, which could be too few, noisy or in general they could not really represent what the user perceives when interacting with the items. Hence, content-based methods are usually of a lower quality than collaborative ones, requiring a lot of feature engineering to improve their performance.

Another common method is collaborative filtering with *side information*. This kind of method adds non-collaborative information into collaborative filtering techniques, in order to cope with the lack of collaborative information. This can be done, for example, by concatenating the columns of the ICM, containing content information about the items, to the rows of the URM and then applying a classical collaborative filtering algorithm, as done in [16]. A more elaborated technique, using machine learning, is SSLIM (Sparse Linear Methods with Side information [45]). It is based on SLIM (Sparse Linear Methods [44]), which is a collaborative filtering method that estimates the missing ratings in $\hat{R}$ as a product between the rating matrix R and an $|I| \times |I|$ matrix of *aggregation coefficients*, S (not to be confused with the similarity matrix). This matrix is learnt by minimizing the regularized objective function of the error between R and RS, where $diag(S) = 0$. SSLIM focuses on incorporating side information (i.e., item features) within SLIM in different ways, as described in [45]. For example, by learning an aggregation matrix S that simultaneously approximates both the rating matrix and the side information matrix, or by learning two different aggregation matrices, one for the ratings and one for side information, while minimizing their distance.

However, while these solutions try to add content information to collaborative techniques, in this work we focus on the opposite. Indeed, one more solution to the cold-start problem is the optimization of content-based methods by exploiting collaborative information. In particular, one common way to achieve this result is by using *feature weighting.*

2.5 Feature Weighting

In content-based filtering, the relevant information used to make recommendations is given by item features. In basic neighborhood-based methods, every feature is given the same weight when computing the similarity between two items. However, it is safe to assume that, when deciding, for example, what movies to watch next, a user would give different importance to different features. Genre could be a lot more discriminative than the executive producer or the costume designer. Moreover, if the user is a fan of a certain actor, they would probably give more credit to the feature corresponding to that actor.

So we can say that a user's judgement is based on a linear combination of the similarities between individual attributes of the items. This translates into the following formulation of the similarity between two items, i and j:

$$sim(i, j) = w \cdot f(i, j) = \sum_{c \in F} w_c \cdot f_c(i, j)$$

where w is the vector of feature weights and $f(i, j)$ is the vector of similarities between individual features of i and j. Vector w is unknown and feature weighting techniques aim to predict the best weights to improve the recommendation quality.

It should be noted that feature weighting is a generalization of *feature selection*, where, instead of selecting features, the algorithm weights them by some continuous value. An example of feature selection applied to recommender systems can be found in the preprocessing of the CiteULike dataset in [62], where TF-IDF (see Section 2.5.1) is used to weight the features and then select a certain number of the highest scoring ones. So, in general, analogously to feature selection [27], we can distinguish feature weighting methods in three categories:

- *Filtering* methods usually rely on methods derived from information retrieval, weighting features depending on some index computed from the data. Thus, they can always be applied with no particular assumptions. Examples of filtering methods are TF-IDF [40, 32] and BM25 [54].
- *Embedding* methods learn feature weights as a part of the model training phase. In particular, in the recommender systems field, examples of embedding methods are UFSM [20] and FBSM [58], which learn the item similarity in functions of the features directly from the interactions. They work with the assumption that the interactions data have a good quality. However, these methods strongly couple the interactions and the weighted features.
- *Wrapper* methods learn feature weights in two phases, by exploiting an already available model that has been previously trained. Examples of wrapper methods for recommender systems are CFW [15] and CFeCBF [17], that learn the weights from a previously built collaborative similarity matrix, HP^3 [7], that learns from graph models, and other methods such as the one described in [10], using XGBoost [11] in order to weight (and then select) features. As opposed to embedding methods, they work on the assumption that the model they learn from has a good quality.

We will now present some specific techniques, a couple of which rely on machine learning.

2.5.1 TF-IDF

TF-IDF is a *filtering* method derived from *information retrieval* and *text mining*. It stands for Term Frequency-Inverse Document Frequency and aims to weight words depending on how much they are frequent but also relevant in documents. In recommender systems we can apply it by considering features in items as words in documents, with the document collection represented by the ICM.

The resulting weights consist of two terms multiplied by each other. TF-IDF for a feature f in an item i can be computed as:

$$TF - IDF_{fi} = TF_{fi} \cdot IDF_f$$

TF

Term Frequency assumes that a word is more important the more it appears in a document. This assumption was made by Hans Peter Luhn [40] when proposing the first form of term weighting.

Similarly, in recommender systems, if we consider features as words and items as documents, we can say that a feature importance depends on its value (if it has one) with respect to the values of other features of the same item. In this case, since the ICM contains boolean values, TF for a feature f of an item i can be computed as:

$$TF_{fi} = \frac{ICM_{if}}{\sum_{c \in F} ICM_{ic}}$$

where ICM_{if} is the element at row i and column j of the ICM, which has value 1 if item i has feature f and 0 otherwise.

IDF

TF, however, may give too much importance to common words. For example, the term "the" in English is so common that TF would tend to emphasize documents that use it more frequently, ignoring other less-common words that are in general more meaningful in distinguishing relevant and non-relevant documents and terms.

Thus, another factor is incorporated that diminishes the weight of terms that appear frequently in the document set (common words) while increasing the weight of terms that rarely occur. Inverse Document Frequency (IDF) was proposed by Karen Spärck Jones [32] as a statistical interpretation of term-specificity, quantified as an inverse function of the number of documents in which the term occurs.

The analogy can be extended to item features in recommender system, where the IDF for a feature f can be computed as:

$$IDF_f = \log \frac{N}{\sum_{i \in I} ICM_{if}}$$

where N is the total number of features in the ICM, computed as:

$$N = \sum_{i \in I, f \in F} ICM_{if}$$

2.5.2 CFW

Collaborative boosted Feature Weighting (CFW) is a machine learning *wrapper* approach to feature weighting [15]. It consists in embedding collaborative information into a content-based model. By weighting item features, it aims to better represent feature importance from the user's point of view. In general, this method assumes that the collaborative filtering model achieves a much higher recommendation quality than the content-based one and is better able to capture the user's perspective [17].

The content-based similarity between two items i and j is computed as a weighted product between their feature vectors f_i and f_j:

$$sim(i,j) = f_i^T W f_j \tag{2.2}$$

Here, W is the feature weights matrix, of size $|F| \times |F|$. Its diagonal elements represent the importance of each individual feature, while the off-diagonal elements determine the correlation among different features. Thus, in order to reduce the number of parameters, W can be represented as the summation of two components:

$$W = D + V^T V \tag{2.3}$$

where D is an $|F| \times |F|$ diagonal matrix containing the individual feature weights and $V^T V$ is a low-rank approximation of the off-diagonal values, with V being an $l \times |F|$ matrix and l the number of latent factors.

Calling S^W the similarity matrix derived from the weighted features, CFW uses Stochastic Gradient Descent (SGD) to search for the best parameters in order to optimize the following objective function:

$$\underset{D,V}{argmin} \quad \left\|S^{CF} - S^W\right\|_F^2 + \lambda \left\|D\right\|_F^2 + \gamma \left\|V\right\|_F^2$$

where S^{CF} is any item-item collaborative similarity, λ and γ are the regularization coefficients and $||\cdot||_F^2$ is the Frobenius norm.

A similar approach has been proposed in the literature with the name *Collaborative-Filtering enriched Content-Based Filtering* (CFeCBF) [17]. This approach derives from the fact that CFW with the V component is outperformed by CFW with only the D component [15]. Indeed, the main difference from CFW is that the weighting matrix is a diagonal matrix D, with no correlation between different features. The similarity of two items i and j is thus computed as:

$$sim(i,j) = \frac{f_i^T D f_j}{||f_i||_F^2 ||f_j||_F^2}$$

also adding normalization. Moreover, the optimization problem to solve, in order to obtain the weighting matrix, becomes the following:

$$\underset{D}{argmin} \quad \left\|S^{CF} - S^D\right\|_F^2 + \lambda \left\|D\right\|_F^2 + \gamma \left\|D\right\|$$

Thus, the CFW and CFeCBF procedures consist of two steps. First, find the optimal parameters for the collaborative algorithm, and second, learn the optimal feature weights that better approximate the item-item collaborative similarity obtained before. The weighting model can either use the different feature correlations (and hence latent factors, CFW D+V) or use only the individual feature weights (CFW with only D, CFeCBF).

2.5.3 FBSM

Factorized Bilinear Similarity Model is a machine learning *embedding* method for feature weighting. Proposed in [58] as an improvement of *User-specific Feature-based Similarity Models* (UFSM) [20], it aims to discover correlations among item features.

The FBSM model and the CFW model share the same similarity function, computed as in equation (2.2), and the same weighting matrix expressed as in equation (2.3). The main difference is in the learning procedure. Indeed, FBSM makes use of SGD with Bayesian Personalized Ranking loss in order to learn the weights directly from the user-item interactions, instead of learning from a previously computed model.

However, interactions have less support than a similarity model, since the latter is an aggregation of multiple interactions data. A similarity model also reduces noise by removing similarities with low support, according to parameters such as top-k or shrink, shown in Section 2.2.1. Moreover, FBSM is an embedding method, so it learns feature weights along with the recommendation model itself. This largely increases the training phase complexity and strongly couples features and interactions, making the model more susceptible to noise. Due to these drawbacks and the experiments presented in [15], which show that CFW outperforms FBSM in a similar scenario to ours, FBSM is not used in our work.

2.6 Evaluation

Recommender systems can mainly be evaluated in three different ways, *offline*, with *user studies* and *online* [57]. Offline evaluation is performed on existing data sets by modeling the user behavior on partitions of this data and evaluating some performance measures such as prediction accuracy. User studies are extensive simulations of an online platform in which a small set of users is asked to perform some tasks on the system and then answer to questions or give feedback on the proposed recommendations. Online evaluation can be conducted through experiments on an already deployed system, for example using A/B testing.

In this work we only use offline evaluation. In particular, we focus on a *top-n recommendation task*, which consists in recommending an ordered list of n items to each user. Therefore we evaluate how much the recommended items are good for the target user, in the sense that the user would interact with the items. We evaluate the correctness of each recommendation, the order quality of the entire list and other useful metrics not related with recommendation accuracy. Here are presented the main metrics we use while evaluating our model.

	Recommended	Not recommended
Interacted	True Positive (TP)	False Negative (FN)
Not interacted	False Positive (FP)	True Negative (TN)

Table 2.1: Confusion matrix for recommendations.

2.6.1 Classification Accuracy Metrics

Classification accuracy metrics measure the frequency with which a recommender system makes correct recommendations to the users [29].

For example, let's say a user on a movie platform liked the movies *Iron Man*, *Captain America* and *The Avengers*, but did not have any interaction with the movie *Thor*. When evaluating a model, data is partitioned in order to train the model on some data and evaluate it on an unseen part of the data. In our case, let's say that *Iron Man* and *Captain America* are included in the training data, while *The Avengers* and *Thor* are included in the data used for evaluation. If the model we are evaluating recommends both *The Avengers* and *Thor* to the target user, we can say that *The Avengers* is a correct recommendation, since the user actually interacted with it, while *Thor* is an incorrect one, since the user did not interacted with it and we cannot assume anything about an unknown value.

Thus, we can define the *confusion matrix*, shown in Table 2.1, which tells how many correct and incorrect recommendation a system makes. In particular, in the confusion matrix are represented the following terms:

- *True Positive (TP)*: items, relevant to the user, that the system correctly recommended.
- *False Positive (FP)*: non-relevant items that the system incorrectly recommended to the target user.
- *True Negative (TN)*: irrelevant items that the system did not recommend.
- *False Negative (FN)*: item, relevant to the user, that the system did not recommended, making a mistake.

Both classification accuracy metrics we are going to present, *precision* and *recall*, make use of these terms.

Precision

Precision measures how many items the recommender system correctly recommend with respect to all the recommendations. Referring to the confusion matrix at Table 2.1, precision for each user u can be computed as:

$$Precision = \frac{\#TP}{\#TP + \#FP}$$

where $\#TP$ and $\#FP$ respectively indicate the number of true positive and false positive items for the target user.

In particular, in recommender system, the *Precision@k* metric is used, which represents the precision on a cut-off list of k recommended items, which are usually the k items with the highest recommendation score for the system. It is computed as [31]:

$$Precision@k = \frac{\#TP_k}{k}$$

where $\#TP_k$ is the number of correctly recommended items among the k items in the recommendation.

Recall

Recall measures how many actually positive items (the ones the target user interacted with) are recommended by the system with respect to all the positive items. Again, referring to Table 2.1, recall for user u can be computed as:

$$Recall = \frac{\#TP}{\#TP + \#FN}$$

where $\#FN$ is the number of false negative items for the target user.

In recommender system, the metric of interest is the *Recall@k*, which represents the recall on a cut-off list of k recommended items, instead of the entire item set. It is computed as [31]:

$$Recall@k = \frac{\#TP_k}{\text{Number of relevant items}}$$

2.6.2 Ranking Metrics

As already stated before, we are dealing with a top-n recommendation task. This means that we would like not only to have correct recommendations in the resulting list, but also for this list to be ordered according to the target user's taste, possibly having more likely correct recommendations at the beginning of the list.

Ranking measures are useful precisely in this scenario. They evaluate how good a recommendation with k items is with respect to how well it is ordered for the target user's taste. In particular, we are going to present metrics that weight the evaluation of each item of the recommendation by their position in the list.

Mean Average Precision

Mean Average Precision (MAP), computed for a recommendation list of k items (MAP@k), is the mean of the Average Precision computed for that recommendation (AP@k) [41]. Average precision for a user u is the average of the *Precision@k* computed for every slice of the recommendation list of size from 1 to k:

$$AP_u@k = \frac{\sum_{i=1}^{k} Precision@i \cdot Rel(u,i)}{\text{Number of relevant items}}$$

where $Rel(u,i)$ is 1 if item i at rank k is relevant for the target user u, 0 otherwise. From this, MAP@k can be computed as:

$$MAP@k = \frac{\sum_{u \in U} AP_u@k}{|U|}$$

Normalized Discounted Cumulative Gain

Normalized Discounted Cumulative Gain (NDCG) is particularly useful when evaluating models for platforms where the users may be presented with long recommendation lists. The reason is that NDCG has a slow decay of positional discount, more precisely logarithmic [2].

Let's define first the Discounted Cumulative Gain (DCG) for a user on a recommendation list of k items [2]:

$$DCG@k = \frac{1}{|U|} \sum_{u \in U} \sum_{i=1}^{k} \frac{2^{rel_{ui}} - 1}{\log_2(i+1)}$$

where rel_{ui} is the relevance of item i for the target user, which may be the interaction between user u and item i in the evaluation partition of the data set. Now, let's define the Ideal DCG (IDCG), which is the DCG of the best possible ordering of the recommendation the user could receive:

$$IDCG = \frac{1}{|U|} \sum_{u \in U} \sum_{i=1}^{n_{ur}} \frac{2^{rel_{ui}} - 1}{\log_2(i+1)}$$

where n_{ur} is the number of relevant items for the target user u.

Then, the NDCG@k can be computed as the normalization of the DCG@k with respect to the IDCG:

$$NDCG@k = \frac{DCG@k}{IDCG}$$

2.6.3 Beyond Accuracy Metrics

Good recommendation accuracy does not completely define how much a recommender system can satisfy the users. For example, a system could reach high accuracy by recommending items that are easy to predict, such as popular items, a task that even a popularity metric could do [29].

Therefore, other metrics that go beyond accuracy are needed to understand if a recommender system is able to make *useful* recommendations with a *good quality* [22]. In particular, the presented metrics focus on two aspects, *coverage* and *diversity* of the recommendations.

Coverage is a measure of the domain of items over which the system can make a recommendation [29]. A larger coverage means that the recommender system is able to recommend more of the available items.

Diversity measures, in general, how much is the variety of items that a system is able to recommend. Having diversity in a recommender system is useful because presenting different choices can often increase the chance that the user would be interested by one of them [2]. Indeed, if the target user dislikes a recommendation from a list of similar items, it is likely that they would dislike all of them. Thus, in some sense, diversity may be seen as the opposite of similarity [57].

Let's now present the specific metrics used in our work.

Item Coverage

There are mainly two possible coverage metrics. *Prediction coverage* measures the percentage of items for which the system is able to make a recommendation. This highly depends on the chosen technique and on its input [22]. *Catalog coverage*, instead, measures the percentage of available items that the system actually recommends to the users [22].

In our work we only consider catalog coverage and we will refer to it as *item coverage* (IC). Item coverage can be computed as the fraction of items that are recommended to at least one user [2]:

$$IC = \frac{|\cup_u^U \hat{r}_u|}{|I|}$$

where $\hat{r}_u$ is the recommendation list computed by the system for user u.

Gini Diversity

Gini Diversity (GD) is a diversity metric inspired by the Gini coefficient and computed as [1]:

$$GD = 2\sum_{i=1}^{|I|}\left[\left(\frac{|I|+1-i}{|I|+1}\right)\times\left(\frac{rec(i)}{total}\right)\right]$$

where, given a list of the number of times each item is recommended, sorted in ascending order, $rec(i)$ is the i-th element of this list and $total$ is the total number of top-N recommendations made across all users (i.e., $total = N \times |U|$). Gini diversity is always in the range $[0, 1]$, however, the scale is reversed with respect to the Gini coefficient [1], so that smaller values represent lower diversity, while larger values represent higher diversity.

Mean Inter-List Diversity

Mean Inter-List Diversity (MIL) is a diversity metric measuring the uniqueness of different users' recommendation lists [64]. The original name given to this metric was *personalization*, but we believe that it does not characterize the metric well, since the highest MIL values come from non-personalized random recommenders.

Given two users u and v the diversity between their recommendation lists of k items can be computed as:

$$h_{uv}(k) = 1 - \frac{q_{uv}(k)}{k}$$

where $q_{uv}(k)$ is the number of common items in the first k places of both lists. Therefore, $h_{uv}(k) = 0$ means that the two users have the same items in their top-k lists, while $h_{uv}(k) = 1$ indicates two completely different lists. From this, mean inter-list diversity can be computed by averaging $h_{uv}(k)$ over all pairs of users:

$$MIL_k = \frac{\sum_{u,v \in U, u \neq v} h_{uv}(k)}{|U| \times |U| - |U|}$$

2.7 Hyperparameter Tuning

Hyperparameter tuning is a procedure that searches for the hyperparameters giving the best performing model with respect to some chosen evaluation on unseen data. There can be different ways to search for the best hyperparameters. One option is to perform a *grid search*, by subsequently evaluating all the combinations of hyperparameters in certain ranges. However, because of the often large number of hyperparameters in a method and because of their wide ranges, grid search is almost never feasible for such a task. Another solution can thus be *random search*, which randomly sets hyperparameter values in the given ranges. Despite its efficiency and the fact that it often outperforms grid search, there are other methods that have been proven to work well in recommender systems. For example, the one we use in our work, *Bayesian optimization*, has been successfully employed in the literature [17] and in various RecSys challenges [4, 14].

Bayesian optimization probabilistic model that aims to optimize an unknown objective function from which we can obtain data points. In our case, the function we want to optimize is the accuracy of the recommender system for which we are performing hyperparameter tuning. Starting from a prior distribution, at each iteration the prior is updated with data resulting from the unknown function, forming a posterior distribution. The posterior is then used to construct an *acquisition function*, used to determine the next evaluation point, which is the next set of hyperparameters. This procedure is repeated a certain number of time and it is initialized by randomly selecting the first hyperparameter sets instead of relying on the acquisition function. Different probabilistic models can be used, in our work we use a *Gaussian process.*

2.8 Quadratic Unconstrained Binary Optimization

In this section, we are going to present the fundamentals of the mathematic model on which our work is built on. In particular, we start by presenting the Quadratic Unconstrained Binary Optimization problem, which is at the core of our model. Then, we explore some classical algorithms used to solve it.

2.8.1 QUBO Problems

Quadratic Unconstrained Binary Optimization (QUBO) is a class of NP-hard problems that has been gaining recognition to embrace a remarkable range of applications in combinatorial optimization [35]. For example, the use of this model for representing and solving optimization problems on graphs, resource allocation problems, set partitioning problems, assignment problems and many others has been reported in the literature [35].

The QUBO model is defined as:

$$\min \quad y = x^T Q x$$

where x is a vector of n binary decision variables and Q is a square $n \times n$ matrix of constants.

While the diagonal elements of Q are the coefficients of the linear terms of the objective function, the off-diagonal elements are the coefficients of the quadratic part, consisting of multiplicative terms between different variables. To better understand this, let's unroll the matrix notation into a scalar one:

$$y = \sum_{i=0}^{n} \sum_{j=0}^{n} x_i x_j Q_{ij} \tag{2.4}$$

So, when $j = i$ we obtain the linear term $x_i^2 Q_{ii}$, while when $j \neq i$ we get the quadratic term $x_i x_j Q_{ij}$. Now, since the variables are binary, we can represent the linear terms as $x_i Q_{ii}$, without elevating at the power of 2.

Typically, the Q matrix is symmetric or in upper triangular form. If it is not, we can always convert Q into one of these forms without loss of generality:

- symmetric:

$$q'_{ij} = \frac{q_{ij} + q_{ji}}{2} \qquad \forall_{i,j} \quad i \neq j$$

- upper triangular:

$$\begin{aligned} q'_{ij} &= q_{ij} + q_{ji} && \forall_{i,j} \quad j > i \\ q'_{ij} &= 0 && \forall_{i,j} \quad j < i \end{aligned}$$

To sum up, the QUBO model is a formulation of a quadratic optimization problem with binary variables and no constraint. Here we present a numerical example of a generic QUBO problem, showing the starting objective function and the Q matrix in the upper triangular form:

$$y = 3x_1 - 4x_2 + 7x_3 + 2x_1x_2 + 5x_1x_3 - 6x_2x_3$$

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad Q = \begin{bmatrix} 3 & 2 & 5 \\ 0 & -4 & -6 \\ 0 & 0 & 7 \end{bmatrix}$$

Many optimization problems can naturally be formulated as instances of the QUBO model. Moreover, since QUBO problems are NP-hard [48], we know, from computational theory, that every NP problem can be reduced in polynomial time to the QUBO formulation. As explained by Glover in [26], in order to reformulate a constrained optimization problem into the QUBO format, the constraints are converted into penalties embedded into the objective function. For example, let's take a quadratic programming problem, which is an optimization problem with a quadratic objective function and linear constraints [46]. In particular, consider a general binary quadratic optimization problem of the form:

$$\begin{aligned} \text{min} \quad & y = x^T C x \\ \text{s.t.} \quad & Ax = b \\ & x \text{ binary} \end{aligned}$$

These constrained quadratic optimization models are converted into equivalent unconstrained QUBO models by converting the constraint $Ax = b$ into a penalty added to the objective function [26]:

$$y = x^T C x + p(Ax - b)^T (Ax - b)$$

where p is a positive scalar that weights the penalty. Doing the math we obtain:

$$y = x^T Cx + p(x^T A^T Ax - 2b^T Ax + b^T b)$$

Considering that $-2b^T A$ is a linear constraint of binary variables, we can add it directly to the diagonal of the matrix resulting from $A^T A$, thus reformulating $p(x^T A^T Ax - 2b^T Ax)$ as $x^T Dx$. Moreover, we can define a constant $c = pb^T b$, thus obtaining the following formula:

$$\begin{aligned} y &= x^T Cx + x^T Dx + c \\ &= x^T Qx + c \end{aligned}$$

where the matrix Q results from the sum of C and D. Dropping the additive constant c, since it does not impact the optimization, we obtain the unconstrained version of the starting constrained problem, which exactly correspond to a QUBO model:

$$\begin{aligned} \min \quad & y = x^T Qx \\ & x \text{ binary} \end{aligned}$$

When a problem is expressed as a QUBO model, any algorithm capable of solving a QUBO model can be used to solve the given problem. It is remarkable how, in many cases, the solution of a problem obtained from its QUBO formulation with a generic QUBO solution method can rival the solution obtained from a method specialized to exploit the problem domain [35].

2.8.2 Solving QUBO Problems with Classical Algorithms

While a few special cases of QUBO are polynomially solvable [5, 47], QUBO is in general an NP-hard problem [48]. This means that, except for small problems, heristics are required to produce good solutions in a reasonable amount of time [35]. Therefore, while exact solution methods exist, we are not going to focus on them, since they are feasible only for problems with a small number of variables.

Here we present some heuristics, used to solve optimization problems, that fit particularly well for solving QUBO models.

Tabu Search

Tabu search is a metaheuristic algorithm proposed by Glover in 1986 [23] and later formalized in [24, 25]. It employs local search to iteratively explore the neighborhood of each potential solution, searching for an improved solution. However, local search methods often become stuck in local minima or plateaus. In order to avoid this problem, tabu search also accepts worsening moves, when improving ones are not available. In addition, it prohibits (from this the name "tabu") moving to solutions previously visited in the last n iterations, where n is a parameter of the algorithm. This adjustment is useful in order to avoid cycling through already visited solutions.

Simulated Annealing

Simulated Annealing (SA) is another metaheuristic algorithm for approximating the global optimum of a given function, often used when the search space is discrete. A first version of the technique was proposed by Pincus in 1970 [50] and later refined by Kirkpatrick et al. [34], who gave it the name we use today, inspired by the analogy with annealing in metallurgy. Both Pincus and Kirkpatrick based their work on the Metropolis algorithm [43].

Simulated Annealing is similar to tabu search, in the sense that also SA explores the search space with local search methods and accepts worsening moves. However, the way in which SA takes a step in the search space is very different.

First of all, let's define s as the current state of the system, characterized by an assignment of values for the variables x of the optimization problem we are solving. Then, let's define the energy function $E(\cdot)$, which computes the energy of the input state by evaluating the objective function in the corresponding assignment of x. The probability of moving from state s, with energy $e = E(s)$, to a neighbor candidate state s', with energy $e' = E(s')$, can be computed as:

$$P(e, e', T)$$

where P is a function of both the energies of the current and candidate states and of a time-varying parameter T, called *temperature*. The temperature initially starts as a positive value and decreases at each step following some *annealing schedule*, stopping at $T = 0$.

Moreover, P is defined in such a way that it corresponds to a positive value even if e' is larger than e (worse solution). In this case, P tends to zero when T tends to zero. Indeed, the more the temperature decreases, the less the algorithm is inclined towards exploring worse solutions, increasing the exploitation of local better solutions. When $T = 0$, the algorithm behaves as a greedy search, moving only to improving solutions.

Simulated Quantum Annealing

Simulated Quantum Annealing (SQA) is the classical implementation of quantum annealing. We will refer to it as SQA in order to not confuse it with the physical realization of quantum annealing, that we are going to discuss in Section 2.9.2. Simulated quantum annealing was proposed in the current form by Kadowaki and Nishimori in 1998, as a modification of simulated annealing [33]. In particular, instead of using thermal fluctuations to move between states, they introduced quantum fluctuations, or *quantum tunneling*, that cause transitions between states.

We are not going to dive into details of this metaheuristic, since quantum annealing is presented in Section 2.9.2, and SQA is just a classical version of quantum annealing simulated with quantum Monte Carlo or other stochastic techniques. However, we can say that the advantage brought by quantum annealing (and thus SQA), is that quantum tunneling is able to move candidate solutions out of shallow local minima with less difficulty than simulated annealing. Indeed, the results in [33] showed that SQA performed better than SA. Instead, the main difference between SQA and a physical quantum annealer is that, in the former, quantum tunneling has to be classically simulated, while in the latter it is a phenomenon that naturally occurs because of the quantum behaviors characterizing the device.

2.9 Quantum Computing

Quantum computing is the use of quantum mechanics phenomena in a computational model. The idea of quantum computing caught on with Richard Feynman in 1982 [21], when he observed that certain quantum mechanical effects could not be simulated efficiently on classical computers. This observation led to the speculation that this problem would have been solved by building a computing device capable of harnessing quantum mechanical effects [53]. The idea of quantum computation gained particular traction when, in 1994, Peter Shor proposed a quantum algorithm able to factorize integers in polynomial time [59, 60]. If this was not enough to increase the interest in the quantum computing field, Lov Grover proposed, two years later, a quantum algorithm used to search for an element in an unordered list, obtaining a polynomial speedup over classical computing. Since those years, quantum computing has been a particularly active area, with many studies and technologies developed in the field [28].

Here we explain some fundamental concepts, before presenting specific quantum computing models. Such as classical computing relies on bits, quantum computing relies on the analogous *quantum bits*, or *qubits*. As their classical counterparts, qubits can be in states 0 and 1, but they have more properties than classical bits. Indeed, qubits are quantum objects capable of exploiting quantum mechanics behaviors such as *superposition* and *entanglement*.

Superposition is the physical property of a qubit of being in both states, 0 and 1, at the same time. Thus, a qubit in superposition cannot be represented as a classical state. Only when a qubit undergoes *measurement* it loses its superposition and its state *collapses* to 0 or 1 with a certain probability, depending on its superposition state.

Entanglement, instead, is a phenomenon observed between two or more different qubits. When two qubits are entangled, their states cannot be described singularly, but they only make sense if observed together. Indeed, the two qubits are bound in such a way that measuring one of them would make the other one collapse into a classical state as well. This is a phenomenon that has no correspondence in classical computing and can only be simulated with an exponential overhead, since it is necessary to take into account the distribution of all the possible states. Moreover, it is key to some interesting applications, such as quantum teleportation.

There are different models of quantum computing. The two models we are going to present are *gate-based* quantum computing and *quantum annealing*, also called *adiabatic* quantum computing. We are going to focus more on quantum annealing, since it is the approach used in our work.

2.9.1 Gate-Based Quantum Computing

Gate-based quantum computing is a quantum computing model presenting some analogies with classical computing. In this model, *quantum gates* are applied to qubits in order to transform their states. They are applied in an imperative way, similarly to instructions in a classical computer. However, because of quantum physics restrictions, all quantum state trasformations, thus quantum gates, have to be reversible [53]. At the end of a quantum computation, qubits are measured, in order to obtain the results of the computation.

In this model, qubits are represented as vectors in a Hilbert space, which is a generalization of the Euclidean space in any finite or infinite number of dimensions. In particular, qubits having a state 0 and 1 can be mathematically represented respectively as:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

using the Dirac notation. However, qubits can also be in a superposition, which means that they are both in state 0 and 1 and is mathematically expressed as:

$$\begin{aligned} |\phi\rangle &= \alpha\,|0\rangle + \beta\,|1\rangle \\ &= \alpha \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \end{aligned}$$

where α and β are, in general, two complex numbers such that $||\alpha||^2 + ||\beta||^2 = 1$. Their norms, $||\alpha||^2$ and $||\beta||^2$, can be seen as the probabilities of the qubit $|\phi\rangle$ to collapse to state $|0\rangle$ or $|1\rangle$, respectively.

As stated before, quantum gates are applied to qubits to transform them and make computations. The different gates can be mathematically represented by matrices. For example, the identity and bit flip (or negation)

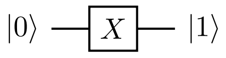


Figure 2.2: Quantum circuit of a basic bit flip (negation) operation on a qubit starting in state $|0\rangle$

operators can be expressed respectively as:

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

They can be applied both to a $|0\rangle$ or $|1\rangle$ qubit or to a qubit in a superposition, as shown here for the bit flip operator:

$$X|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle$$

$$X|\phi\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \beta \\ \alpha \end{pmatrix} = |\neg\phi\rangle$$

A graphical representation of a quantum gate applied to a qubit can be seen in Figure 2.2. What is represented there is a *quantum circuit*, a sequence of quantum gates applied to one or more qubits. The one presented in Figure 2.2 is fairly simple, but quantum circuits can be built with a sequence of operations on different qubits, analogously to instructions in a classical computer.

There exist different quantum gates, also working on more than one qubit. However, since gate-based quantum computing is not the focus of our work, we are going to present only two more of them.

The *Hadamard gate*, H, is a quantum gate that transforms a qubit in a state, $|0\rangle$ or $|1\rangle$, into a qubit in a perfectly balanced superposition, having the same probability of collapsing to 0 or 1. It can be mathematically represented by the following matrix:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

When applied to a $|0\rangle$ and $|1\rangle$ qubits we obtain respectively $\left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right)$ and $\left(\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}\right)$, which obey to $||\alpha||^2 + ||\beta||^2 = 1$. In particular, we can see that $||\alpha||^2 = ||\beta||^2 = \frac{1}{2}$, which means that these qubits have the exact same probability of collapsing to 0 or 1. The negative number at the bottom right corner of the H matrix is necessary to make the operation reversible. Indeed, if a qubit coming from a Hadamard gate has value $\left(\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}\right)$, we know for sure that it was in state $|1\rangle$ before applying the gate.

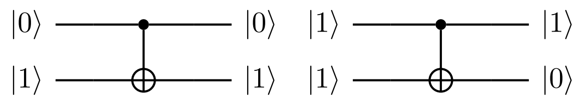


Figure 2.3: Quantum circuit showing how the CNOT gate works, applied to $|01\rangle$ and $|11\rangle$

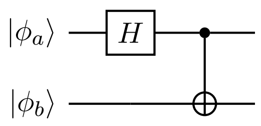


Figure 2.4: Quantum circuit representing the entanglement between two different qubits, using the Hadamard gate and the CNOT gate.

Another fairly important quantum gate is the *Controlled-NOT gate*, or CNOT. The CNOT is a gate applied to two qubits, instead of one. Its effect is that of flipping the least significant qubit only if the most significant qubit, called the *control qubit*, is $|1\rangle$. Explaining the mathematical formulation of the CNOT gate is out of our scope. However, two quantum circuits showing the CNOT applied to two different couples of qubits are shown in Figure 2.3.

The Hadamard and the CNOT gates are also extremely important because they can be used to achieve quantum entanglement between two different qubits. Again, the explanation for this phenomenon is out of our scope, but the circuit used to entagle two qubits is shown in Figure 2.4.

2.9.2 Quantum Annealing

Quantum annealing, or adiabatic quantum computing, is another quantum computing paradigm. Instead of using gates in order to transform qubits, it exploits the natural tendency of every physical system to evolve to its minimum energy state. Thus, quantum annealing is used to search for optimal or good solutions to NP-hard optimization problems.

To better understand how quantum annealing can achieve this, let's start by defining the underlying model of the physical system we are going to consider. In a physical realization of a superconducting quantum annealer, qubits are quantum objects that define the physical system on which the quantum annealing procedure is going to run. As already stated before, qubits can be in a superposition of the 0 and 1 states. However, at the end of the annealing process they all collapse to a classical state. What determines how the qubits' states evolve and collapse is the way they are influenced and correlated each other with the biases and the coupling strengths applied to them.

A bias is a magnetic field applied to a qubit in order to tilt its probability of ending up in state 0 or 1. This way, instead of having the same probability of collapsing to 0 or 1, the qubit collapses to the lower energy state determined by the applied bias.

A coupler, instead, is a physical device that can make two qubits tend to the same or the opposite states. The coupling strength determines how much two coupled qubits are correlated. Coupling relies on the quantum phenomenon of entanglement. When two qubits are entangled, they can be thought as a single entity with four different states, instead of two. The energies of these states are determined by the biases applied to the two qubits and by their coupling strength.

This observation can be extended to an entire network of superconducting qubits. Indeed, the energy landscape of such a system is determined by the biases and coupling strengths of all the qubits. Moreover, the energy of a particular state of the system is defined by its *Hamiltonian*. A Hamiltonian is a mathematical description of a physical system in terms of its energies[1]. In particular, the Hamiltonian for a physical quantum annealer is the following:

$$\mathcal{H} = \underbrace{-\frac{A(s)}{2}\left(\sum_i \hat{\sigma}_x^{(i)}\right)}_{\text{Initial Hamiltonian}} + \underbrace{\frac{B(s)}{2}\left(\sum_i h_i \hat{\sigma}_z^{(i)} + \sum_{i>j} J_{ij}\hat{\sigma}_z^{(i)}\hat{\sigma}_z^{(j)}\right)}_{\text{Final Hamiltonian}} \tag{2.5}$$

where $\hat{\sigma}_{x,z}^{(i)}$ are Pauli matrices operating on qubit q_i, h_i are qubit biases and J_{ij} are the coupling strengths between qubits q_i and q_j. The Hamiltonian changes during the annealing process depending on some annealing schedule, controlled by the parameters $A(s)$ and $B(s)$[2]. In particular, the annealing schedule starts at time $t = 0$ with the anneal fraction $s = 0$ and $A(0) >> B(0)$ and goes up to time t_f, when it stops at $s = 1$ and $A(1) << B(1)$. Thus, it controls how much the two terms of the Hamiltonian influence the annealing procedure.

The first term of equation (2.5) is called *inital* or *tunneling* Hamiltonian. Its lowest-energy state is the one where all qubits are in a superposition state. During annealing, the second term, called *final* or *problem* Hamiltonian is introduced. Its lowest-energy state is determined by the biases and the coupling strengths, which are set by the user. At the end of the quantum annealing process, when all the qubits are in a classical state, only the problem Hamiltonian describes the energy of the system. This means that, if we are able to map an optimization problem as the final Hamiltonian, the annealing would result in a solution to the optimization problem.

It should be noted that the problem Hamiltion is expressed as an Ising model[3]. In this model, variables s_i are spin up and spin down, correspond-

[1]https://docs.dwavesys.com/docs/latest/c_gs_2.html
[2]https://docs.dwavesys.com/docs/latest/c_qpu_0.html
[3]https://docs.dwavesys.com/docs/latest/c_gs_3.html

ing to states +1 and -1. The objective function of the Ising model is the following:

$$E_{ising}(s) = \sum_{i=1}^{N} h_i s_i + \sum_{i=1}^{N} \sum_{j=i+1}^{N} J_{ij} s_i s_j$$

where N is the number of variables s_i, h_i corresponds to the qubits' biases and J_{ij} to the coupling strengths. This equation is used to describe the Hamiltonian of the final classical state resulting from the annealing.

Therefore, mapping an optimization problem as an Ising model would enable us to search for solutions of that problem, as stated before. In particular, the conversion between an Ising model and a QUBO model, presented in Section 2.8.1, is straightforward. Indeed, the discrete variables s of an Ising model are related to the discrete variables q of a QUBO model as:

$$s = 2q - 1$$

Moreover, the biases h can be directly mapped into the diagonal of the Q matrix, while the coupling strengths in J can be reported as the off-diagonal elements of Q. Thus, since the QUBO formulation is equivalent to the Ising model, it can be used to describe the problem Hamiltonian. Then, quantum annealing can be used to search for the minimum energy of this Hamiltonian, which corresponds to the solution of the given problem expressed as a QUBO model.

The main advantage of quantum annealing over simulated annealing is that the former can rely on quantum tunneling. Quantum tunneling is a phenomenon that allows quantum annealing to penetrate high barriers in the energy landscape, without any increase in energy, differently from SA, which has to climb over these barriers to escape local minima [18]. Moreover, a physical realization of a quantum annealer guarantees that the involved quantum phenomena happen naturally because of quantum mechanics, without the need of simulating them, like in simulated quantum annealing. Finding a good solution to a QUBO problem is a matter of microseconds for quantum annealing, thanks to the expressed quantum phenomena.

The Annealer we used: D-Wave 2000Q

D-Wave Systems is a canadian company that has been studying and building quantum annealers for years. The annealer we use in our work is the last one available for cloud access, the D-Wave 2000Q. This quantum annealer has 2048 qubits, laid out on a *Quantum Processing Unit* (QPU) with a *Chimera* topology, shown in Figure 2.5.

The Chimera graph is composed of a 16 $\times$ 16 lattice of Chimera unit cells, which are bipartite graphs of 8 qubits Figure 2.6. Each qubit is also connected by couplers with qubits on the same row or column of other unit cells. This creates a sparse topology of connected qubits, which means that

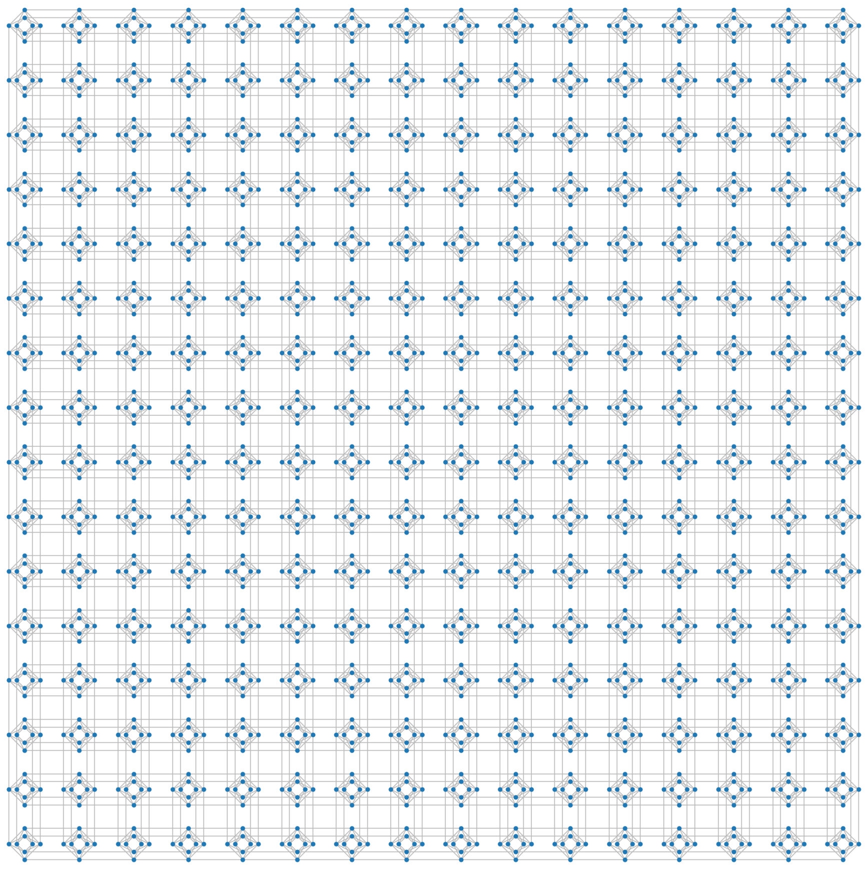
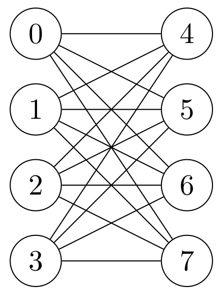


Figure 2.5: Chimera graph showing the arrangement of the qubits in D-Wave 2000Q.

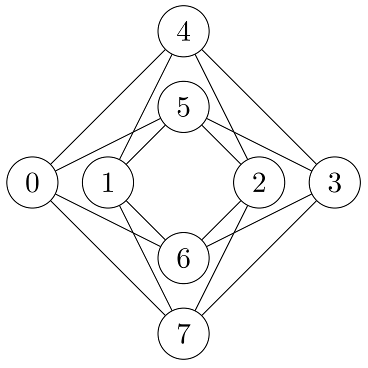
to represent a single logical qubit we may need more than one physical qubit, depending on how many connections the logical qubit has. Thus, in order to map the graph given by the QUBO model, where nodes are the problem variables and edges represent the coupling strengths, to the Chimera graph of the QPU, a procedure called *minor embedding* is needed. Although minor embedding is a NP-hard problem itself, there exist heuristics able to achieve good results with fair computational complexity [9]. We should point out that the maximum number of nodes, in a fully connected graph, that are embeddable on the QPU is 65.

Therefore, the complete workflow when solving a problem with quantum annealing on a D-Wave QPU is the following [42]:

1. represent the given problem in a QUBO or Ising formulation;



(a) Chimera unit cell in column (bipartite) form.



(b) Chimera unit cell in cross form.

Figure 2.6: Chimera unit cell in its two common renderings, with numbers representing qubits in the cell.

2. if the problem comprises too many variables, decompose it in order to solve each piece of the problem separately;
3. minor-embed the input problem into the QPU graph through a minor embedding heuristic;
4. query the QPU with the minor-embedded problem, which means to physically set biases and coupling strengths on the device and run the quantum annealing procedure;
5. retrieve the post-processed results.

The QPU is accessible and can be queried through D-Wave's cloud service, Leap[4].

Is D-Wave's QPU really quantum?

Many studies have been conducted and published in the literature about D-Wave's QPUs. In particular, a recurring question about this quantum annealer is if it really leverages quantum phenomena. This question has been solved by Boixo et al. [8] and by Lanting et al. [38] in 2014. The first paper shows how the behavior of D-Wave's physical quantum annealing is consistent with the expected one, in the form of SQA, confirming that D-Wave's QPU is indeed an actual implementation of quantum annealing. Moreover, the second paper presents experimental evidences for the presence of quantum entanglement in D-Wave's QPU.

An additional question regards the computational advantage given by D-Wave's quantum annealing devices over classical computing. In [18], Denchev et al. studied the computational performance of one of D-Wave's

[4]https://cloud.dwavesys.com/leap/

QPUs against classical methods, such as simulated annealing or simulated quantum annealing. The results show that, in certain settings, although there is no exponential speed-up, the QPU achieves a speed-up around 10^8.

Hybrid Quantum-Classical Systems

As already stated before, D-Wave's quantum annealer has a limit on the number of logical qubits embeddable into the QPU. This means that it could be difficult to solve large problems with many variables on the QPU alone. Therefore, hybrid quantum-classical approaches can be adopted. By intelligently decomposing the problem, state-of-the-art classical methods are used to solve large problems, allocating to the QPU parts of the problem that best fit quantum annealing. An implementation of this approach is offered by D-Wave's cloud service, Leap, to solve problems of up to 10000 variables.

Chapter 3

Model

The purpose of this work is to find a feasible application of quantum annealing to the field of recommenter systems.

As explained in Section 2.9.2, quantum annealing is able to find good solutions for NP-hard *binary* optimization problems [8] in a very low amount of time. Given the usage of binary variables, quantum annealing seemed to naturally fit into a *feature selection* approach. As described in Section 2.5, feature selection is a specialization of feature weighting. For this reason, also feature selection can be used to cope with the cold-start problem, by embedding collaborative information into a content-based model. In this case, instead of weighting the item features with continuous values, they are weighted with binary values $\{0, 1\}$, hence being kept or discarded.

The domain knowledge of a collaborative filtering model drives the selection of the item features used to build a new content-based model. For this reason, the proposed method is called *Collaborative-driven Quantum Feature Selection* (CQFS).

Therefore, the first section of this chapter explains how the feature selection model, making use of both collaborative and content information, can be formulated as a QUBO model, in order to be solved with quantum annealing. Then, we present the entire algorithm pipeline that makes use of the selection given by CQFS to create a content-based method capable of better dealing with the cold-start problem.

3.1 Defining the Optimization Problem

The focus of this work is on using quantum annealing to select the best features in order to improve content-based recommendations when dealing with the cold-start problem. This means that, as explained in Section 2.9.2, we need to formulate a binary optimization problem as a QUBO model that, when solved, gives a selection of the item features that best represent collaborative information. Here are presented the key components of the optimization problem at the core of CQFS.

3.1.1 Variables

Let's start defining the required optimization problem by its variables. Since they should be binary variables and we are dealing with a feature selection problem, the variables represent the selection of each item feature. Therefore, we have $|F|$ variables $x_f \in \{0,1\}$, where F is the set of item features. When $x_f = 1$, variable f is selected, while $x_f = 0$ means that f is not considered in the model. From now on, we will refer to the vector of binary variables as x.

3.1.2 Collaborative and Content Information

When dealing with the cold-start problem, one possible approach is to embed collaborative information into a content-based model, in such a way that the latter behaves more similarly to a collaborative filtering model. In other feature weighting techniques, such as CFW (Section 2.5.2), this effect is obtained by learning the weights that let the content-based model better approximate the collaborative filtering one.

However, while CFW makes use of SGD in order to directly learn the collaborative similarity structure, CQFS is solved through a quadratic optimization problem. Because of this, we need to define an objective function and some constraints. Since the formulation required by quantum annealing is the one of a QUBO model, the latter cannot be hard constraints, but they have to be represented as penalties into the objective function. But what kind of objective function can be used to represent the need of incorporating collaborative information into a content-based model?

The key to the answer stands in the comparison between the collaborative filtering and the content-based models. What we want to achieve is to make content information tend to the collaborative one, excluding information which comes exclusively from the content-based model, while including the information expressed by both models. Driving the selection with this criteria, the result could be a content-based model that better represents the information it has in common with the collaborative filtering one.

What is called *information* here can be translated in practice as the *similarity model.* Indeed, by comparing the similarity matrix S^{CF}, resulting from a collaborative filtering method, and the similarity matrix S^{CBF}, from a content-based one, both of size $|I| \times |I|$, we can point out the following scenarios:

1. The similarity between two items i and j is present both in S^{CF} and in S^{CBF}. This means that the relation between these two items is expressed both by collaborative and content information and we do

	S_{ij}^{CF}	S_{ij}^{CBF}	**How should the model behave?**
1.	> 0	> 0	Relation between i and j expressed by both CF and CBF model. Encourage this similarity.
2.	$= 0$	$= 0$	Relation absent in both models. No need to take any action.
3.	$= 0$	> 0	CBF is expressing a relation not present in CF. Penalize the presence of this similarity in the CBF model.
4.	> 0	$= 0$	CBF does not have a similarity expressed by the CF model. Nothing can be done, since there is no data to work on.

Table 3.1: Summary of all the possible scenarios for the similarity between two items i and j in a collaborative and a content-based model.

not need to embed any new information, just confirm what is already in the model.

2. The similarity between two items i and j is absent both from S^{CF} and from S^{CBF}. Again, this means that both models are expressing the same kind of information. However, differently from the previous case, no action should be taken by the model, also because zero similarity means no common features to work on with feature selection.

3. The similarity between two items i and j is present only in S^{CBF} and not in S^{CF}. In this case, the content-based model is expressing something which is not expressed by the collaborative one. Since we want the content-based model to behave more like the collaborative one, we aim to avoid the presence of this similarity.

4. The similarity between two items i and j is present only in S^{CF} and not in S^{CBF}. In this case nothing can be done, through feature selection, in order to embed this information into the content-based model, since the items generating this collaborative similarity do not have common features.

From these observations we can say that the objective function we need is a function that is minimized when the selected item features generate a S^{CBF} that has the largest possible number of similarities in common with the chosen S^{CF} and the least possible number of similarities absent from S^{CF}. All the possible conditions are summarized in Table 3.1.

3.1.3 Objective Function

The proposed approach directly compares two similarity matrices, whose rows and columns represent items. However, our model should select features. To better understand the transition from items to features, let's analyze first an objective function built to select items instead of features.

	S_{ij}^{CF}	S_{ij}^{CBF}	Q'_{ij}
1.	> 0	> 0	< 0
2.	$= 0$	$= 0$	0
3.	$= 0$	> 0	> 0
4.	> 0	$= 0$	0

Table 3.2: Value of the coefficient Q'_{ij} for each possible condition on the collaborative and content-based similarities described in Table 3.1.

Let's call z the vector of binary variables z_i representing the selection of each item i. Formulated as a QUBO model, the optimization problem is the following:

$$\begin{aligned} \min \quad & y' = z^T Q' z \\ & z \text{ binary} \end{aligned}$$

where Q' is the matrix containing the coefficients of the objective function that we want to build. Following the approach summarized in Table 3.1, the optimization procedure should avoid selecting two items that generate a similarity only in the content-based model. Instead, items that generate similarities present both in the collaborative and in the content-based model should be selected together. In order to have this kind of selection, the coefficient Q'_{ij} associated with the quadratic term $z_i z_j$, or with the linear term z_i if $j = i$, should be:

- *positive* if the similarity S_{ij} between item i and j is present only in S^{CBF};
- *negative* if S_{ij} is present both in S^{CF} and in S^{CBF};
- *zero* otherwise.

Referring to the same notation of Table 3.1, the values that Q'_{ij} should take, depending on the similarities of the collaborative and the content-based model, are reported in Table 3.2.

In this way, similarities appearing only in the content-based model are penalized, in the sense that, if the items that generate these similarities are selected, the objective function increases. Instead, similarities in common between the two models are encouraged since, when the items that generate them are selected, the objective function decreases.

In order to build the matrix Q', we treat positive and negative coefficients separately. In particular, we call K the $|I| \times |I|$ matrix containing the negative coefficients, associated with similarities we want to keep; we call E the $|I| \times |I|$ matrix containing the positive coefficients, associated with the similarities we would like to eliminate from the model. All the nonzero values of matrix K have value -1, while all the nonzero values of matrix E have value 1. Then, we call *Item Penalization Matrix* (IPM) the matrix Q', since it consists of the penalization coefficients given to the variables z_i corresponding to items. The IPM can thus be computed as a weighted sum of the two component matrices:

$$IPM = \alpha K + \beta E \tag{3.1}$$

where α and β are the weighting parameters controlling how much the negative and positive components should influence the selection.

To sum up, the IPM is the Q matrix of a QUBO model (Section 2.8.1) whose solution corresponds to items that should make the content-based similarity matrix better approximate the collaborative filtering one. Now we need to translate this problem to a feature selection problem. In order to do this, we have to define the following optimization problem:

$$\begin{aligned} \min \quad & y = x^T Q x \\ & x \text{ binary} \end{aligned}$$

where x is the vector of binary variables x_f corresponding to the item features of set F and Q is the $|F| \times |F|$ penalization matrix we need to build.

The similarity between two items, in a content-based model, derives from the features of the two items. In order to preserve or remove this similarity, we should respectively encourage or discourage the selection of the two items' features. In particular, we can apply a similar criteria to the one previously seen for item selection. First of all, let's consider two features f and g, the corresponding variables of the optimization problem, x_f and x_g, and the coefficient that binds them in the objective function, Q_{fg}. Since there could be more than two items having features f and g, let's refer to each possible couple of those items as i and j. From this, we can say that each couple of similarities $(S_{ij}^{CF}, S_{ij}^{CBF})$ verifying conditions 1 or 3 of Table 3.1 should contribute to coefficient Q_{fg}, respectively with a negative and a positive value, as shown in Table 3.2. In particular, we already presented the contribution of each couple of items i and j as the IPM in equation (3.1). Therefore, Q_{fg} can be computed as:

$$\begin{aligned} Q_{fg} &= \sum_{i \in I} \sum_{j \in I} ICM_{if} \cdot ICM_{jg} \cdot IPM_{ij} \\ &= \sum_{i \in I} ICM_{if} \sum_{j \in I} IPM_{ij} \cdot ICM_{jg} \end{aligned}$$

It is straightforward the translation of the last equation into a matrix notation, referring to Q as the *Feature Penalization Matrix* (FPM):

$$FPM = ICM^T \cdot IPM \cdot ICM$$

The FPM is an $|F| \times |F|$ matrix that represents the objective function needed for our feature selection model. It is the Q matrix of a QUBO model whose solution should select features in such a way that the resulting content-based model better approximates the collaborative filtering one.

Combinations of k variables

One way to exploit the flexibility of the QUBO model is by introducing soft constraints as penalties in its objective function. In particular, an additional step in our model provides that only a certain percentage of variables are selected. Thus, the solution should consist of only a certain number of k variables in x having value 1. This can be done by imposing that the model is minimized for each of the k-combinations of its variables. The penalty added to the objective function in order to obtain this result is the following:

$$s \left(\sum_{f=1}^{|F|} x_f - k \right)^2$$

where s is a parameter called *strength*, controlling the magnitude of the penalty. For example, in a problem having 3 variables, if $k = 2$ and $s = 1$, having all the 3 variables equal to 1 would result in an objective function equal to 1:

$$1[(1 + 1 + 1) - 2]^2 = 1$$

Instead, if in the same scenario we have $s = 3$, the objective function would equal 3:

$$3[(1 + 1 + 1) - 2]^2 = 3$$

penalizing more a wrong solution.

Therefore, the resulting optimization problem for CQFS is:

$$\begin{aligned} \min \quad & y = x^T FPM x + s \left(\sum_{f=1}^{|F|} x_f - k \right)^2 \\ & x \text{ binary} \end{aligned}$$

considering both the FPM and the k-combinations constraint.

3.2 Algorithm Pipeline

Now that we presented the underlying model of our quantum feature selection method, we are going to describe the complete workflow of our experiments. We start from data splitting and preprocessing and then we explain the three phases in which the building of the final recommendation model is divided.

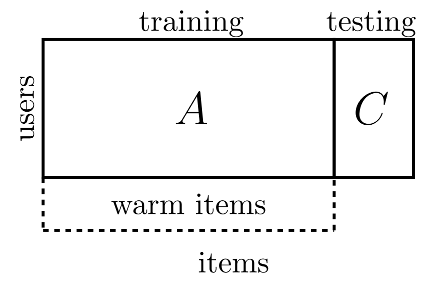


Figure 3.1: Testing cold item split, where A contains the warm items while C contains the cold items.

3.2.1 Data Splitting

When evaluating recommender systems, we need to split the ratings data, the URM, in order to obtain a *training* and a *testing* set. The former is used to train the system, while the latter contains the interactions that the system should try to predict. Here we present the splits we are going to need. First, a testing set on which evaluation metrics for the final recommender system are computed. The, we present the two different validation splits used to optimize the recommendation models. At last, we describe how and why we need to preprocess the ICM during the training phase.

Testing Set

First of all, let's focus on the testing set. We are dealing with the cold-start problem which, as explained in Section 2.4, arises when we have to recommend new items that have no user-item interactions on the platform. Therefore, the testing set to be used in this scenario is a set consisting of all the interactions of a certain number of chosen items.

Thus, we need to perform a *cold item split* [17], which consists in iteratively selecting items from the URM, until the cumulative number of these items' interactions surpasses a certain percentage of the initial interactions. All the interactions of the selected items are moved to the testing set, while they are set to zero in the training set. In this way, the selected items become cold items for the training set. As already stated in Section 2.4, cold items are items that do not have any user interaction. We will refer to the remaining training data as *warm items*, instead. A graphical representation of the testing cold item split is shown in Figure 3.1, where A is the warm items set and C is the testing set.

So, we are recreating a cold-start scenario in the training set. Indeed, the task of our model is to recommend these cold items to the users, based only on the user-item interactions of the training set and on the item features of the ICM. It should be noted that the testing set should not be accessed through any phase of the experiment apart from the evaluation of the final model. This guarantees that no component of CQFS has ever seen the testing data, item features included.

Validation Sets

Now that we defined the testing set, containing all the interactions of the cold items, let's focus on the remaining training set. As later explained in Section 3.2.2, we need to optimize different recommender models, with the procedure described in Section 2.7. The optimization consists in choosing the hyperparameters of the models that give the best performance.

However, how can we evaluate the performance of each hyperparameter set using only the training data, with no access to the testing set? The key is to split again the warm items data into two sets, a training set and a *validation* set. When trying different hyperparameters, the training set is used to train the recommender system, while the validation set is used to evaluate the model on some metrics. The hyperparameters giving the best performance on the validation set are used to build a new model, using the entirety of the warm items, since at deployment we should use all the available data.

In general, depending on the model we are going to optimize, we need different kinds of splits. Indeed, when optimizing the collaborative model needed by CQFS, we use a *random holdout* split, which randomly selects a certain percentage of all the interactions. This way, we can build and optimize the collaborative model on a training and a validation sets having no cold items.

Instead, we want the final content-based method to be evaluated in a cold-start scenario. This means that also the optimization itself should be done under the same conditions. Therefore, we perform another cold item split, obtaining a validation set that contains all the interactions of a certain percentage of items. This way we can select the hyperparameters that should make the model perform best in a cold-start scenario.

To sum up, in our experiments we have two different validation splits, corresponding to two different training sets, both extracted from the warm items. The first one is a random holdout used to optimize the collaborative filtering model in the first phase of the experiment (see Figure 3.2). The second one is instead a cold item split used to optimize the final content-based model produced after the feature selection phase (see Figure 3.3).

Item Features in the Different Phases

After describing how the URM should be split during the experiments, let's shift to the features data, the ICM. Differently from collaborative information, in a cold-start scenario features are available for the cold items. Indeed,

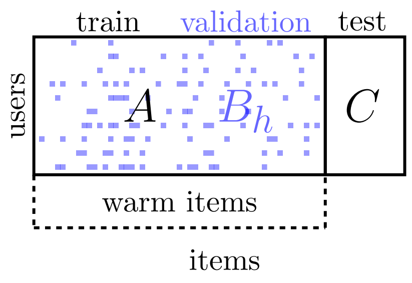


Figure 3.2: URM split, A and B$_h$ contains the warm items, with B$_h$ deriving from a random holdout split, while C contains the cold items.

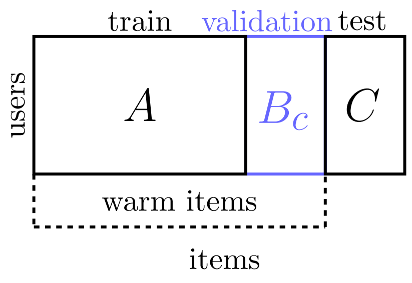


Figure 3.3: URM split, A and B$_c$ contains the warm items, with B$_c$ deriving from a cold item split, while C contains the cold items.

when adding a new item to a platform, we do not have any user interaction, but we already have the features that characterize that item. For example, a new movie on Netflix already has all the information about the cast, the genre, and so on, even if it does not have any user interaction. Therefore, the entire ICM is used to build the final content-based model, including features of the cold items.

However, during the optimization of this final model and previously, when building the content-based similarity before the feature selection phase, we should not use the entire ICM. The reason is that we do not have the new items on the platform yet, so we do not have access to their feature data. Hence, in the ICM used in these two phases we set to zero the feature information of the cold items. This guarantees that the testing data never appears during training phases, but only on the last evaluation of the final model.

3.2.2 Building the CQFS Recommendation System

The next and last step is to build a recommendation system that exploits the feature selection method described in Section 3.1. In particular, the recommendation pipeline consists of three main components:

1. *Collaborative and content-based models*: the first elements needed are the collaborative filtering and content-based similarity matrices from which CQFS computes the FPM, as described in Section 3.1. Since CQFS is a wrapper method, the collaborative model should be trained and optimized (see Section 2.7 for hyperparameter tuning) on the warm items beforehand, in order to work with an already well performing model. Instead, the content-based model can be derived from a simple dot product, since we only need to know if a similarity between two items is present or not, thus not requiring heuristics or optimizations. Also, in this case we only want warm items, so the ICM without the cold items' interactions is used.

2. *Quantum Feature Selection*: then, once we have both the collaborative and content-based models, we can build the quadratic optimization problem. This problem, formulated as a QUBO, can be solved with quantum annealing or hybrid techniques, giving a selection of features that should improve the content-based model by better approximating the collaborative filtering one. Moreover, a constraint can be introduced, directly into the objective function, in order for the solution to have k selected features, as explained in Section 3.1.3. Before sending the problem to the appropriate solver, the FPM is normalized.

3. *Final content-based model*: after the feature selection phase, we use its result to select which features to keep and to remove from the ICM. This time, we start from the initial ICM, the one containing all the items, even the cold ones. We use this ICM because in a cold-start scenario we have access to the cold items' features, although we do not have any interactions for them. Indeed, the URM used to build this model contains only the warm items. With the new ICM resulting from the selection, we build a content-based recommender system which, after being optimized (Section 2.7) using the cold item split validation set (Section 3.2.1), can be used to cope with the cold-start problem.

Chapter 4

Experiments and Results

In this chapter we are going to present in details the experiments that were carried out and their results. In particular, we start by describing the baseline algorithms, which are later compared with our solution, and the collaborative filtering methods used to train CQFS. We give some context on how we chose the data sets to evaluate our work on. Then, for each data set we present the corresponding experiments as follows:

1. we describe the data set itself and its composition, along with some numbers about ratings and features, showing the splits that were made, according to Section 3.2.1;
2. we present the results of quantum feature selection, comparing selections made with different CQFS parameters;
3. we show the results of the final content-based model, after feature selection, and we compare them with the baselines.

4.1 Baseline Algorithms

We propose three baseline algorithms to test against our model in the cold-start scenario recreated following the splits in Section 3.2.1:

- Content-based filtering (CBF): a pure content-based filtering approach is used to have a standard baseline. It consists of an optimized content-based model built from the item features, as explained in Section 2.3. The chosen method is item-based k-nearest neighbors and its parameters, with their respective search ranges used in hyperparameter tuning are presented in Table 4.1. This is the simplest possible way to cope with the cold-start problem, by relying only on the neighborhood model given by the unweighted item features. We will refer to this method as *ItemKNN CBF*;

- TF-IDF: features are selected according to their TF-IDF weights, in a similar manner to [62]. Features are first weighted by their TF-IDF scores and then a certain percentage (we experimented with 40, 60, 80 and 95%) of the highest weighted features are selected. Only the values corresponding to the selected features are used to build the baseline content-based model. Thus, we apply an ItemKNN CBF after TF-IDF, which will be reffered to as TF-IDF CBF or simply, if there is no ambiguity, TF-IDF. The hyperparameter tuning ranges are the same as the previous model, shown in Table 4.1.

- CFeCBF: feature weights are learnt through machine learning from a previously built collaborative model, as explained in Section 2.5.2. The baseline content-based model is then computed from the weighted item features with an ItemKNN CBF. Although this is not a feature selection method, we decided to include it among our baselines because of its successful application on cold-start scenarios. We could have selected features with the highest weights, as with TF-IDF, but we think that this might have denatured the model since it learns and optimizes the weights as a whole, differently from TF-IDF, which computes an index. Hyperparameters for this algorithm are divided into model parameters and learning parameters. The former are the ones used for the content-based model, while the latter are the ones used for the SGD learning procedure with early stopping, such as epochs, learning rate, regularization coefficients and so on. Exclusive parameters to this algorithm are the D matrix initialization, which can be composed of 1s, or can be randomly initialized, and if D should contain only positive values (if it is true, negative values are clamped to 0). Hyperparameter ranges for the optimization of this model are presented in Table 4.2. We will refer to this method as CFeCBF with an appended letter, depending on the collaborative model from which it learns (K for ItemKNN CF, P for PureSVD and R for RP^3_β).

4.2 Collaborative Filering Models

To train CQFS we chose three different collaborative filtering algorithms having a similarity model:

- item-based k-nearest neighbors collaborative filtering (Section 2.2.1), whose optimization is done through hyperparameter tuning with the ranges shown in Table 4.3; we will refer to this method as *ItemKNN CF*;

- PureSVD (Section 2.2.2), whose hyperparameters and respective ranges are presented in Table 4.4; it should be noted that, since we need a

Hyperparameter	Range or Values	Distribution
top-K	5 - 1000	uniform
shrink	0 - 1000	uniform
similarity	cosine	categorical
normalize	true/false	categorical
feature weighting	none/TF-IDF/BM25	categorical

Table 4.1: Hyperparameter tuning ranges for ItemKNN CBF.

Hyperparameter	Range or Values	Distribution
top-K	300	uniform
shrink	0	uniform
similarity	cosine	categorical
normalize	true	boolean
learning rate	10^{-5} - 10^{-2}	log-uniform
SGD mode	Adam	categorical
L1 regularization	10^{3} - 10^{-2}	log-uniform
L2 regularization	10^{3} - 10^{-1}	log-uniform
epochs	300	uniform
use dropout	true	categorical
dropout	30% - 80%	uniform
D initialization	ones/random	categorical
positive only D	true/false	categorical
add zeros quota	0.5 - 1.0	uniform

Table 4.2: Hyperparameter tuning ranges for CFeCBF.

similarity matrix to work with, we use the variant of PureSVD that computes it from the items' latent factors;

- RP^3_β (Section 2.2.3), whose hyperparameters and respective ranges are presented in Table 4.5;

The best hyperparameters for each collaborative filtering method are found using a random holdout validation split as described in Section 3.2.1. Then, the collaborative filtering models used by CQFS are built with these hyperparameters on the entirety of the warm items.

When referring to CQFS in result tables we will indicate the collaborative model from which it learns with an appended letter (K for ItemKNN CF, P for PureSVD and R for RP^3_β).

4.3 Data Sets

The method evaluation consists in an offline evaluation, described in Section 2.6, that requires the use of an existing data set. Since the proposed method is a feature selection technique, we require data sets presenting feature data

Hyperparameter	Range or Values	Distribution
top-K	5 - 1000	uniform
shrink	0 - 1000	uniform
similarity	cosine	categorical
normalize	true/false	categorical
feature weighting	none/TF-IDF/BM25	categorical

Table 4.3: Hyperparameter tuning ranges for ItemKNN CF.

Hyperparameter	Range or Values	Distribution
number of factors	1 - 350	uniform
top-K	5 - 1000	uniform

Table 4.4: Hyperparameter tuning ranges for PureSVD.

for the items. Moreover, because of the physical limits imposed by the quantum annealer and by the hybrid quantum-classical system (Section 2.9.2), we need data sets with no more than a certain number of features that could be selected. For these reasons, after some initial evaluations and preprocessing, *The Movies Dataset* and *Xing Challenge 2017* have been chosen.

4.4 The Movies Dataset

The Movies Dataset is a data set publicly available on Kaggle[1]. It consists of data coming from TMDB[2], integrated into the data of the Full MovieLens Dataset by GroupLens[3]. It contains ratings and various metadata of movies released up to July 2017. It is based on MovieLens20M, which contains only genres and publication years as item features, enriched with many more metadata coming from the TMDB Open API, such as crew and cast members, plot keywords, release dates, production companies and so on. This is a good addition in our scenario, dealing with the cold-start problem,

[1]https://www.kaggle.com/rounakbanik/the-movies-dataset
[2]https://www.themoviedb.org/
[3]https://grouplens.org/

Hyperparameter	Range or Values	Distribution
top-K	5 - 1000	uniform
α	0 - 2	uniform
β	0 - 2	uniform
normalize	true/false	categorical

Table 4.5: Hyperparameter tuning ranges for RP^3_β.

Rating	Interactions
0.5	403156
1.0	841687
1.5	402829
2.0	1759928
2.5	1252990
3.0	5250081
3.5	3109644
4.0	6985966
4.5	2164214
5.0	3802586
Total	25973081

Table 4.6: Number of user-item interactions for each explicit rating in The Movies Dataset.

with respect to MovieLens. Indeed, not only do we have a lot more content information to work with, but metadata collected from TMDB are *editorial features*. This kind of features is much more reliable than user-assigned tags, which could be very sparse and noisy. Moreover, in a cold-start scenario user tags would not be available on new items, while editorial features, being a description of the movies' characteristics, would be known in advance and available for building a model.

The Movies Dataset has a URM with around 26 million ratings from 270882 users for 44711 items. The ratings are explicit, in a range between 0.5 and 5 with a step of 0.5, distributed as shown in Table 4.6 and in Figure 4.1. Instead, in Table 4.7 there is the distribution of the interactions with respect to the items. As we can see, half of the items have less than 10 interactions, with only around a quarter of them having at least 5 interactions. However, no preprocessing procedure is applied to the URM.

The ICM is a boolean matrix indicating if an item has a certain feature. There are around 1.5 million values in the ICM, containing 368953 features for the 44711 items. Features are extracted from the data set's metadata by parsing and vectorizing the information. Examples of present features are genre, year, production company and country, original language and so on. From the crew data, the extracted features are represented by the names of the cast and crew members. The distribution of all these features is presented in Table 4.8, where we can see how more than half of the features belong to only one item. The noise introduced by these features can be removed by filtering out all the features with less than a certain number of interactions. Indeed, before using feature data in our model, features are preprocessed by removing all the ones that belong to less than 5 items.

However, we are not going to treat all the features at the same time in our model. Indeed, we are going to make a distinction between credit

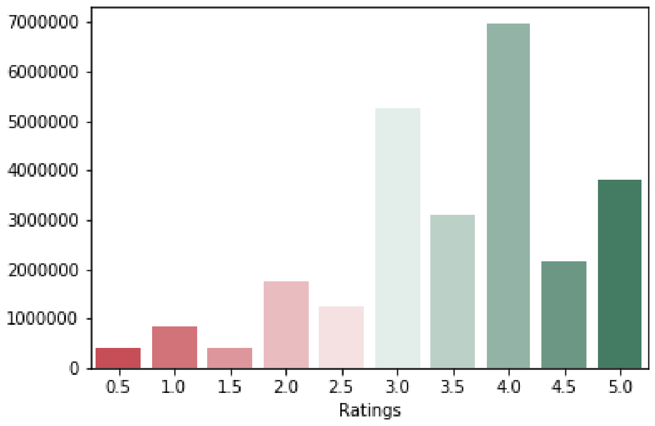


Figure 4.1: Ratings count of The Movies Dataset.

N	Items with N interactions	Items with **at least** N interactions
1	7567	44711
5	1665	26933
10	665	21108
15	399	18404
20	248	16678
25	202	15493
30	143	14631
35	113	13971
40	106	13416
45	76	12887
50	92	12455

Table 4.7: Number of items that have N or more interactions in The Movies Dataset.

N	Features belonging to N items	Features belonging to **at least** N items
1	222702	367123
5	10177	48613
10	2485	19231
15	1010	10275
20	520	6234
25	337	4162
30	183	2862
35	137	2057
40	87	1553
45	64	1199
50	32	937

Table 4.8: Distribution of all the item features in The Movies Dataset.

features, such as cast and crew members, and metadata features, such as genre, year and so on. We ran different experiments for the two types of features, both using a hybrid quantum-classical approach to solve the CQFS optimization problem, because of the high number of features used. Here we present a brief description of the two resulting data sets and the results obtained from them.

4.4.1 The Movies Dataset - Credits

First, let's analyze the features related to the movies' credits. In particular, as stated before, the extracted data regards crew and cast members of the movies. Each entry corresponds to one person having a particular role in the making of the movie. Some of the most relevant jobs are reported in Table 4.9, along with the number of features appearing with those jobs in the original data, the occurrences of these features in the credits ICM and the percentage of movies with at least a feature with that job.

Features in the credits ICM are represented by names, indicating if the corresponding person has been involved in the production of the movie. There are 960614 values in the credits ICM, with 44711 item and 343018 features. As we can see in Table 4.9, the majority of these features correspond to actors (202748), which cover more than a half of the nonzero values of the credits ICM. However, directors represent another popular job category, covering almost every movie in the data set with at least one feature corresponding to a director.

After applying the filter that removes features belonging to less than 5 items, the features distribution greatly changes, as shown in Table 4.10. All the categories are largely impacted in terms of number of features, which is around $\frac{1}{10}$ of the original. However, occurrences and the percentage of movies with at least one feature from those jobs are not influenced in the

Job	Features	Occurrences	Percentage
Director	19740	48999	0.9896
Actor	202748	562053	0.9557
Producer	20017	43507	0.4885
Writer	19794	30398	0.4554
Editor	8047	23770	0.4379
Director of Photography	5684	20640	0.4244
Screenplay	12817	25143	0.3491
Original Music Composer	5081	15773	0.3107
Costume Design	4014	11010	0.2230
Production Design	4134	10524	0.2204

Table 4.9: Jobs appearing in more than 20% of the movies in the credits ICM, ordered by percentage of movies in which they appear.

Job	Features	Occurrences	Percentage
Actor	25159	308725	0.8695
Director	2293	23163	0.4927
Director of Photography	1068	13668	0.2934
Editor	1264	13621	0.2707
Producer	1820	17621	0.2653
Original Music Composer	646	9480	0.2005
Casting	570	9615	0.1461
Screenplay	1058	8409	0.1451
Costume Design	513	5859	0.1257
Production Design	569	5181	0.1136

Table 4.10: Jobs appearing in more than 10% of the movies in the credits ICM **after filtering**, ordered by percentage of movies in which they appear.

same way. Indeed, they set to around $\frac{1}{2}$ of the original values, apart from the percentage of movies containing actors, which is only 10% less, because of the very high number of features representing actors.

Given this filtering results, we had to choose what features to use in our experiment, since the limit on the number of variables for the hybrid quantum-classical solver is set to 10000 (see Section 2.9.2). Actors are represented by a number of features too large for this limit, that would have required a more aggressive filtering to be lowered enough. Thus, we chose to use the directors features. Indeed, while the filtering process greatly reduces the number of features from this category, the occurrences are only halved and around half of the movies have at least one director. Moreover, a number of features not too close to the limit of variables has been useful to analyze the performance of the hybrid approach on a medium-sized problem. Thus, the ICM is built from the director features only, with 23163 boolean values indicating which of the 2293 features belong to which items.

Set	Interactions	Covered users	Covered items
Training	18180781	269473	35289
Validation	2594836	263644	22390
Testing	5195793	254039	8593

(a) Random holdout split.

Set	Interactions	Covered users	Covered items
Training	18125425	268485	31354
Validation	2649073	233737	4764
Testing	5195201	253625	8593

(b) Cold item split.

Table 4.11: Interactions per split of The Movies Dataset.

Data Splitting

As described in Section 3.2.1, before training any model we need to split the data. First, we perform a cold item split in order to obtain a testing set containing around 20% of the interactions. Then, depending on the model we need to optimize, we use two different kinds of split, a random holdout or a cold item split, to obtain the appropriate validation and training sets, with respectively around 10% and 70% of the total number of interactions. In Table 4.11 we report some data about the obtained sets, with the ones resulting from random holdout in Table 4.11a and the ones from cold item split in Table 4.11b.

As we can see, the testing sets for both types of split contain the same number of items. However, they have a different number of interactions. This is due to the fact that we do not allow cold users (the ones without any interaction) in the training set, thus we remove them also from validation and testing set. This can happen when all the interactions of a user are moved to the validation and testing sets. In our case, this difference does not create any problem, since the random holdout split is used only to optimize the collaborative filtering models. The number of users that have at least one interaction in each of the three sets is reported in the *covered users* column of the tables.

It should be noted that the numbers of covered items between the three sets of the cold item split (Table 4.11b) sum up to the total number of items, 44711, as expected from this kind of split.

Collaborative Filtering Models

The collaborative filtering models chosen for this set of experiments on director credits data are ItemKNN CF, PureSVD and RP^3_β. The best hyperparameters found by the hyperparameter tuning procedure (Section 2.7) are

Model	Hyperparameter	Value
ItemKNN CBF	top-K	10
	shrink	19
	similarity	cosine
	normalize	true
	feature weighting	none

Table 4.12: Best hyperparameters found for an ItemKNN CBF model built with director credits and optimized on the validation random holdout split.

Model	PREC	REC	NDCG	MAP	IC	GD	MIL
ItemKNN CBF	0.0145	0.0152	0.0139	0.0084	0.2503	0.0312	0.9765
ItemKNN CF	0.1298	0.1931	0.1719	0.1373	0.1641	0.0115	0.9605
PureSVD	0.1333	0.1981	0.1839	0.1438	0.0214	0.0030	0.9074
RP^3_β	0.1381	0.2022	0.1783	0.1468	0.2165	0.0117	0.9605

Table 4.13: Comparison between the quality of an ItemKNN CBF built with director credits against the chosen collaborative methods on the random holdout validation split.

presented in Table 4.14. Instead, a comparison between these methods and an ItemKNN CBF (best hyperparameters shown in Table 4.12) built with the director credits ICM, and optimized on the random holdout validation set, is presented in Table 4.13. The metrics are evaluated on the validation set of the warm items random holdout split (see Section 3.2.1 and Figure 3.2), used to optimize the methods.

This is done to understand how much the collaborative methods are good with respect to the content-based one. As we can see, the collaborative methods greatly outperform the content-based algorithm. This means that we can safely rely on collaborative information to improve the accuracy of a content-based model through wrapped methods, such as CQFS.

Baseline Models

As stated before, the chosen baseline algorithms are a standard ItemKNN CBF built from the directors ICM with all the 2293 features, an ItemKNN CBF built after selecting the highest weighted features by TF-IDF, with different selection percentages (40, 60, 80 and 95%), and a CFeCBF, which learns weights for all the features from the three optimized collaborative models. The best hyperparameters found for these models are reported in Table 4.15, Table 4.16 and Table 4.17. On the base of previous results in the literature [15, 17], we set default values for some parameters of CFeCBF methods (the ones having a fixed value in Table 4.2), thus we report only

Model	Hyperparameter	Value
ItemKNN CF	top-K	31
	shrink	752
	similarity	cosine
	normalize	true
	feature weighting	none
PureSVD	number of factors	31
	top-K	1000
RP^3_β	top-K	69
	α	0.0
	β	0.5583
	normalize	true

Table 4.14: Best hyperparameters found for the collaborative filtering algorithms applied on The Movies Dataset.

Model	Hyperparameter	Value
ItemKNN CBF	top-K	6
	shrink	31
	similarity	cosine
	normalize	false
	feature weighting	TF-IDF

Table 4.15: Best hyperparameters found for the ItemKNN CBF baseline of director credits experiments.

the non-fixed parameters.

Quantum Feature Selection

Once the collaborative filtering models are optimized, we can use them in the CQFS model to learn which features to select. As explained in Section 3.1.3, the similarity matrices built by the collaborative models are compared to the similarity matrix built by the dot product $ICM \cdot ICM^T$. In this case, director features are highly sparse, which results in a similarity matrix having a very low number of nonzero values and density, reported in Table 4.18. Here are also reported the same properties about the similarities derived from the collaborative methods, in order to make a quick comparison. In particular, we show what percentage of the content similarity is covered by the collaborative similarity. As we can see, only around 10% of the content similarities are present in the different collaborative models, meaning that we should expect around 90% of the content similarities to be penalized with a positive value in the CQFS model.

Now, let's analyze the quantum feature selection itself. We carried out

Model	Hyperparameter	Value
TF-IDF CBF 40%	top-K	18
	shrink	995
	similarity	cosine
	normalize	true
	feature weighting	none
TF-IDF CBF 60%	top-K	26
	shrink	7
	similarity	cosine
	normalize	false
	feature weighting	TF-IDF
TF-IDF CBF 80%	top-K	6
	shrink	25
	similarity	cosine
	normalize	true
	feature weighting	none
TF-IDF CBF 95%	top-K	8
	shrink	994
	similarity	cosine
	normalize	true
	feature weighting	TF-IDF

Table 4.16: Best hyperparameters found for the TF-IDF CBF baselines of director credits experiments.

Model	Hyperparameter	Value
CFeCBF K	learning rate	7.59×10^{-3}
	L1 regularization	1.13×10^{-3}
	L2 regularization	1.35×10^{-3}
	epochs	15
	dropout	75.97%
	D initialization	ones
	positive only D	true
	add zeros quota	6.95×10^{-1}
CFeCBF P	learning rate	1.02×10^{-3}
	L1 regularization	1×10^{-2}
	L2 regularization	1×10^{-1}
	epochs	20
	dropout	80%
	D initialization	random
	positive only D	true
	add zeros quota	1.0
CFeCBF R	learning rate	9.32×10^{-4}
	L1 regularization	6.70×10^{-3}
	L2 regularization	9.73×10^{-2}
	epochs	45
	dropout	39.29%
	D initialization	random
	positive only D	false
	add zeros quota	5.10×10^{-1}

Table 4.17: Best hyperparameters found for the CFeCBF baselines of director credits experiments.

Model	S nonzero values	S density	Covered CBF %
CBF dot product	198512	9.93×10^{-5}	-
ItemKNN CF	1117286	5.59×10^{-4}	9.76%
PureSVD	36129000	1.81×10^{-2}	13.13%
RP^3_β	1445598	7.23×10^{-4}	9.40%

Table 4.18: Data about content and collaborative similarities for director credits of The Movies Dataset.

Parameter	Values
α	1
β	1, 10^{-1}, 10^{-2}, 10^{-3}
p	40%, 60%, 80%, 95%
s	1, 10^{-1}, 10^{-2}, 10^{-3}

Table 4.19: Tested parameters for the initial experiment on director credits of The Movies Dataset.

initial experiments with different values for CQFS parameters, reported in Table 4.19. In particular, we always set the coefficient of the negative components of the FPM, α, to 1. Instead, we tried different values for the coefficient of the positive components of the FPM, β, in order to understand how the similarities to discard, which are many more in number, could have influenced the selection. Moreover, we made use of the k-combinations penalty in the objective function in order to select a certain percentage p of all the features (40, 60, 80 and 95%). The penalty is weighted by a strength parameter, s, for which we tried different values. These experiments were done only with ItemKNN CF, in order to have an initial set of results to understand how the parameters influenced our model.

In Figure 4.2 we can see how precise the procedure was in selecting the requested number of features. In this case, every combination of parameters β and s gave around the desired number of selected features, for each selection percentage p, as we can see from the annotations on the figures, which indicate the ratio of the number of actually selected features on the desired number. Since the number of actually selected features is really close to the one we desired, we can carry out our experiments on this data set with the default parameters, $\alpha = 1$, $\beta = 1$, $s = 1$.

In Table 4.20 we show some statistics about the coefficients in the FPM, before applying the k-combinations constraint, for the default parameters of CQFS. In particular we make a distinction between coefficients of the linear terms of the objective function (the ones on the diagonal, see Section 2.8.1) and coefficients of the quadratic terms of the objective function. As we can see, the majority of both linear and quadratic coefficients are positive. This is consistent with what we expected from the previously highlighted percentages of similarities to exclude in Table 4.18, since positive values are assigned to features that generate those similarities. Moreover, $\beta = 1$ means that the positive terms E of equation (3.1), used to build the FPM, have the same weight of the negative terms K of the same equation.

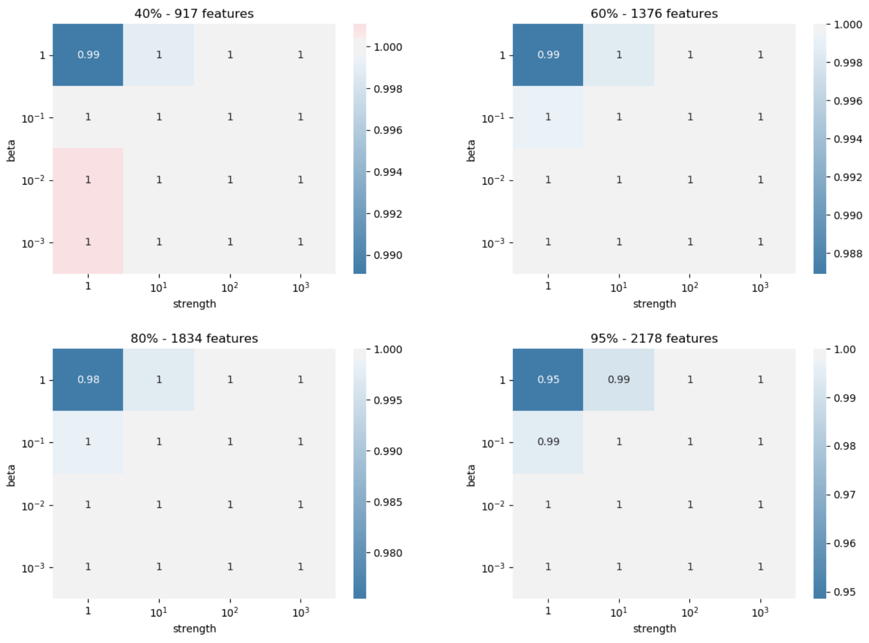


Figure 4.2: Ratios of the number of actually selected features on the desired number, for various parameters of CQFS applied to director credits of The Movies Dataset.

CF Model	Coefficients	% negative	% positive	Min	Max	Mean
ItemKNN CF	Linear	3.27%	95.68%	-86	2222	70.07
	Quadratic	2.80%	97.20%	-21	218	9.39
PureSVD	Linear	5.36%	93.15%	-410	2082	64.45
	Quadratic	6.82%	93.18%	-74	585	8.18
RP^3_β	Linear	2.53%	96.47%	-182	2178	70.71
	Quadratic	0.84%	99.16%	-20	218	9.58

Table 4.20: Statistics about values in linear and quadratic coefficients of the FPM, for each collaborative method, in the director credits experiments.

	Model	PREC	REC	NDCG	MAP	IC	GD	MIL
Baselines	ItemKNN CBF	0.0606	0.0307	0.0328	0.0318	0.3619	0.0377	0.9312
	TF-IDF 60%	0.0716	**0.0410**	**0.0447**	**0.0376**	0.3084	0.0404	0.9440
	CFeCBF K	0.0547	0.0344	0.0323	0.0231	0.2173	0.0226	0.9043
	CFeCBF P	0.0501	0.0338	0.0301	0.0202	0.0839	0.0149	0.8875
	CFeCBF R	0.0540	0.0359	0.0349	0.0244	0.2352	0.0291	0.9386
CQFS	CQFS K 60%	0.0582	0.0284	0.0338	0.0283	0.1963	0.0206	0.9028
	CQFS P 60%	**0.0724**	0.0373	0.0424	0.0355	0.1905	0.0270	0.9355
	CQFS R 60%	0.0606	0.0290	0.0355	0.0309	0.1967	0.0211	0.9076
	CQFS K 80%	**0.0729**	0.0363	0.0412	0.0364	0.2645	0.0300	0.9313
	CQFS P 80%	**0.0726**	0.0392	0.0409	0.0351	0.2595	0.0339	0.9438
	CQFS R 80%	**0.0739**	0.0363	0.0414	0.0375	0.2629	0.0292	0.9292

Table 4.21: Evaluation metrics computed on The Movies Dataset testing set for baseline algorithms and CQFS using director credits features.

Final Content-based Model

After selecting features with CQFS, we set to 0 the values of the non-selected features in the ICM. Then, for each different selection, we built the final content-based model as an ItemKNN CBF and we optimized it with hyperparameter tuning on the ranges in Table 4.1. We decided to not report hyperparameters for the final content-based methods, because of the overall high number of models built (one for each combination of CQFS parameters). This is valid for other sets of experiments as well.

Let's recall from Section 3.2.1 that the ICM used for optimization did not contain any data about cold items, while the one used for testing evaluation did. The results of the testing evaluation on a cutoff at 10 of the recommendations are reported in Table 4.21, along with the results of the baselines on the same testing set. We do not report the results for each selection percentages for TF-IDF and CQFS. Instead, we report only the best ones in terms of precision and, in the case of CQFS, also the 60% selection in order to compare it directly with TF-IDF and with CQFS at 80%. Accuracy metrics of CQFS which surpass the baselines are reported in bold, in order to highlight them. Instead, if a baseline outperformed every other method in a certain accuracy metric, only its corresponding value is highlighted.

As we can see, the only model comparable with CQFS is TF-IDF, which is able to select less features and obtain better recall, NDCG and MAP values, while CQFS achieves better precision results, in general. An interesting result is given by CQFS with PureSVD when selecting 60% of the features. Indeed, with the same percentage of features, while sacrificing on item coverage, it is able to achieve better precision than TF-IDF with a comparable NDCG, even higher than other CQFS parameters. Moreover, CQFS with PureSVD at 60% outperforms CQFS with both ItemKNN CF and RP^3_β at 60%, in every accuracy metric.

A more detailed comparison between CQFS and TF-IDF is shown in

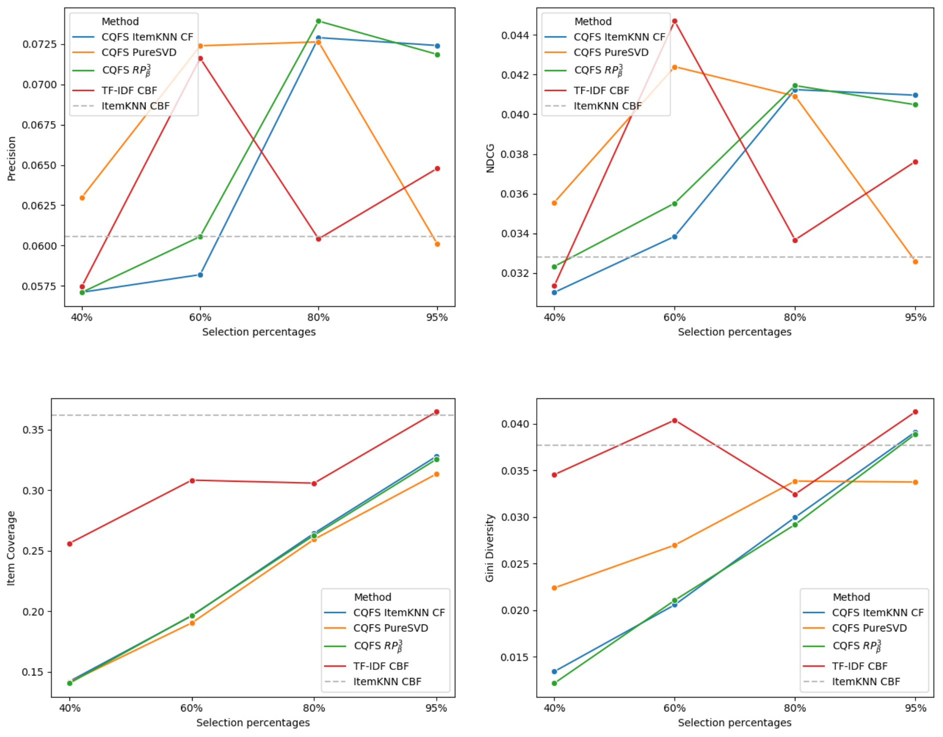


Figure 4.3: Plots comparing four metrics (precision, NDCG, item coverage and Gini diversity) on different experiments made on director credits of The Movies Dataset with CQFS and TF-IDF, for the four tested selection percentages. ItemKNN CBF baseline is shown for reference.

Figure 4.3. Here, the results on four metrics are plotted against the tested selection percentages. Despite giving good results at 60%, TF-IDF seems to not be as consistent as CQFS at higher percentages. The only exception is CQFS with PureSVD, which has a very different behavior than the other two models, giving the best results with 60 and 80% of the features, while going event below the ItemKNN CBF baseline with 95%.

However, the fact that TF-IDF CBF is able to achieve such accuracy using only 60% of the features, confirmed as well by CQFS PureSVD, means that many of the director features could be noisy or not representing the actual interests of the users. This is confirmed by the accuracy of the CFeCBF weighting algorithm which, even making use of all the features, is not able to outperform any other method.

With respect to item coverage and diversity of the recommendations, we can denote how ItemKNN CBF and TF-IDF CBF both have higher scores than almost all the other methods. One of the reasons is that the number of features used to build ItemKNN CBF are more than the ones used for the other methods, thus the model presents many similarities which are not expressed by the others and that generate recommendations of more items. Indeed, this is confirmed by the coverage and diversity of the methods that used 95% of the features, which have higher scores than with fewer selected features. Moreover, the lower scores achieved by CQFS and CFeCBF methods, which both learn from a collaborative model, are explained by the bias that collaborative filtering methods have towards popular items. This bias tends to reduce the diversity of recommended items, which is reflected in CFeCBF and in the CQFS models with fewer features (40, 60 and 80%).

4.4.2 The Movies Dataset - Metadata

The other set of item features from The Movies Dataset, not extracted from credits data, are the *metadata*. Among them, some interesting features found are 20 genres, 23692 unique production companies, 133 spoken languages and 161 production countries.

All the metadata features total to a number of 25935, a lot less than the credits, but the ICM has 488690 nonzero values, which makes the metadata ICM more populated than the credits ICM. Indeed, when filtering the features in order to remove the ones belonging to less than 5 items, the number of features remaining is almost the same in every relevant category, as shown in Table 4.22. The only categories showing a large cut are production company, which may be due to the less successful companies or the ones established only for a few movies, and movie collection, since collections with less than 5 movies are discarded. Feature categories not reported in the table are categories containing only one feature, indicating for example if the movie has already been released.

The total number of features remaining is 3058 and the total number

	Before filtering		After filtering	
Category	Features	Occurrences	Features	Occurrences
Genre	20	89821	20	89821
Original Language	89	44697	57	44645
Production Company	23692	69782	2493	39922
Production Country	161	48698	105	48572
Release Date	135	44626	126	44613
Spoken Language	133	52588	88	52493
Movie Collection	1695	4433	160	1191

Table 4.22: Number of features and ICM occurrences per metadata category before and after filtering the metadata features to remove the ones belonging to less than 5 items.

of occurrences in the filtered ICM amounts to 455300. Thus, given that the number of features after filtering is way below the limit imposed by the hybrid quantum-classical approach (Section 2.9.2), we decided to make our experiments on the filtered metadata ICM.

Data Splitting

The training, validation and testing sets used for experiments with metadata features are the same used for experiments with director credits features. Again, for the validation set we used the random holdout split (Table 4.11a) to optimize the collaborative model, and the cold item split (Table 4.11b) to optimize the final content-based model.

Collaborative Filtering Models

Differently from the credit features experiments, here we decided to select PureSVD as the only collaborative model used by CQFS. This choice was made in order to limit the computational costs of the experiments and the costs of the quantum-classical solver. Therefore, we chose PureSVD because of the interesting results shown in previous experiments and because of its lower computational requirements.

The best hyperparameters for this model are the same as with credits experiments, since user interaction data and split sets do not change between the two sets of experiments. Thus, they are reported in Table 4.14. Instead, a comparison between PureSVD and an ItemKNN CBF (best hyperparameters in Table 4.23) built with the metadata ICM is presented in Table 4.24. As with director credits experiments, the metrics are evaluated

Model	Hyperparameter	Value
ItemKNN CBF	top-K	6
	shrink	989
	similarity	cosine
	normalize	true
	feature weighting	BM25

Table 4.23: Best hyperparameters found for an ItemKNN CBF model built with metadata features and optimized on the validation random holdout split.

Model	PREC	REC	NDCG	MAP	IC	GD	MIL
ItemKNN CBF	0.0305	0.0367	0.0383	0.0253	0.3603	0.0308	0.9643
PureSVD	0.1333	0.1981	0.1839	0.1438	0.0214	0.0030	0.9074

Table 4.24: Comparison between the quality of an ItemKNN CBF built with metadata features against the chosen collaborative method on the random holdout validation split.

on the validation set of the warm items random holdout split (see Section 3.2.1 and Figure 3.2), used to optimize the methods.

From this comparison we can observe how, also in this case, the collaborative method is able to outperform the accuracy of the content-based algorithm. This suggests that embedding collaborative information into the content-based model, we should be able to improve its accuracy.

Baseline Models

As baseline models we have a standard ItemKNN CBF built from the metadata ICM with all the 3058 features, an ItemKNN CBF model built after selecting the highest weighted features by TF-IDF, with different selection percentages (40, 60, 80 and 95%), and a CFeCBF learning from the optimized PureSVD collaborative model. The best hyperparameters found for these models are reported in Table 4.25, Table 4.26 and Table 4.27. Also in this case, only the non-fixed hyperparameters of CFeCBF are reported.

Quantum Feature Selection

After optimizing the PureSVD collaborative model, we used it to select features through CQFS. If we compare data about the similarity model obtained with PureSVD and the content similarity model obtained with the dot product of the metadata ICM with itself, we find a very different scenario than with director credits features.

Model	Hyperparameter	Value
ItemKNN CBF	top-K	989
	shrink	2
	similarity	cosine
	normalize	false
	feature weighting	TF-IDF

Table 4.25: Best hyperparameters found for the ItemKNN CBF baseline of metadata features experiments.

Model	Hyperparameter	Value
TF-IDF CBF 40%	top-K	989
	shrink	728
	similarity	cosine
	normalize	false
	feature weighting	TF-IDF
TF-IDF CBF 60%	top-K	1000
	shrink	1000
	similarity	cosine
	normalize	false
	feature weighting	TF-IDF
TF-IDF CBF 80%	top-K	999
	shrink	486
	similarity	cosine
	normalize	false
	feature weighting	TF-IDF
TF-IDF CBF 95%	top-K	993
	shrink	17
	similarity	cosine
	normalize	false
	feature weighting	TF-IDF

Table 4.26: Best hyperparameters found for the TF-IDF CBF baselines of metadata features experiments.

Model	Hyperparameter	Value
CFeCBF P	learning rate	7.25×10^{-4}
	L1 regularization	1.62×10^{-3}
	L2 regularization	5.06×10^{-3}
	epochs	25
	dropout	79.91%
	D initialization	random
	positive only D	true
	add zeros quota	5.53×10^{-1}

Table 4.27: Best hyperparameters found for the CFeCBF baseline of metadata features experiments.

Model	S nonzero values	S density	Covered CBF %
CBF dot product	1304257094	6.52×10^{-1}	-
PureSVD	36129000	1.81×10^{-2}	2.77%

Table 4.28: Data about content and collaborative similarities for metadata features of The Movies Dataset.

Parameter	Values
α	1
β	1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}
p	40%, 60%, 80%, 95%
s	1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}

Table 4.29: Tested parameters for the experiments on metadata features of The Movies Dataset.

Indeed, as shown in Table 4.28, the number of similarities in the content model are now two order of magnitude more than the number of similarities in PureSVD. Moreover, the collaborative similarities cover only a small fraction of the content ones, while the content model contains 99.95% of the collaborative similarities. This means that the CQFS model should penalize with positive values almost all the similarities present in the content model, since they are absent from the collaborative one.

Because of this imbalance, we tested CQFS with the different parameters reported in Table 4.29. The results of the selection with these parameters are shown in Figure 4.4, in the form of heat maps of the ratios between the actual number of the selected features and the desired number. These results confirm the expected behavior, showing how, with standard parameters, CQFS is not able to make a precise selection.

Some statistics about the values in the FPM, before applying the k-combinations constraint, are reported in Table 4.30, for each tested value of parameter β. As we can see, when $\beta = 1$ almost every value of the FPM is positive, which was expected, because of the high imbalance between similarities to discard and similarities to maintain. Although similar percentages of negative and positive values also characterize the director credits experiments, it is not the same for the actual values. Indeed, here we have maximum values of many more orders of magnitude. This contributes to unbalance the FPM, leading to the selection of only around 10% of the desired features when $\beta = 1$ and $s = 1$. Instead, decreasing β we obtain the opposite effect, as seen in the FPM with $\beta = 1 \times 10^{-4}$, where we have negative numbers with very high absolute values, as opposed to positive numbers with low values.

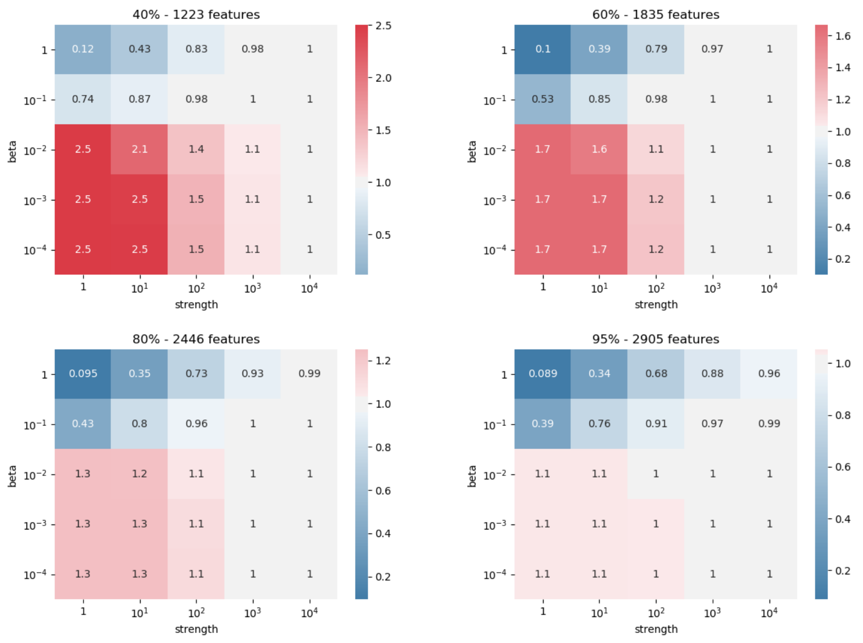


Figure 4.4: Ratios of the number of actually selected features on the desired number, for various parameters of CQFS applied to metadata features of The Movies Dataset.

β	Coefficients	% negative	% positive
1	Linear	6.64	91.79
	Quadratic	0.76	99.24
1×10^{-1}	Linear	38.49	61.41
	Quadratic	21.20	78.80
1×10^{-2}	Linear	57.42	42.48
	Quadratic	46.31	53.69
1×10^{-3}	Linear	58.99	40.91
	Quadratic	49.66	50.34
1×10^{-4}	Linear	59.16	40.75
	Quadratic	49.99	50.01

(a) Percentages of negative and positive values in the FPM.

β	Coefficients	Min	Max	Mean
1	Linear	-918	1232949796	1827092.09
	Quadratic	-13015	1230629157	12905.58
1×10^{-1}	Linear	-16330.00	90784127.09	126252.23
	Quadratic	-595767.60	90613262.49	851.95
1×10^{-2}	Linear	-23432471.19	23807.87	-43714.23
	Quadratic	-23994607.62	633749.32	-350.72
1×10^{-3}	Linear	-34854124.93	823.00	-60747.95
	Quadratic	-34865488.91	30110.62	-470.48
1×10^{-4}	Linear	-35996305.70	2.33	-62436.55
	Quadratic	-35996331.80	510.11	-483.10

(b) Minimum, maximum and mean of the values in the FPM.

Table 4.30: Statistics about values in linear and quadratic coefficients of the FPM, for each collaborative method, in the metadata features experiments.

Because of this imbalance, we decided to use higher s values, in order to obtain selections with the correct number of selected features. Thus, we used the feature selections given by CQFS with $s = 10^2$ and $s = 10^3$; we excluded 10^4 because increasing the strength of the k-combinations constraint means that the solver could be more inclined to select the correct number of variables rather than optimizing our main problem. As values of β we decided to test 1, 1×10^{-1}, 1×10^{-2} and 1×10^{-3}, although the best accuracy was obtained with $\beta = 1 \times 10^{-2}$ and $\beta = 1 \times 10^{-3}$. Thus, we are going to focus more in details on these two values.

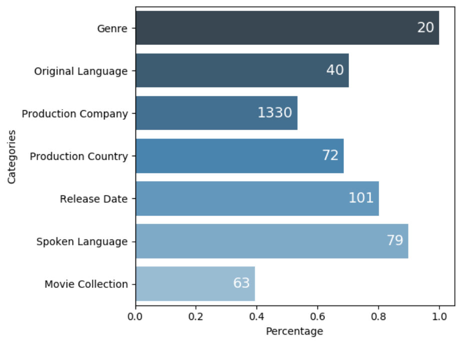
To better understand how CQFS selects the features, we show in figure Figure 4.5 how many features for each category it selects with selection percentage 40% and different values of β and s. The bar plots represent the percentage of features in the various categories that have been selected, annotated with the absolute number of features selected from each category. As we can see, all the genres are selected with every combination of parameters and the percentages for each category are generally consistent with different values for β and s. However, it should be noted that when $s = 10^2$ CQFS selects many more features (around 1800) than the desired number (1223), as opposed to $s = 10^3$ (around 1300). For this reason, from now on we will report results from experiments with the k-combinations strength parameter s set to 10^3.

For the sake of completeness, in Figure 4.6 we show the percentage of features from each category selected by CQFS with the remaining selection percentages (60, 80 and 95%) with $s = 10^3$ and variable β. Here we can see how the percentages of features selected from each category are consistent when varying the selection percentage of CQFS.

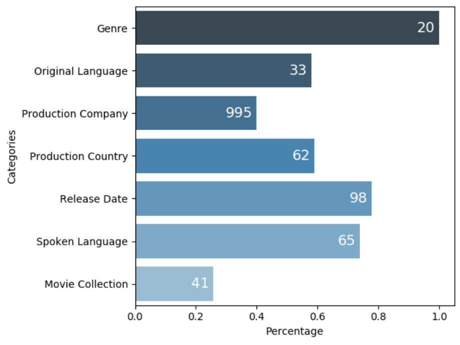
Final Content-based Model

After selecting feature with CQFS, we set to 0 the values of the non-selected features in the ICM. Then, for each combination of parameters (percentage selection and β, since $s = 10^3$), we built the final content-based model as an ItemKNN CBF optimized with hyperparameter tuning on the ranges shown in Table 4.1. The results of the testing evaluation on a cutoff at 10 of the recommendations are reported in Table 4.31, along with the results of the baselines on the same testing set.

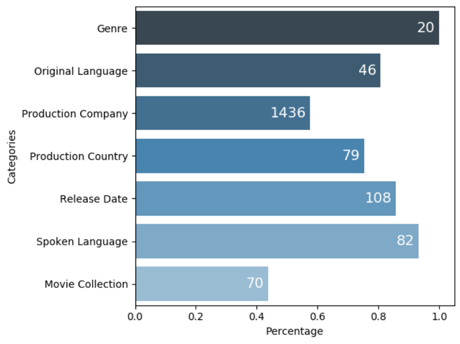
As with the director credits experiments, we only report the most interesting results between the collected ones. In particular, we chose the best performing selection percentage overall. We can see how CQFS outperforms all the baselines in every accuracy metric, even though by a small amount. However, what is really interesting about these results is the fact that CQFS accuracy scores are very close with the ItemKNN CBF baseline ones, even though CQFS uses only 40% of the features. This means that more than half of the metadata features are not meaningful when making content-based



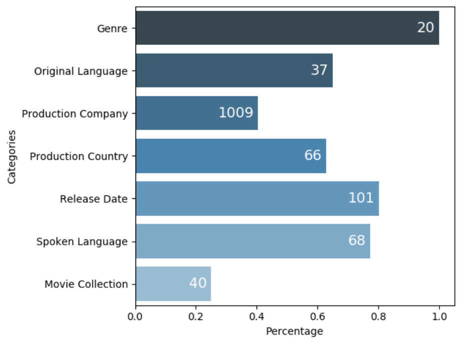
(a) $\beta = 1 \times 10^{-2}$, $s = 10^2$



(b) $\beta = 1 \times 10^{-2}$, $s = 10^3$



(c) $\beta = 1 \times 10^{-3}$, $s = 10^2$

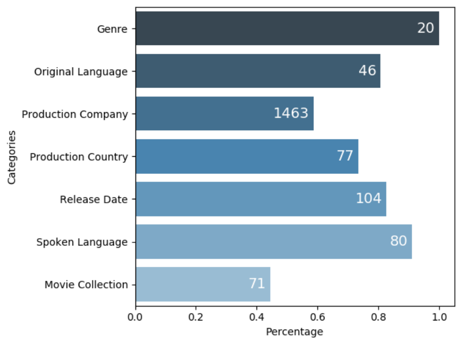


(d) $\beta = 1 \times 10^{-3}$, $s = 10^3$

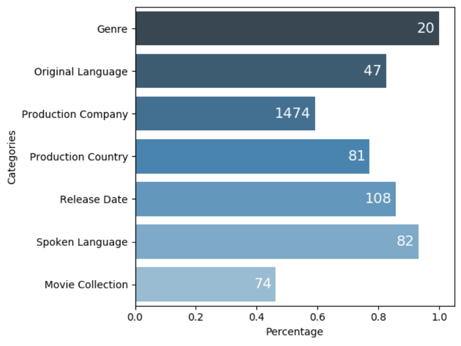
Figure 4.5: Bar plots showing the percentage of features selected for each category with CQFS at 40% with different β and s values.

Model		PREC	REC	NDCG	MAP	IC	GD	MIL
Baselines	ItemKNN CBF	0.1057	0.0698	0.0669	0.0660	0.6746	0.0830	0.9510
	TF-IDF 95%	0.1053	0.0694	0.0664	0.0656	0.6724	0.0556	0.9507
	CFeCBF P	0.0670	0.0438	0.0411	0.0364	0.3048	0.0238	0.8502
CQFS P 40%	$\beta = 1 \times 10^{-2}$	**0.1058**	**0.0701**	**0.0672**	**0.0663**	0.6461	0.0538	0.9502
	$\beta = 1 \times 10^{-3}$	0.1056	**0.0702**	**0.0671**	**0.0662**	0.6473	0.0538	0.9502

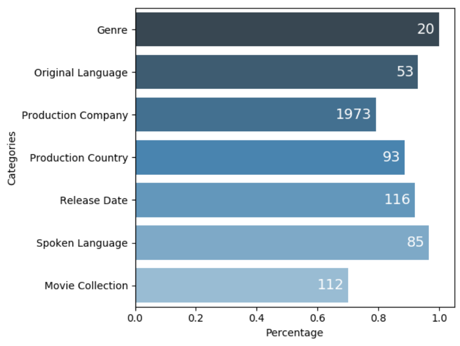
Table 4.31: Evaluation metrics computed on The Movies Dataset testing set for baseline algorithms and CQFS using metadata features.



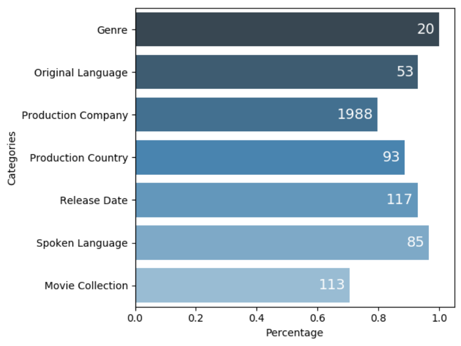
(a) $p = 60\%$, $\beta = 1 \times 10^{-2}$



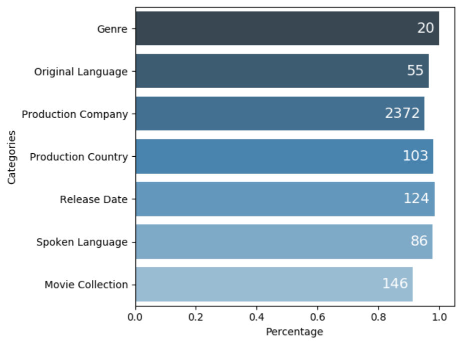
(b) $p = 60\%$, $\beta = 1 \times 10^{-3}$



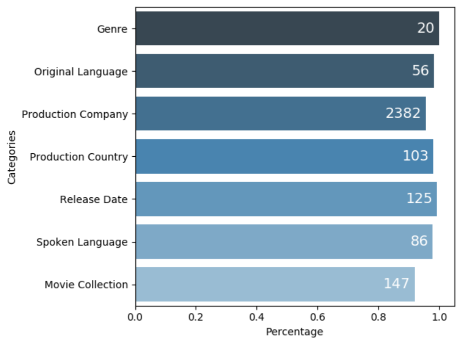
(c) $p = 80\%$, $\beta = 1 \times 10^{-2}$



(d) $p = 80\%$, $\beta = 1 \times 10^{-3}$



(e) $p = 95\%$, $\beta = 1 \times 10^{-2}$



(f) $p = 95\%$, $\beta = 1 \times 10^{-3}$

Figure 4.6: Bar plots showing the percentage of features selected for each category with CQFS at 60, 80 and 95% with different β and $s = 10^3$.

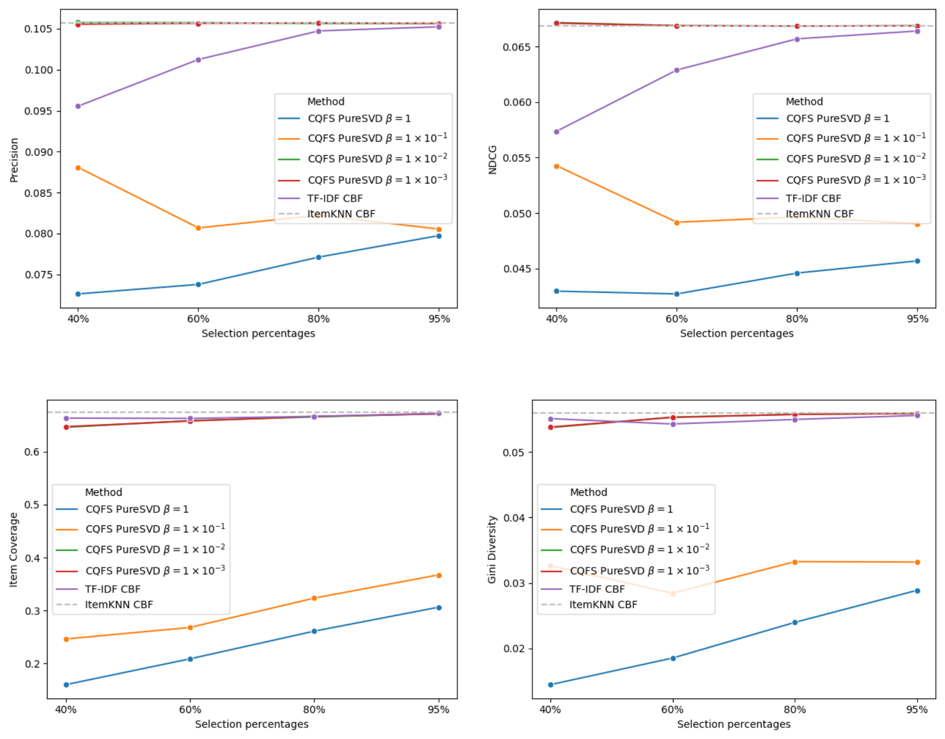


Figure 4.7: Plots comparing four metrics (precision, NDCG, item coverage and Gini diversity) on different experiments made on metadata features of The Movies Dataset with CQFS and TF-IDF, for the four tested selection percentages. ItemKNN CBF baseline is shown for reference.

recommendations. This is confirmed as well by the item coverage of CQFS, which is close to the one obtained by ItemKNN CBF, that instead uses all the features.

It should be noted, again, that these results were obtained by setting the strength parameter to 10^3 and testing two different values of β, 1×10^{-2} and 1×10^{-3}. However, if we analyze Figure 4.7, we can see how higher values of β (1 and 1×10^{-1}) result in lower accuracy, not even comparable with the baselines. This is probably due to the imbalance towards positive values of the FPM shown in Table 4.30 for these two values of β. In particular, almost every selection, using these parameter values, discards every genre feature, suggesting how much genres are relevant when making recommendations with The Movies Dataset.

Interaction	Meaning
0	Xing showed the item to the user.
1	The user clicked on the item.
2	The user bookmarked the item on Xing.
3	The user clicked a button to reply to or apply for the job posting.
4	The user eliminated the job posting from the suggestions.
5	The company showed interest into the user.

Table 4.32: Meaning of the interactions found in the original Xing data set.

4.4.3 Xing Challenge 2017

Xing Challenge 2017 is the data set used in the 2017 ACM RecSys Challenge[4]. Xing is a business social network, and this data set is a sample of Xing's data set. This means that it is not complete and it is artificially enriched with noise in order to anonymize data.

The data set contains the interactions between users of the platform and items, which represent job postings from companies. The interactions go from 0 to 5, with each value representing a different kind of interaction, as described in Table 4.32. In our experiments we do not use interactions with value 4 and 5, that are set to 0 in our URM, while we make use of interactions with value 0, 1, 2 and 3. Moreover, since these values do not represent explicit ratings, we set all the nonzero values to 1. The URM contains 4972869 interactions between 688645 users and 1306054 items. The distribution of the interactions with respect to the items is shown in Table 4.33. Because of the high number of items, many of which do not even have interactions, we preprocess the URM, in order to extract the 5-cores, which are all the users and items that have at least 5 interactions. Thus, we obtain a URM with 3357049 interactions (1.6 million less than the original one) between 190373 user and 88984 items.

The data set also contains features of the job postings. Example of these features are the industry and the discipline for the job posting, the required career level and the type of emplyment. The ICM of Xing Challenge 2017 is a boolean matrix containing around 32 million values between 1.3 million items and 107870 features. However, when applying the preprocessing to the URM, we need to keep only the items remained in the URM. Thus, the values in the ICM reduce to 2.7 million, between 88984 items and 43456 features. In Table 4.34 we show the number of features and corresponding occurrences in the ICM for the main categories of features in Xing Challenge 2017. Every feature category has at least one value for each item in the ICM.

Because of the low number of features belonging to most of the categories, we decided to solve CQFS for this data set with the quantum-only approach, directly using the QPU. This means that we had to work with a number

[4]http://www.recsyschallenge.com/2017/

	Before filtering		After filtering	
N	Items with N interactions	Items with **at least** N interactions	Items with N interactions	Items with **at least** N interactions
1	205243	508608	0	88984
5	22683	125048	16707	88984
10	5937	56963	4375	38927
15	2772	35152	2026	22923
20	1414	24449	1082	15336
25	927	18455	615	11150
30	645	14481	393	8518
35	463	11667	293	6772
40	333	9668	178	5546
45	250	8172	154	4688
50	182	7001	130	4018

Table 4.33: Number of items that have N or more interactions in Xing Challenge 2017, before and after preprocessing the URM.

	Before filtering		After filtering	
Category	Features	Occurrences	Features	Occurrences
Title Concept	11330	4018547	6019	298707
Industry	23	1306054	23	88984
Discipline	23	1306054	22	88984
Career Level	7	1306054	6	88984
Employment Type	5	1306054	5	88984
Is Paid	2	1306054	2	88984
Country	4	1306054	4	88984
Region	17	1306054	17	88984
Tags	96459	19172589	37358	1766133

Table 4.34: Number of features and ICM occurrences per feature category before and after preprocessing of Xing Challenge 2017.

of features complying with the limit imposed by the QPU. Therefore, we decided to use the following categories:

- industry: 23 features;
- discipline: 22 features;
- country: 4 features, consisting in Germany, Austria, Switzerland and a feature for all the other countries;
- region: 17 features, consisting in 16 regions of Germany and 1 feature indicating non-specified regions.

Since the total number of features from these categories is 66, while the limit of variables that can be embedded on the QPU is 65, we decided to drop 1 feature, the region representing non-specified regions. We chose this feature because it only indicates that the job posting is not for Germany, which can be inferred already from the country feature. Hence, this region feature is of limit usefulness, since it is not possible to use it as an indication of proximity. The resulting ICM has 346315 values between 88984 items and 65 features.

Data Splitting

As described in Section 3.2.1, we need to split the data set in order to properly train and evaluate our models. Analogously to the experiments with The Movies Dataset, we first perform a cold item split to obtain a testing set containing around 20% of all the interactions. Then, we perform two different types of split to obtain two different couples of validation and training sets. A random holdout split is needed to optimize the collaborative model, while another cold item split is needed to optimize the final model. The corresponding validation and training sets contain respectively around 10% and 70% of all the interactions. In Table 4.35 we report some data about the sets obtained from the two different splits. The observations made on the splits of The Movies Dataset are valid in this case as well.

Collaborative Filtering Models

Although we trained three collaborative filtering models (ItemKNN CF, PureSVD and RP^3_β) on Xing Challenge 2017, we decided, as for the experiments on metadata of The Movies Dataset, to use only one of them for CQFS. In particular, in this case we chose the ItemKNN CF model, which gave, on Xing Challenge 2017, better results than PureSVD and comparable with RP^3_β. The best hyperparameters found for this model during optimization are reported in Table 4.36. A comparison between this optimized model and an ItemKNN CBF model optimized on the random holdout validation split (best hyperparameters in Table 4.36) is shown in Table 4.37.

Set	Interactions	Covered users	Covered items
Training	2331120	190271	69586
Validation	354330	190043	52138
Testing	671050	170448	19395

(a) Random holdout split.

Set	Interactions	Covered users	Covered items
Training	2349927	190271	59891
Validation	335523	141185	9698
Testing	671050	170448	19395

(b) Cold item split.

Table 4.35: Interactions per split of Xing Challenge 2017.

Model	Hyperparameter	Value
ItemKNN CF	top-K	1000
	shrink	1000
	similarity	cosine
	normalize	true
	feature weighting	TF-IDF
ItemKNN CBF	top-K	153
	shrink	763
	similarity	cosine
	normalize	true
	feature weighting	BM25

Table 4.36: Best hyperparameters found for the ItemKNN CF model used for CQFS and for an ItemKNN CBF model optimized on the random holdout validation split in order to be compared with the collaborative method.

Model	PREC	REC	NDCG	MAP	IC	GD	MIL
ItemKNN CBF	0.0029	0.0167	0.0089	0.0053	0.7662	0.2401	0.9944
ItemKNN CF	0.1148	0.4894	0.4089	0.3661	0.7303	0.1393	0.9899

Table 4.37: Comparison between the quality of an ItemKNN CBF built with the chosen 65 features against the chosen collaborative method on the random holdout validation split.

Model	Hyperparameter	Value
ItemKNN CBF	top-K	460
	shrink	998
	similarity	cosine
	normalize	false
	feature weighting	none

Table 4.38: Best hyperparameters found for the ItemKNN CBF baseline of Xing Challenge 2017 experiments.

Again, we can see how the collaborative filtering method outperforms the accuracy of the content-based one. This means that we should be able to improve the accuracy of the content-based model by embedding collaborative information into it.

Baseline Models

As baseline models we used a standard ItemKNN CBF built from the ICM with all the 65 features, an ItemKNN CBF model built after selecting the highest weighted features by TF-IDF, with different selection percentages (40, 60, 80 and 95%), and a CFeCBF learning from the optimized ItemKNN CF model. The best hyperparameters found for these models are reported respectively in Table 4.38, Table 4.39 and Table 4.40. Again, we report only the non-fixed hyperparameters of CFeCBF.

Quantum Feature Selection

After optimizing the ItemKNN CF model, we used it to select features with CQFS. As opposed to The Movies Dataset experiments, in this case we directly used the QPU to search for a solution of the CQFS optimization problem. Therefore, we had to run a minor embedding procedure for each experiment, in order to map the variables of the CQFS optimization problem (the features) to the Chimera graph of the QPU. The result of the minor embedding of one of our experiments on Xing Challenge 2017 ($p = 40\%$, $\beta = 1$, $s = 1$) is shown in Figure 4.8. The physical qubits used to represent variables of the problem and their couplers are drawn in blue onto the underlying Chimera graph of the QPU, drawn in grey. One of the problem variables is highlighted in red, to show how one logical variable has to be represented by more than one physical qubits, because of the sparsity of the graph. As we can see, in the bottom left corner, some qubits are missing with respect to Figure 2.5, which instead shows the complete structure of a Chimera graph. This is due to the fact that not all the qubits and couplers are always available for computation[5].

[5]https://docs.dwavesys.com/docs/latest/c_qpu_0.html#dwave-short-qpu-architecture

Model	Hyperparameter	Value
TF-IDF CBF 40%	top-K	439
	shrink	383
	similarity	cosine
	normalize	false
	feature weighting	none
TF-IDF CBF 60%	top-K	557
	shrink	0
	similarity	cosine
	normalize	false
	feature weighting	none
TF-IDF CBF 80%	top-K	325
	shrink	399
	similarity	cosine
	normalize	false
	feature weighting	none
TF-IDF CBF 95%	top-K	489
	shrink	977
	similarity	cosine
	normalize	false
	feature weighting	none

Table 4.39: Best hyperparameters found for the TF-IDF CBF baselines of Xing Challenge 2017 experiments.

Model	Hyperparameter	Value
CFeCBF P	learning rate	1.93×10^{-4}
	L1 regularization	1×10^{-3}
	L2 regularization	1×10^{-1}
	epochs	35
	dropout	30%
	D initialization	random
	positive only D	true
	add zeros quota	5×10^{-1}

Table 4.40: Best hyperparameters found for the CFeCBF baseline of Xing Challenge 2017 experiments.

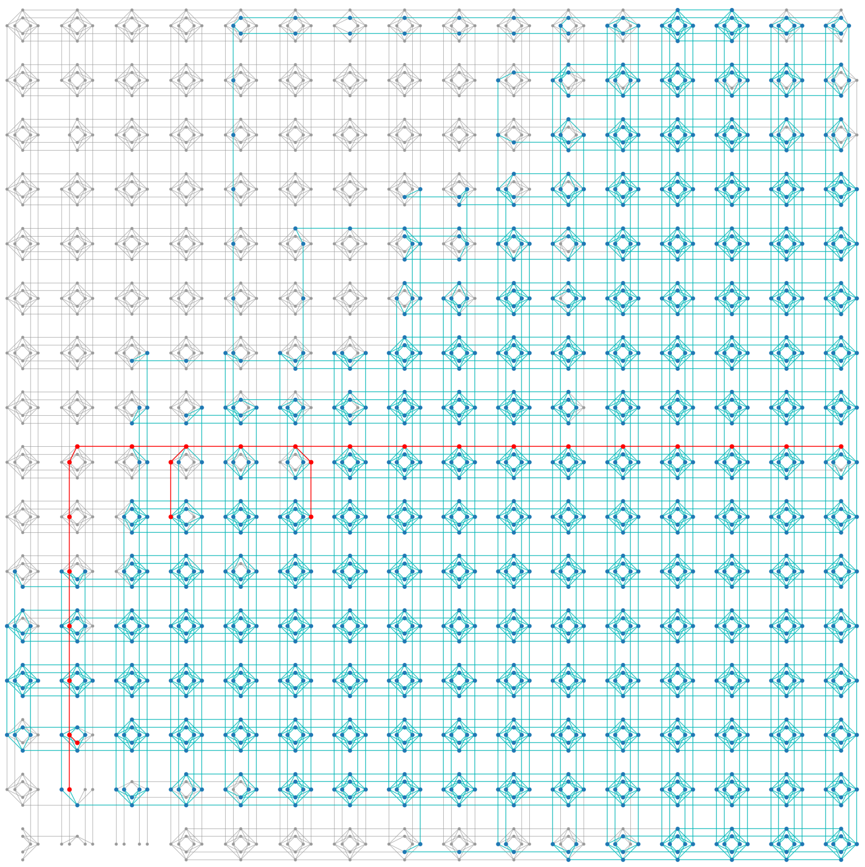


Figure 4.8: Resulting minor embedding of one of the Xing Challenge 2017 experiments into the Chimera graph.

Model	S nonzero values	S density	Covered CBF %
CBF dot product	4237352674	5.35×10^{-1}	-
ItemKNN CF	14508391	1.83×10^{-3}	0.33%

Table 4.41: Data about content and collaborative similarities for features of Xing Challenge 2017.

Parameter	Values
α	1
β	1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}
p	40%, 60%, 80%, 95%
s	1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}

Table 4.42: Tested parameters for the experiments on features of Xing Challenge 2017.

In Table 4.41 we can see a comparison between the collaborative similarity and the content similarity built with a dot product of the ICM with itself. Similarly to the experiments with metadata features of The Movies Dataset, the content model constains almost all the collaborative similarities (95.64%), which represent only a small fraction of the content model (0.33%). Again, the CQFS model should penalize with positive values almost all the content similarities, since they are absent from the collaborative model.

Because of the analogy with the experiments on metadata features of The Movies Dataset, we decided to carry out the quantum feature selection experiments with the same parameters of CQFS, shown in Table 4.42. After building the FPM for each experiment, we ran the quantum annealing procedure on the QPU with the default annealing schedule of $20\mu s$. We requested 1000 solutions from the QPU for each optimization problem, for a total of $20ms$ per problem. The results of the selection with the chosen parameters are reported in Figure 4.9, as heat maps of the ratios between the number of actually selected features and the desired number. Although the scenario is similar to the one of metadata features of The Movies Dataset, the results of the selection show a different behavior. Indeed, while increasing β helps selecting a number of features closer to the desired one, increasing s does not have any effect. In general, there is less consistency in selections with different parameters, with respect to what we expected from previous experiments.

To better understand why the selection gave these results, we report in Table 4.43 some statistics about the FPM of the problem, before applying the k-combinations constraint, for different values of β. We can see how

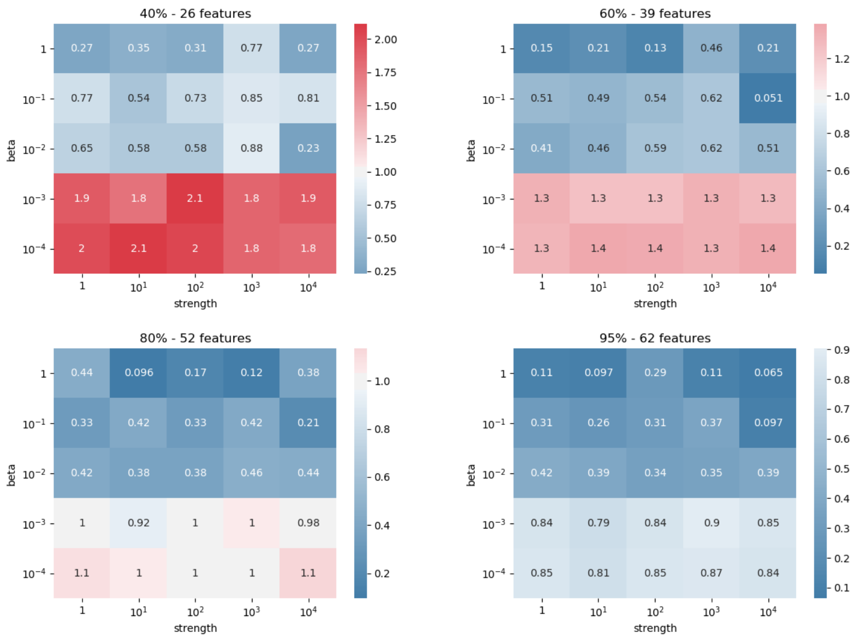


Figure 4.9: Ratios of the number of actually selected features on the desired number, for various parameters of CQFS applied to features of Xing Challenge 2017.

β	Coefficients	% negative	% positive
1	Linear	0.00	100.00
	Quadratic	0.00	100.00
1×10^{-1}	Linear	6.15	93.85
	Quadratic	0.00	100.00
1×10^{-2}	Linear	73.85	26.15
	Quadratic	2.52	97.48
1×10^{-3}	Linear	100.00	0.00
	Quadratic	96.13	3.87
1×10^{-4}	Linear	100.00	0.00
	Quadratic	100.00	0.00

(a) Percentages of negative and positive values in the FPM.

β	Coefficients	Min	Max	Mean
1	Linear	1946	4052351488	85418720.00
	Quadratic	1742	1156289408	14469818.00
1×10^{-1}	Linear	-6465.60	393784384	8164415.00
	Quadratic	161.60	112619688	1406726.25
1×10^{-2}	Linear	-421328.38	27689854	435510.94
	Quadratic	-34086.14	8183357.50	100272.65
1×10^{-3}	Linear	-8906682	-59.99	-337190.91
	Quadratic	-2284606.75	9380.88	-30365.82
1×10^{-4}	Linear	-12569218	-61.80	-414508.34
	Quadratic	-3328999.25	-2.98	-43429.25

(b) Minimum, maximum and mean of the values in the FPM.

Table 4.43: Statistics about values in linear and quadratic coefficients of the FPM, for each collaborative method, in Xing Challenge 2017 experiments.

coefficients in the FPM are greatly unbalanced towards positive values for higher β values. Indeed, when $\beta = 1$, all the FPM coefficients are positive, which would lead the QPU to not select any variable, without the k-combinations constraint. Even when β is low enough to obtain some negative quadratic coefficients, the imbalance grows in the opposite direction, with almost $\frac{3}{4}$ of the linear coefficients being negative. At $\beta = 1 \times 10^{-4}$ we have the complete opposite of $\beta = 1$, with every coefficient in the FPM being negative. Thus, we could associate the poor selection results with the difficulty of the problem, given by the higher imbalance of the objective function with respect to the previous experiments.

Model		PREC	REC	NDCG	MAP	IC	GD	MIL
Baselines	ItemKNN CBF	**0.0166**	**0.0611**	**0.0442**	**0.0272**	0.9996	0.3986	0.9823
	TF-IDF 95%	0.0160	0.0564	0.0379	0.0212	0.9996	0.3955	0.9813
	CFeCBF K	0.0108	0.0357	0.0225	0.0123	0.9892	0.3263	0.9795
CQFS K 40%	$\beta = 1 \times 10^{-3}$ $s = 1$	0.0157	0.0483	0.0346	0.0181	0.9959	0.3900	0.9821
CQFS K 95%	$\beta = 1 \times 10^{-3}$ $s = 10^3$	0.0154	0.0507	0.0295	0.0139	0.9999	0.3900	0.9805

Table 4.44: Evaluation metrics computed on Xing Challenge 2017 testing set for baseline algorithms and CQFS.

Final Content-based Model

Even though almost no selection returned the desired number of features, we decided to carry out the experiments building the final content-based model with the selection given by CQFS with paramters $\beta = 1 \times 10^{-3}$ and with two different strengths, $s = 1$ and $s = 1e3$. We chose these parameters because they seemed to achieve overall higher selection precision than the others, although the number of selected features is nearly the same for each selection percentage (around 50).

After setting to 0 the ICM values of the non-selected features, we built the final ItemKNN CBF model and optimized it with hyperparameter tuning on the ranges in Table 4.1. We report in Table 4.44 the results of the evaluation on a cutoff at 10 of the recommendations, along with the baseline results. The reported results are the best ones in terms of precision. Although the two CQFS evaluations are annotated with 40% and 95%, which are the selection percentages used in the two models, the corresponding content-based recommenders are built using respectively 50 and 56 features.

As we can see, no model is able to outperform the ItemKNN CBF baseline in accuracy metrics. The reason could be the already low number of features used for these experiments, with every item having exactly one feature from each of the four categories (industry, discipline, country and region). Thus, it seems that selecting only a subset of features, in this case, is not enough to improve the accuracy of the content-based model.

In Figure 4.10, we show the comparison of the CQFS and TF-IDF CBF models over four metrics, for the different selection percentages. The CQFS methods have parameter $\beta = 1 \times 10^{-3}$ and the two different strengths we chose. It should be noted how the increase of accuracy metrics of TF-IDF CBF is monotonic, with respect to the increasing percentages. This suggests that the lower accuracy of the CQFS methods against the ItemKNN CBF using all the features is not due to the embedding of collaborative information, but to the selection itself, reducing the number of features, as discussed before. Another interesting aspect is the high item coverage and

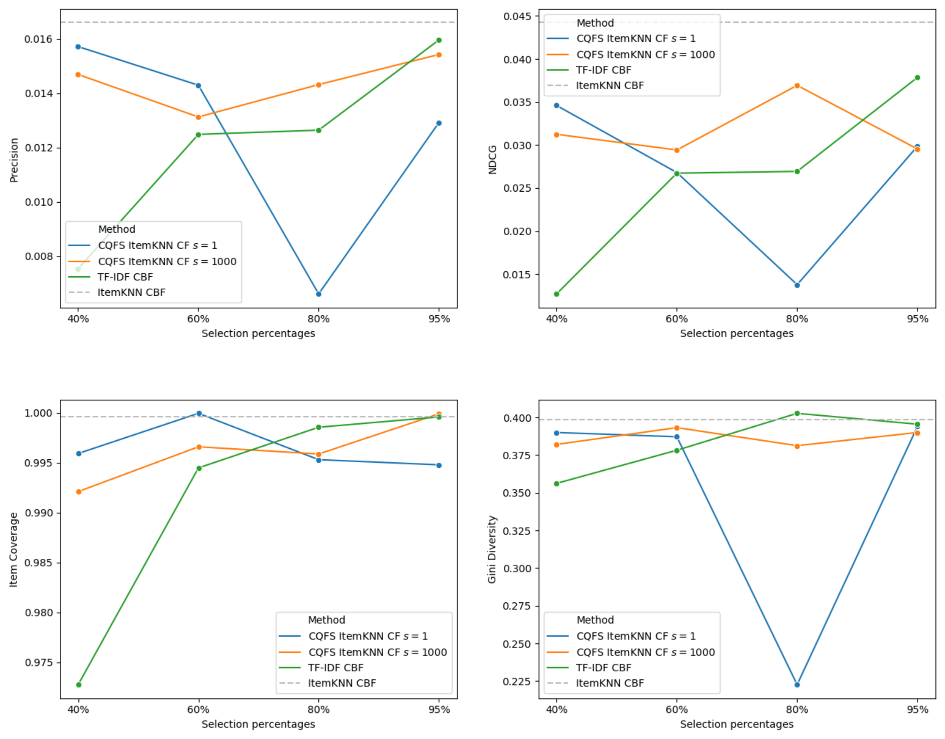


Figure 4.10: Plots comparing four metrics (precision, NDCG, item coverage and Gini diversity) on different experiments made on features of Xing Challenge 2017 with CQFS and TF-IDF, for the four tested selection percentages. ItemKNN CBF baseline is shown for reference.

diversity, which seem to be due to the particular subset of data we are using, since they are expressed by almost every model.

4.5 A Note on Computational Time Complexity

One of the main advantages of CQFS over other methods used to embed collaborative information into a content-based model, such as CFeCBF, is the required computational time to execute. Indeed, the solution of the optimization problem with a quantum annealer or a hybrid quantum-classical approach has a lower time complexity then machine learning methods.

In particular, the QPU has a constant complexity for each problem, which means that it has an overall linear complexity $O(n)$, function of the number of times n we ask the QPU to search for a solution of the given problem, and not of the number of variables. Instead, the hybrid algorithm imposes a time limit on the search procedure. This time limit is generally set to a linear function of the number of variables, but it can be set by the user as well.

For this reason, the biggest computational time of CQFS is required for building the FPM, a procedure whose most complex operation is a matrix multiplication, which has in general a polynomial complexity function of the dimensions of the involved matrices.

Chapter 5

Conclusions

In this thesis we proposed a new feature selection method for recommender systems, aimed at embedding collaborative information into a content-based model. We developed this method, Collaborative-driven Quantum Feature Selection, in order to improve recommendation results in a cold-start scenario, where pure collaborative methods are not effective. We compared our model with other techniques in three different sets of experiments, derived from two data sets, The Movies Dataset and Xing Challenge 2017. In particular, two sets of experiments were made on The Movies Dataset, using two different types of item features, movie credits and metadata.

Quantum feature selection was carried out with a hybrid quantum-classical approach for experiments on The Movies Dataset, and with a pure quantum annealing solution for experiments on Xing Challenge 2017. Both sets required some preprocessing in order to cope with the limits imposed by the current quantum technology.

We observed how feature selection with the hybrid approach gave consistent results with what we expected, while the pure quantum technique had some difficulties in correctly selecting the desired number of features on Xing Challenge 2017. We associated these difficulties with the very high imbalance between collaborative and content information.

With respect to the final content-based models built after feature selection, we obtained interesting results in terms of accuracy metrics. In experiments over credit features of The Movies Dataset, we obtained an improved precision using only 80%, or even 60%, of the given features. Moreover, we observed how the results from selection can change when using different collaborative filtering methods to build the CQFS optimization problem. In experiments over metadata features of The Movies Dataset, we obtained comparable results with the proposed baselines, even though CQFS needed only 40% of the features to be competitive. At last, in experiments over Xing Challenge 2017, we observed how selecting features did not improve accuracy over the baselines. Analyzing the behavior of the tested methods, we supposed that it was due to the already low number of used features, a number needed to be able to embed the corresponding optimization problem into the quantum processing unit performing quantum annealing.

In general, we obtained promising results, considering that physical quantum annealing is a relatively new technology. One of the most interesting aspects of this solution is the low computational complexity needed to solve the NP-hard optimization problem associated with CQFS. However, we observed how the hybrid quantum-classical approach is currently more viable in our scenario, because of its less strict limits on the number of problem variables. Therefore, future works include, first of all, improvements on the CQFS model, aimed at tackling problems that arose during the experiments, such as the high imbalance between collaborative and content information. Additionally, further development of quantum annealing technology could open up new possibilities for experiments carried out exclusively with pure quantum techniques. Indeed, a new quantum processing unit is close to be released later this year, offering more qubits and a denser coupling graph, extending the limits of the current generation we used in our work.

Bibliography

[1] Gediminas Adomavicius and YoungOk Kwon. "Improving Aggregate Recommendation Diversity Using Ranking-Based Techniques". In: *IEEE Trans. Knowl. Data Eng.* 24.5 (2012), pp. 896–911.

[2] Charu C. Aggarwal. *Recommender Systems - The Textbook.* Springer, 2016.

[3] Fabio Aiolli. "Efficient top-n recommendation for very large scale binary rated datasets". In: *Seventh ACM Conference on Recommender Systems, RecSys '13, Hong Kong, China, October 12-16, 2013.* Ed. by Qiang Yang et al. ACM, 2013, pp. 273–280.

[4] Sebastiano Antenucci et al. "Artist-driven layering and user's behaviour impact on recommendations in a playlist continuation scenario". In: *Proceedings of the ACM Recommender Systems Challenge, RecSys Challenge 2018, Vancouver, BC, Canada, October 2, 2018.* ACM, 2018, 4:1–4:6.

[5] Francisco Barahona. "A solvable case of quadratic 0-1 programming". In: *Discret. Appl. Math.* 13.1 (1986), pp. 23–26.

[6] Robert M Bell and Yehuda Koren. "Improved neighborhood-based collaborative filtering". In: *KDD cup and workshop at the 13th ACM SIGKDD international conference on knowledge discovery and data mining.* Citeseer. 2007, pp. 7–14.

[7] Cesare Bernardis, Maurizio Ferrari Dacrema, and Paolo Cremonesi. "A novel graph-based model for hybrid recommendations in cold-start scenarios". In: *CoRR* abs/1808.10664 (2018).

[8] Sergio Boixo et al. "Evidence for quantum annealing with more than one hundred qubits". In: *Nature physics* 10.3 (2014), pp. 218–224.

[9] Jun Cai, William G. Macready, and Aidan Roy. "A practical heuristic for finding graph minors". In: *CoRR* abs/1406.2741 (2014).

[10] Adriana Cano. "Rating aware feature selection in content-based recommender systems". In: *POLITesi* (2019).

[11] 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, San Francisco, CA, USA, August 13-17, 2016*. Ed. by Balaji Krishnapuram et al. ACM, 2016, pp. 785–794.

[12] Colin Cooper et al. “Random walks in recommender systems: exact computation and simulations”. In: *23rd International World Wide Web Conference, WWW ’14, Seoul, Republic of Korea, April 7-11, 2014, Companion Volume*. Ed. by Chin-Wan Chung et al. ACM, 2014, pp. 811–816.

[13] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. “Performance of recommender algorithms on top-n recommendation tasks”. In: *Proceedings of the 2010 ACM Conference on Recommender Systems, RecSys 2010, Barcelona, Spain, September 26-30, 2010*. Ed. by Xavier Amatriain et al. ACM, 2010, pp. 39–46.

[14] Edoardo D’Amico et al. “Leveraging Laziness, Browsing-Pattern Aware Stacked Models for Sequential Accommodation Learning to Rank”. In: *Proceedings of the Workshop on ACM Recommender Systems Challenge*. RecSys Challenge ’19. Copenhagen, Denmark: Association for Computing Machinery, 2019.

[15] Maurizio Ferrari Dacrema, Alberto Gasparin, and Paolo Cremonesi. “Deriving Item Features Relevance from Collaborative Domain Knowledge”. In: *Proceedings of the Workshop on Knowledge-aware and Conversational Recommender Systems 2018 co-located with 12th ACM Conference on Recommender Systems, KaRS@RecSys 2018, Vancouver, Canada, October 7, 2018*. Ed. by Vito Walter Anelli et al. Vol. 2290. CEUR Workshop Proceedings. CEUR-WS.org, 2018, pp. 1–4.

[16] Maurizio Ferrari Dacrema et al. “A Troubling Analysis of Reproducibility and Progress in Recommender Systems Research”. In: *CoRR* abs/1911.07698 (2019).

[17] Yashar Deldjoo et al. “Movie genome: alleviating new item cold start in movie recommendation”. In: *User Model. User Adapt. Interact.* 29.2 (2019), pp. 291–343.

[18] Vasil S. Denchev et al. “What is the Computational Value of Finite-Range Tunneling?” In: *Phys. Rev. X* 6 (3 2016), p. 031015.

[19] Lee R. Dice. “Measures of the Amount of Ecologic Association Between Species”. In: *Ecology* 26.3 (1945), pp. 297–302.

[20] Asmaa Elbadrawy and George Karypis. “User-Specific Feature-Based Similarity Models for Top-n Recommendation of New Items”. In: *ACM Trans. Intell. Syst. Technol.* 6.3 (2015), 33:1–33:20.

[21] Richard P Feynman. “Simulating physics with computers”. In: *Int. J. Theor. Phys* 21.6/7 (1982), pp. 467–488.

[22] Mouzhi Ge, Carla Delgado-Battenfeld, and Dietmar Jannach. “Beyond accuracy: evaluating recommender systems by coverage and serendipity”. In: *Proceedings of the 2010 ACM Conference on Recommender Systems, RecSys 2010, Barcelona, Spain, September 26-30, 2010.* Ed. by Xavier Amatriain et al. ACM, 2010, pp. 257–260.

[23] Fred W. Glover. “Future paths for integer programming and links to artificial intelligence”. In: *Comput. Oper. Res.* 13.5 (1986), pp. 533–549.

[24] Fred W. Glover. “Tabu Search - Part I”. In: *INFORMS J. Comput.* 1.3 (1989), pp. 190–206.

[25] Fred W. Glover. “Tabu Search - Part II”. In: *INFORMS J. Comput.* 2.1 (1990), pp. 4–32.

[26] Fred W. Glover, Gary A. Kochenberger, and Yu Du. “Quantum Bridge Analytics I: a tutorial on formulating and using QUBO models”. In: *4OR* 17.4 (2019), pp. 335–371.

[27] Isabelle Guyon and André Elisseeff. “An Introduction to Variable and Feature Selection”. In: *J. Mach. Learn. Res.* 3 (2003), pp. 1157–1182.

[28] Laszlo Gyongyosi and Sándor Imre. “A Survey on quantum computing technology”. In: *Comput. Sci. Rev.* 31 (2019), pp. 51–71.

[29] Jonathan L. Herlocker et al. “Evaluating collaborative filtering recommender systems”. In: *ACM Trans. Inf. Syst.* 22.1 (2004), pp. 5–53.

[30] Yifan Hu, Yehuda Koren, and Chris Volinsky. “Collaborative Filtering for Implicit Feedback Datasets”. In: *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy.* IEEE Computer Society, 2008, pp. 263–272.

[31] Mahdi Jalili et al. “Evaluating Collaborative Filtering Recommender Algorithms: A Survey”. In: *IEEE Access* 6 (2018), pp. 74003–74024.

[32] Karen Spärck Jones. “A statistical interpretation of term specificity and its application in retrieval”. In: *Journal of Documentation* 60.5 (2004), pp. 493–502.

[33] Tadashi Kadowaki and Hidetoshi Nishimori. “Quantum annealing in the transverse Ising model”. In: *Phys. Rev. E* 58 (5 1998), pp. 5355–5363.

[34] Scott Kirkpatrick, D. Gelatt Jr., and Mario P. Vecchi. “Optimization by Simmulated Annealing”. In: *Sci.* 220.4598 (1983), pp. 671–680.

[35] Gary A. Kochenberger et al. “The unconstrained binary quadratic programming problem: a survey”. In: *J. Comb. Optim.* 28.1 (2014), pp. 58–81.

[36] Yehuda Koren. “Factorization meets the neighborhood: a multifaceted collaborative filtering model”. In: *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008.* Ed. by Ying Li, Bing Liu, and Sunita Sarawagi. ACM, 2008, pp. 426–434.

[37] Yehuda Koren, Robert M. Bell, and Chris Volinsky. “Matrix Factorization Techniques for Recommender Systems”. In: *Computer* 42.8 (2009), pp. 30–37.

[38] T. Lanting et al. “Entanglement in a Quantum Annealing Processor”. In: *Phys. Rev. X* 4 (2 2014), p. 021041.

[39] Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. “Content-based Recommender Systems: State of the Art and Trends”. In: *Recommender Systems Handbook.* Ed. by Francesco Ricci et al. Springer, 2011, pp. 73–105.

[40] Hans Peter Luhn. “A Statistical Approach to Mechanized Encoding and Searching of Literary Information”. In: *IBM J. Res. Dev.* 1.4 (1957), pp. 309–317.

[41] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval.* Cambridge University Press, 2008.

[42] Catherine C. McGeoch et al. “Practical Annealing-Based Quantum Computing”. In: *Computer* 52.6 (2019), pp. 38–46.

[43] Nicholas Metropolis et al. “Equation of State Calculations by Fast Computing Machines”. In: *The Journal of Chemical Physics* 21.6 (1953), pp. 1087–1092.

[44] Xia Ning and George Karypis. “SLIM: Sparse Linear Methods for Top-N Recommender Systems”. In: *11th IEEE International Conference on Data Mining, ICDM 2011, Vancouver, BC, Canada, December 11-14, 2011.* Ed. by Diane J. Cook et al. IEEE Computer Society, 2011, pp. 497–506.

[45] Xia Ning and George Karypis. “Sparse linear methods with side information for Top-N recommendations”. In: *Proceedings of the 21st World Wide Web Conference, WWW 2012, Lyon, France, April 16-20, 2012 (Companion Volume).* Ed. by Alain Mille et al. ACM, 2012, pp. 581–582.

[46] Jorge Nocedal and Stephen J. Wright. “Quadratic Programming”. In: *Numerical Optimization.* 2nd ed. Springer, 2006. Chap. 16, pp. 448–492.

[47] Panos M. Pardalos and Somesh Jha. “Graph separation techniques for quadratic zero-one programming”. In: *Computers & Mathematics with Applications* 21.6 (1991), pp. 107–113.

[48] Panos M. Pardalos and Somesh Jha. “Complexity of uniqueness and local search in quadratic 0-1 programming”. In: *Oper. Res. Lett.* 11.2 (1992), pp. 119–123.

[49] Bibek Paudel et al. “Updatable, Accurate, Diverse, and Scalable Recommendations for Interactive Applications”. In: *ACM Trans. Interact. Intell. Syst.* 7.1 (2017), 1:1–1:34.

[50] Martin Pincus. “Letter to the Editor - A Monte Carlo Method for the Approximate Solution of Certain Types of Constrained Optimization Problems”. In: *Oper. Res.* 18.6 (1970), pp. 1225–1228.

[51] Ali Mustafa Qamar et al. “Similarity Learning for Nearest Neighbor Classification”. In: *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy.* IEEE Computer Society, 2008, pp. 983–988.

[52] Francesco Ricci et al., eds. *Recommender Systems Handbook.* 1st ed. Springer, 2011.

[53] Eleanor Gilbert Rieffel and Wolfgang Polak. “An introduction to quantum computing for non-physicists”. In: *ACM Comput. Surv.* 32.3 (2000), pp. 300–335.

[54] Stephen E. Robertson and Steve Walker. “Some Simple Effective Approximations to the 2-Poisson Model for Probabilistic Weighted Retrieval”. In: *Proceedings of the 17th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval. Dublin, Ireland, 3-6 July 1994 (Special Issue of the SIGIR Forum).* Ed. by W. Bruce Croft and C. J. van Rijsbergen. ACM/Springer, 1994, pp. 232–241.

[55] Badrul Munir Sarwar et al. “Item-based collaborative filtering recommendation algorithms”. In: *Proceedings of the Tenth International World Wide Web Conference, WWW 10, Hong Kong, China, May 1-5, 2001.* Ed. by Vincent Y. Shen et al. ACM, 2001, pp. 285–295.

[56] Badrul Sarwar et al. “Application of dimensionality reduction in recommender system-a case study”. In: *Proc. KDD Workshop on Web Mining for e-Commerce: Challenges and Opportunities (WebKDD).* ACM, 2000.

[57] Guy Shani and Asela Gunawardana. “Evaluating Recommendation Systems”. In: *Recommender Systems Handbook.* Ed. by Francesco Ricci et al. Springer, 2011, pp. 257–297.

[58] Mohit Sharma et al. “Feature-based factorized Bilinear Similarity Model for Cold-Start Top-n Item Recommendation”. In: *Proceedings of the 2015 SIAM International Conference on Data Mining, Vancouver, BC, Canada, April 30 - May 2, 2015*. Ed. by Suresh Venkatasubramanian and Jieping Ye. SIAM, 2015, pp. 190–198.

[59] Peter W. Shor. “Algorithms for Quantum Computation: Discrete Logarithms and Factoring”. In: *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*. IEEE Computer Society, 1994, pp. 124–134.

[60] Peter W. Shor. “Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer”. In: *SIAM J. Comput.* 26.5 (1997), pp. 1484–1509.

[61] Amos Tversky. “Features of similarity.” In: *Psychological Review* 84.4 (1977), pp. 327–352.

[62] Chong Wang and David M. Blei. “Collaborative topic modeling for recommending scientific articles”. In: *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*. Ed. by Chid Apté, Joydeep Ghosh, and Padhraic Smyth. ACM, 2011, pp. 448–456.

[63] Jun Wang et al. “Probabilistic relevance ranking for collaborative filtering”. In: *Inf. Retr.* 11.6 (2008), pp. 477–497.

[64] Tao Zhou et al. “Solving the apparent diversity-accuracy dilemma of recommender systems”. In: *Proceedings of the National Academy of Sciences* 107.10 (2010), pp. 4511–4515.