Comparing Simple and Hybrid Collaborative Filtering Within an Artificial Recommendation Problem

Project Report for Course on Data Mining, Winter Term 2021/'22

Michael Vrazitulis
University of Trento
Matricola: 229531
Erasmus exchange student
MSc Language Science and Technology (2nd year)

1 INTRODUCTION

Recommendation systems are an everyday phenomenon in our modern digitised world. Their use cases range broadly, from e-commerce to media and entertainment, from social media to banking and finance, among other domains. One further domain potentially profiting from such systems is the medical sector. Specifically, it seems that doctors who choose adequate therapies for their patients are doing so on the basis of considering a variety of different factors, such as the medical history of a particular patient or the type of medical condition they are struggling with. Given enough data on such factors, one could conceptualise this process as yet another recommendation problem. This is precisely the aim of the present project. But since real medical data of the kind that would be needed to test a recommendation algorithm on is not so readily available, here, instead, we use artificially generated datasets that are constrained in certain ways. Thus, while the present project's focus clearly lies more on methodological considerations, independent of any specific application, it can also partially be viewed as aiming for a proof-of-concept for tackling real recommendation problems with similar properties as the examined artificial one.

2 RELATED WORK

The main theoretical foundations regarding recommendation systems, which this project is based on, are laid out by [1]. An examination of various collaborative-filtering approaches, including hybrid ones, i.e. ones that are also incorporating aspects from content-based systems, was carried out by [6]. With respect to clustering in non-Euclidean spaces, [4] propose the PAM algorithm, a heuristic solution to the k-medoids problem. That algorithm was further improved by [5] into a qualitatively equivalent version, FasterPAM, but with remarkable improvements regarding run-time complexity reduction. For both PAM (or FasterPAM) there is an approximate adaptation, CLARA (or FasterCLARA), conceived by [3]. That adaptation is much more scalable to very large datasets, as it requires PAM-clustering only a selected number of subsamples of a dataset. Useful considerations about the intricacies of clustering high-dimensional data, concerning, for example, the appropriate choice of a distance metric, are provided by [2].

3 PROBLEM STATEMENT

Let us first establish the following definitions:

(1) $\mathcal{C}$, $\mathcal{T}$ and $\mathcal{P}$ are sets of entities.
(2) Σ is the set of all possible alphanumeric strings.
(3) Every entity $c \in \mathcal{C}$ is a set of $\langle name, value \rangle$ pairs, where *name* and *value* are strings $\in \Sigma$.
(4) Every entity $t \in \mathcal{T}$ is a set of $\langle name, value \rangle$ pairs, where *name* and *value* are strings $\in \Sigma$.
(5) Every entity $p \in \mathcal{P}$ is a set of $\langle name, value \rangle$ pairs, where *name* is a string and *value* is either,
   - a string $\in \Sigma$,
   - a number $\in \mathbb{R}$,
   - a set of entities, consisting of $\langle name', value' \rangle$ pairs with $name' \in \Sigma$ and $value' \in \Sigma \cup \mathbb{R} \cup \{\infty\}$.
(6) $\mathcal{E}$ is a set of entities, with $\mathcal{E} = \mathcal{C} \cup \mathcal{T} \cup \mathcal{P} \cup \{value \mid \langle name, value \rangle \in p$ for some $p \in \mathcal{P}$ **and** $value \notin \Sigma \cup \mathbb{R}\}$.
(7) $id \subseteq \mathcal{E} \times \Sigma$ is an injective function that maps every entity $e \in \mathcal{E}$ to a unique identifier string.
(8) A dataset $\mathcal{D}$ is a tuple, with $\mathcal{D} = \langle \mathcal{C}, \mathcal{T}, \mathcal{P}, id \rangle$.

For our particular recommendation problem, we also make these further assumptions:

(9) For any string $\sigma \in \Sigma$ and any entity $e \in \mathcal{E}$, there is **at most** one pair $\langle name, value \rangle \in e$ such that $\sigma = name$.
(10) For all $c \in \mathcal{C}$, if there is a string $\sigma \in \Sigma$ such that $\sigma = name$ for some pair $\langle name, value \rangle \in c$, then for all $c' \in \mathcal{C}$, there is a pair $\langle name', value' \rangle \in c'$ such that $\sigma = name'$.
(11) For all $t \in \mathcal{T}$, if there is a string $\sigma \in \Sigma$ such that $\sigma = name$ for some pair $\langle name, value \rangle \in t$, then for all $t' \in \mathcal{T}$, there is a pair $\langle name', value' \rangle \in t'$ such that $\sigma = name'$.
(12) For all $p \in \mathcal{P}$, if there is a string $\sigma \in \Sigma$ such that $\sigma = name$ for some pair $\langle name, value \rangle \in p$, then for all $p' \in \mathcal{P}$, there is a pair $\langle name', value' \rangle \in p'$ such that $\sigma = name'$.
(13) For all $p \in \mathcal{P}$, if $value \in \Sigma$ for some pair $\langle name, value \rangle \in p$, then for all $p' \in \mathcal{P}$, for all pairs $\langle name', value' \rangle \in p'$, $value' \in \Sigma$.
(14) For all $p \in \mathcal{P}$, if $value \in \mathbb{R}$ for some pair $\langle name, value \rangle \in p$, then for all $p' \in \mathcal{P}$, for all pairs $\langle name', value' \rangle \in p'$, $value' \in \mathbb{R}$.
(15) For all $p \in \mathcal{P}$, if $value \subseteq \mathcal{E}$ for some pair $\langle name, value \rangle \in p$, then for all $p' \in \mathcal{P}$, for all pairs $\langle name', value' \rangle \in p'$, $value' \subseteq \mathcal{E}$.
(16) Every entity $p \in \mathcal{P}$ has the structure,

$$p = \{ \langle \texttt{"conditions"}, \{pc_{p,1}, \ldots, pc_{p,n}\} \rangle, \langle \texttt{"trials"}, \{tr_{p,1}, \ldots, tr_{p,m}\} \rangle, \ldots \}$$

where any entity $pc_{p,i}$ $(1 \leq i \leq n)$ has the structure,

$$pc_{p,i} = \{ \langle \texttt{"diagnosed"}, d \in \mathbb{N}_0 \rangle, \langle \texttt{"cured"}, q \in \mathbb{N}_0 \cup \{\infty\} \rangle, \langle \texttt{"kind"}, \chi \in \text{range}(id) \rangle \}$$
with $\chi = id(c)$ for some entity $c \in C$,

and any entity $tr_{p,j}$ $(1 \leq j \leq m)$ has the structure,

$$tr_{p,j} = \{ \langle \texttt{"start"}, a \in \mathbb{N}_0 \rangle, \langle \texttt{"end"}, b \in \mathbb{N}_0 \rangle, \langle \texttt{"condition"}, \kappa \in \text{range}(id) \} \rangle, \langle \texttt{"therapy"}, \theta \in \text{range}(id) \} \rangle, \langle \texttt{"successful"}, s \in \mathbb{R} \rangle \}$$
with $0 \leq s \leq 100$ **and** $\kappa = id(pc_{p,i})$ for some entity $pc_{p,i}$ **and** $\theta = id(t)$ for some entity $t \in \mathcal{T}$ **and** if $s = 100$, then $\langle \texttt{"cured"}, \infty \rangle \notin id^{-1}(\kappa)$ **and** $d \leq a \leq b \leq q$ where $\langle \texttt{"diagnosed"}, d \rangle \in id^{-1}(\kappa)$, $\langle \texttt{"cured"}, q \rangle \in id^{-1}(\kappa)$.

(17) For all $p \in \mathcal{P}$, if $value \subseteq \mathcal{E}$ for some pair $\langle name, value \rangle \in p$, then $name \in \{\texttt{"conditions"}, \texttt{"trials"}\}$.

We might refer to entities $c \in C$ as conditions, to entities $t \in \mathcal{T}$ as therapies, and to entities $p \in \mathcal{P}$ as patients.

The present recommendation problem is to algorithmically implement a function which, given a dataset $\mathcal{D}$, takes any patient and any of their yet-uncured conditions as its input, i.e. an entity $p \in \mathcal{P}$ and some $pc_{p,i} \in value$ where $\langle \texttt{"conditions"}, value \rangle \in p$ and $\langle \texttt{"cured"}, \infty \rangle \in pc_{p,i}$, and returns an ordered list of five therapies as its output, i.e. five distinct entities $t_1 \ldots t_5 \in \mathcal{T}$. Those five therapies shall be the most **utile** ones for the patient and condition in question. Utility, here, describes the likelihood of quickly and successfully treating the patient's condition which one would associate with administering a novel trial of that therapy, i.e. an inserting a new entity $tr_{p,m+1}$ into the value side of $\langle \texttt{"trials"}, \{tr_{p,1}, \ldots, tr_{p,m}\} \rangle$. The fundamentally underlying assumption is that all the values present in $\mathcal{D}$ are (fictional) empirical data and thus might display exploitable statistical patterns by which that likelihood could be inferred. For our interpretation of the problem, we define the utility criterion which the desired function shall maximise as follows:

$$utility(tr_{p,j}) = \frac{s}{1 + ln(1 + b - a)} \tag{18}$$

Here, the variables a, b, and s describe the numerical values associated with the attributes `"start"`, `"end"`, and `"successful"` within any trial entity as defined above in (16). Thus, the best possible outcome would be $utility(tr_{p,j}) = 100$, which would require $s = 100$ and $a = b$. With the established real-world analogy, this would be a therapy trial that cures the patient fully and immediately within a single day, assuming the values a and b to be representing dates (say e.g. 31/12/2021 as a value 44,559 or 19/01/2022 as a value 44,578) The worst possible outcome, $utility(tr_{p,j}) = 0$, would entail $s = 0$, corresponding to a therapy trial that completely failed to provide any remedy to the patient, regardless of its duration.

4 SOLUTION

4.1 Storing Utilities in a 3D Tensor

When designing a recommendation system, it is often useful to conceptualise the problem as a large sparse matrix, referable to as a utility matrix (see [1]). In a classical example such as recommending a movie or some other consumable item to a user of a certain platform, based e.g. on ratings similar users have assigned to similar items in the past, this conceptualisation is quite straightforward: The two dimensions of the matrix correspond to users and items, respectively, and the matrix stores all real utility (e.g. rating) values for any $\langle user, item \rangle$ combination for which such information is available. The task then is to, quite literally, fill the gaps, i.e. to find an algorithm that estimates the expected utility of any $\langle user, item \rangle$ combination for which a real utility value does not yet exist. However, for the present problem, two dimensions are not enough to represent the utility space we are exploring, as we have not only patients (a.k.a. users) and therapies (a.k.a. items), but also an additional categorical variable, conditions. Instead, we might construct a three-dimensional **utility tensor**:

$$UT(\mathcal{D}) \in \mathbb{R}^{|\mathcal{P}| \times |C| \times |\mathcal{T}|} \tag{19}$$

Let us also establish the notation $UT(D)_{p,c,t}$ for referring to any cell in the tensor that corresponds to some triple $\langle p \in \mathcal{P}, c \in C, t \in \mathcal{T} \rangle$. Then, to determine the initial entry for a cell, we compute:

$$\begin{aligned} UT(\mathcal{D})_{p,c,t} = max(\{0\} \cup \{utility(tr_{p,j}) \mid 1 \leq j \leq |value| \\ \text{with } \langle \texttt{"trials"}, value \rangle \in p \\ \textbf{and } \langle \texttt{"kind"}, id(c) \rangle \in id^{-1}(\kappa) \\ \text{with } \langle \texttt{"conditions"}, \kappa \rangle \in tr_{p,j} \\ \textbf{and } \langle \texttt{"therapy"}, id(t) \rangle \in tr_{p,j}\}) \end{aligned} \tag{20}$$

In other words, we compute the utilities of all trials of therapy t for treating the kind of condition c that patient p has undergone in the past, and select the maximum utility among those found, since it could be the case that a patient is treated with the same therapy with respect to the same kind of condition more than once during their medical history. Any cell for which a real value is missing will be assigned 0 by default.

Even though this depends on the specific dataset, on average, we can expect such a utility tensor to be only very sparsely populated with real values for the majority of all possible $\langle p, c, t \rangle$ triples. Especially the number of patients, $|P|$, might be way too large when dealing with a dataset that roughly conforms with the real-world analogy's intuition that patients are infinitely scalable in their number, whereas the numbers of possible (medical) conditions and therapies will stay comparably small across datasets. That is why we could choose to first summarise groups of similar patients into clusters, thereby reducing data sparsity, before actually computing any utility estimates. Another motivation for **clustering patients** (rather than conditions or therapies) is that, as defined in Section 3, patient entities have a quite informative structure already by default (i.e. the obligatory `"conditions"` and `"trials"` attributes), thus lending themselves quite nicely for a reliable computation of distances that will underlie the clustering process.

4.2 Measuring Similarity Among Patients

As a distance metric for determining the similarity or dissimilarity between some vector representations $\overrightarrow{v_1}$ and $\overrightarrow{v_2}$ of two patients $p_1, p_2 \in \mathcal{P}$, we pick the **cosine distance**, as it has been argued to yield slightly more robust results than e.g. Euclidean distance in

the case of sparse, high-dimensional representations (see [2]):

$$distance(\overrightarrow{v_1}, \overrightarrow{v_2}) = \frac{\overrightarrow{v_1} \cdot \overrightarrow{v_2}}{\|\overrightarrow{v_1}\| \|\overrightarrow{v_2}\|} \tag{21}$$

Now, the obvious question arises, of course, how to construct such vector representations for patient entities. Within the framework of an approach known to theory as **collaborative filtering**, we would simply construct such a vector out of the utility values associated with any specific patient in the utility matrix or tensor. In our case, for some patient $p \in \mathcal{P}$, this would be a 'slice' of $UT(\mathcal{D})$, i.e. the matrix $UT(\mathcal{D})_p$. Thus, a simple vector for patient p would be the matrix matrix $UT(\mathcal{D})_p$ flattened into one dimension. At this point, we are also going to normalise the utility values to range between 0 and 1, with the minimum and maximum values being scaled to 0 and 1 respectively (below, $\overrightarrow{1}_n$ denotes a vector of length n and with all its entries being 1):

$$simpleVec(p) = \frac{flatten[UT(\mathcal{D})_p] - min[UT(\mathcal{D})] \cdot \overrightarrow{1}_{|\mathcal{P}||C||\mathcal{T}|}}{max[UT(\mathcal{D})] - min[UT(\mathcal{D})]} \tag{22}$$

4.3 Hybrid Vector Representations of Patients

Using the kind of representation just introduced in (22) for clustering patients can serve as one of the baselines for subsequent evaluation. However, for the full approach we want to introduce here, we are going to extend this representation by an encoding of other features present in any entity $p \in \mathcal{P}$. One challenge that arises at this point is that the full structure of patient entities is left unspecified, as described in Section 3. This means that depending on the specific dataset we are looking at, we can expect patient entities to display various other attributes than the pre-specified `"conditions"` and `"trials"`. By our definition of the problem, the values of such further attributes of a patient can be either categorical, i.e. $\in \Sigma$, or numerical, i.e. $\in \mathbb{R}$.

The selection of all **further relevant attributes** that a patient might have is carried out as described by Algorithm 1 below. Note that in addition to simply selecting all attributes that are not either `"conditions"` or `"trials"`, we also group attributes by whether they are numerical or categorical, and discard any attribute for which it is the case that either,

- it has the same value for all patients
- it is categorical and has a unique value for each patient
- it is numerical and its values strongly correlate with those of an already selected numerical attribute (Pearson's correlation coefficient $\rho \geq 0.9$).

We will then normalise the values of the relevant numerical attributes to range between 0 and 1, in the same way as we did for the utilities. Having accomplished that, we should take care of one last, but crucial missing puzzle piece: The information about the date of diagnosis of any condition any patient had caught in the past (diagnosis recency), alongside the ending date of any therapy trial any patient had to undergo (trial recency). We might, for example, want to distinguish two patients from each other who have the same kind of condition in their medical record, but during more than twenty years apart. We also would like to distinguish two patients who had the same kind of condition around the same time, but were subjected to the same set of therapies in different orders. We can find such information encoded in form of the $\langle$`"diagnosed"`$, d\rangle$ values of patient-condition entities and the $\langle$`"end"`$, b\rangle$ values of trial entities, respectively. Both trial-recency and diagnosis-recency values will also be normalised to range within the boundaries of the interval between 0 and 1.

We can now proceed to construct a more informative vector representation for each patient than the one suggested in (22). Note that this new vector representation displays the exact same values as $simpleVec(p)$ in its first $|C||\mathcal{T}|$ dimensions, but has a substantial number of further dimensions which will store the entity attribute values which were just discussed:

(23) $hybridVec(p)$ is a vector with the following dimensions:

- $1, \ldots, |C||\mathcal{T}|$: **utility values**
- $|C||\mathcal{T}| + 1, \ldots, 2|C||\mathcal{T}|$: **trial-recency values**

Algorithm 1 Find further relevant attributes of patient entities

Input: $\mathcal{P}$ is the set of patient entities
Output: $catgAttributes$ are relevant categorical attributes
$numrAttributes$ are relevant numerical attributes

$p_0 \leftarrow$ an arbitrary entity $p \in \mathcal{P}$
$catgAttributes \leftarrow \{\}$
$numrAttributes \leftarrow \{\}$
$otherLists \leftarrow \{\}$
for each $\langle name, value\rangle \in p_0$ **do**
  if $name \in$ {`"conditions"`, `"trials"`} **then**
    continue
  end if
  if $value \in \Sigma$ **then** ▷ $name$ is a categorical attribute
    $catgValsSet \leftarrow \{\}$
    for each $p \in \mathcal{P}$ **do**
      select $value'$ **with** $\langle name, value'\rangle \in p$
      $catgValsSet \leftarrow catgValsSet \cup \{value'\}$
    end for
    if $|catgValsSet| \notin \{1, |\mathcal{P}|\}$ **then**
      $catgAttributes \leftarrow catgAttributes \cup \{name\}$
    end if
  else
    assert $value \in \mathbb{R}$ ▷ $name$ is a numerical attribute
    $numrValsList \leftarrow \langle\rangle$
    for each $p \in \mathcal{P}$ **do**
      select $value'$ **with** $\langle name, value'\rangle \in p$
      $numrValsList \leftarrow numrValsList \oplus \langle value'\rangle$
    end for
    if $min(numrValsList) = max(numrValsList)$ **then**
      continue
    else if $\exists l.\ l \in otherLists \wedge \rho_{numrValsList,\, l} \geq 0.9$ **then**
      continue
    end if
    $otherLists \leftarrow otherLists \cup \{numrValsList\}$
    $numrAttributes \leftarrow numrAttributes \cup \{name\}$
  end if
end for
return $\langle catgAttributes, numrAttributes\rangle$

- $2|C||\mathcal{T}|+1, \ldots, 2|C|(|\mathcal{T}|+1)$: **diagnosis-recency values**
- $2|C|(|\mathcal{T}|+1)+1, \ldots, 2|C|(|\mathcal{T}|+1)+|numrAttributes|$: **numerical attribute values**
- For every relevant **categorical attribute,** where $hash \in (\Sigma \times \Sigma) \times (\mathbb{N}^+ \setminus \{1, \ldots, 2|C|(|\mathcal{T}|+1)+|numrAttributes|\})$ is some hash function mapping to various dimensions: $hash(\langle name, value \rangle)$: **set to 1**

Essentially, the two alternative functions for constructing vector representations for patients, *simpleVec* and *hybridVec* are also mirroring the previously mentioned conceptual distinction often made in recommendation-system theory (see [6]): Representing users (here, patients) solely by the real utility values they have displayed for various items in the past, but not by any other user-specific features, is the basis of any plain **collaborative-filtering** method. Instead, we opt for a **hybrid approach**, i.e. one that is based on collaborative filtering, but extends the utility vectors representing users with available feature information for the clustering process, thus also resembling **content-based** methods.

4.4 Clustering in a Non-Euclidean Space

Finally, we can move on to the actual clustering process which we need in order to tackle the data sparsity problem regarding our utility data, as laid out in Section 3. Since with cosine distance we picked a non-Euclidean underlying similarity/dissimilarity metric, we have to apply a clustering method suited for non-Euclidean spaces. One such method is ***k*-medoids.** As k-medoids can't be solved optimally in less than polynomial time, the PAM algorithm (see [4]) was devised. That algorithm employs a greedy search which may not necessarily find the globally optimal solution to the k-medoids problem, but can run much faster in return. Whereas the run-time complexity of plain PAM is $\boldsymbol{O(k(N-k)^2)}$, with N being the number of data points and k being the apriori chosen number of clusters, a more recent, further developed version of the algorithm, FasterPAM (find the full algorithm specification in [5]), offers a run-time complexity of just $\boldsymbol{O(N^2)}$ instead, while retaining the same quality of output. However, quadratic complexity is still not good enough when tackling a clustering problem as the present one, with our $N = |\mathcal{P}|$ expected to be scalable up to very large numbers. For these kinds of use cases, a heuristic extension to the PAM algorithm has been devised: The CLARA algorithm (see [3]) applies PAM not to the whole dataset, but instead to a selected number of n random samples drawn from the dataset, each of a fixed size s. It uses the cost value which is part of PAM's output in order to estimate which of the n resulting clusters, being associated with minimal cost, best fits the full-sized dataset. Thus, run-time complexity reduces to $\boldsymbol{O(ks^2+k(N-k))}$, i.e. is now just linear instead of quadratic with respect to N. A summary of the steps is also given in Algorithm 2 below. Note that since original CLARA algorithm was using plain PAM as a module internally and that here the equivalent, but more efficient FasterPAM module is used instead, this newly combined version of the algorithm is often dubbed **FasterCLARA**, by analogy.

Applying this algorithm to the present problem, we choose the CLARA parameters $n = 5$ (a common choice) and $k = 100$, $s = 500$ (rather arbitrary choices). In summary, we now expect to obtain a representation of patients divided into 100 different clusters.

Algorithm 2 FasterCLARA (based on [3] and [5])

Input: X is a set of N points ▷ here, set of patient vectors
k is the desired number of clusters ▷ here, $k = 100$
n is the chosen number of samples ▷ here, $n = 5$
s is the chosen size for each sample ▷ here, $s = 500$
Output: $\mathcal{M}$ is a set of k cluster medoids
$\overrightarrow{\mathsf{A}}$ is an N-vector of cluster assignments
for $i \leftarrow 1$ to n **do**
  $S_i \leftarrow \text{RandomSample}(X, s)$
  $\langle a_i, m_i \rangle \leftarrow \underline{\text{FasterPAM}}(S_i, k)$
end for
$\mathcal{M} \leftarrow \text{Best}(m_1, \ldots, m_n)$
$\overrightarrow{\mathsf{A}} \leftarrow \text{AssignPointsToNearestMedoid}(X, \mathcal{M})$
return $\langle \mathcal{M}, \overrightarrow{\mathsf{A}} \rangle$

4.5 Condensing the Utility Tensor

Algorithm 3 Get condensed utility tensor

Input: $\mathcal{M}$ is a set of k cluster medoids
$\overrightarrow{\mathsf{A}}$ is an N-vector of cluster assignments
$\mathcal{D} = \langle C, \mathcal{T}, \mathcal{P}, id \rangle$ is our dataset
$UT(\mathcal{D})$ is the raw utility tensor of $\mathcal{D}$
Output: $CUT(\mathcal{D})$ is the condensed utility tensor of $\mathcal{D}$
assert $\mathcal{M} \subseteq \mathcal{P}$ **and** $|\overrightarrow{\mathsf{A}}| = |\mathcal{P}|$
$agglomeratedU \leftarrow$ a mapping from medoids $\in \mathcal{M}$ to empty lists
for $i \leftarrow 1$ to $|\mathcal{P}|$ **do**
  $m \leftarrow \mathsf{A}_i$ ▷ medoid the ith patient was assigned to
  $agglomeratedU_m \leftarrow agglomeratedU_m \oplus UT(\mathcal{D})_m$
end for
$CUT(\mathcal{D}) \leftarrow$ an all-zero tensor $\in \mathbb{R}^{|\mathcal{M}| \times |C| \times |\mathcal{T}|}$
for each $m \in \mathcal{M}$ **do**
  for each $patientMatrix \in agglomeratedU_m$ **do**
    $CUT(\mathcal{D})_m \leftarrow CUT(\mathcal{D})_m + \frac{patientMatrix}{|agglomeratedU_m|}$
  end for
end for
return $CUT(\mathcal{D})$

The next step is to make use of the obtained clusters by 'condensing' the raw utility tensor which had been computed as given in (20). Algorithm 3 describes this process. The core idea is to get a new, smaller tensor of size of only $k \times |C| \times |\mathcal{T}|$ instead of the raw $|\mathcal{P}| \times |C| \times |\mathcal{T}|$, where utility values are stored **averaged by cluster** instead of individually by patient.

4.6 Retrieving Recommendable Therapies

Finally, we can get to the computation of predicted utilities, which will allow us to recommend the five best therapies simply by choosing those with the top five highest predicted utility values. As shown in Algorithm 3, computing the predicted utility value for any therapy amounts to calculating a weighted sum of its utility values

Algorithm 4 Compute predicted utilities under hybrid approach

Input: $\langle p,\ pc_{p,input}\rangle$ are the patient & condition of interest
m_p is the cluster medoid least distant to p
$\mathcal{D} = \langle C, \mathcal{T}, \mathcal{P}, id\rangle$ is our dataset
$CUT(\mathcal{D})$ is the condensed utility tensor of $\mathcal{D}$
Output: $predU$ maps therapies to their predicted utilities

$catgAttributes \leftarrow$ GetRelevantCategoricalAttributes(C)
$biCo \leftarrow max(\{\binom{|C|}{|\text{SetOfValues}(attr)|} \mid attr \in catgAttributes\})$
select $mostInfoAttr$ **with** $biCo = \binom{|C|}{|\text{SetOfValues}(mostInfoAttr)|}$
$sameValConds \leftarrow \{c' \mid c' \in C \backslash \{c\}$ **and** $value = value'$ where $(mostInfoAttr, value) \in c, (mostInfoAttr, value') \in c')\}$
$predU \leftarrow$ a mapping from therapies $\in \mathcal{T}$ to zeros
for each $c' \in C$ **do**
  if $c' \notin sameValConds$ **then**
    continue
  end if
  for each $m' \in \mathcal{M}$ **do**
    $w \leftarrow 1.01 - distance(m_p, m')$ ▷ $1.01\underline{}$ for smoothing
    if $c = c'$ **then**
      $w \leftarrow 0.8 \cdot w$
    else ▷ factor in condition attribute
      $w \leftarrow \frac{0.2 \cdot w}{|sameValConds|}$
    end if
    for each $t \in \mathcal{T}$ **do**
      $predU_t \leftarrow predU_t + w \cdot CUT(\mathcal{D})_{m',c,t}$
    end for
  end for
end for
$catgAttributes \leftarrow$ GetRelevantCategoricalAttributes($\mathcal{T}$)
$biCo \leftarrow max(\{\binom{|C|}{|\text{SetOfValues}(attr)|} \mid attr \in catgAttributes\})$
select $mostInfoAttr$ **with** $biCo = \binom{|C|}{|\text{SetOfValues}(mostInfoAttr)|}$
$valsToAvgU \leftarrow$ maps from values $\in$ SetOfValues($maxInfoAttr$) to zeros
for each $t \in \mathcal{T}$ **do**
  select $value$ **with** $(mostInfoAttr, value) \in t$
  $valCount \leftarrow$ NumOccurrences($\langle mostInfoAttr, value\rangle, \mathcal{T}$)
  $valsToAvgU \leftarrow valsToAvgU + \frac{predU_t}{valCount}$
end for
for each $t \in \mathcal{T}$ **do** ▷ factor in therapy attribute
  select $value$ **with** $(mostInfoAttr, value) \in t$
  $valCount \leftarrow$ NumOccurrences($\langle mostInfoAttr, value\rangle, \mathcal{T}$)
  if $valCount \geq 2$ **then**
    $w \leftarrow \frac{0.2 \cdot valCount}{valCount - 1}$
    $predU_t \leftarrow (1 - w)\, predU_t + w \cdot valsToAvgU_{value}$
  end if
end for
select $trials$ **with** ("trials", $trials$) $\in p$
for $j \leftarrow 1$ to $|trials|$ **do** ▷ disprefer therapies already tried
  select κ **with** ("condition", κ) $\in tr_{p,j}$
  if $id(pc_{p,input}) = \kappa$ **then**
    select θ **with** ("therapy", θ) $\in tr_{p,j}$
    $predU_{id^{-1}(\theta)} \leftarrow predU_{id^{-1}(\theta)} - 1.01 \cdot |\mathcal{M}|$
  end if
end for
return $predU$

across clusters, with the highest weight (1.01) being associated to the cluster whose member the patient of interest is and the lowest possible weight (0.01) being associated with any cluster maximally distant from the patient's cluster. When predicting utilities under the hybrid approach, we will also factor in the **most informative categorical attribute** of condition and therapy entities respectively. This is done by choosing the categorical attribute the size of the set of values of which has the largest binomial coefficient with respect to the overall size of the entity set (C or $\mathcal{T}$). Then, the average predicted utility among other entities with the same value for that attribute will account for 20% of the predicted utility value for a particular entity; the other 80% will be computed solely on the basis of that particular entity itself. By contrast, under the non-hybrid, 'simple' collaborative-filtering approach, those attribute values will be ignored and thus not factored in in any way.

In case of extremely sparse data which would result in there being fewer than five therapies with non-trivial predicted utility values, a random sample the size of the missing number of such therapies will be drawn and appended to the recommendations in order to always consistently report a list of therapies of length 5.

5 IMPLEMENTATION

The algorithmic solution presented in Section 3 was implemented using Cython. Numerical values of the kind referred to as $\in \mathbb{R}$ in the previous sections were stored as 32-bit floating point numbers. To ensure memory efficiency, large and sparse vectors, matrices and tensors were stored in form of dictionaries, with any missing dictionary key being assumed to represent a dimension/cell where the corresponding entry is zero, or some other default value. For clustering, an out-of-the-box implementation of the FasterPAM algorithm, provided by the authors of [5] themselves in form of their PyPI package `kmedoids`, was used.

In order not to have to minimise run times for repeated patient-condition queries, intermediate results that are useful across queries were stored as files on the hard drive. These include,

- the raw utility tensor,
- the selected relevant attributes for patients, conditions, and therapies, alongside their corresponding values,
- the *hybridVec* representations of patients,
- the mapping from patients to cluster medoids,
- the condensed utility tensor.

An evaluation mode, allowing for automated testing of a large number of queries on modified versions of the input dataset from which some successful cases have been removed, was implemented as well. More on that is going to be laid out below in Section 7.

6 DATASET

Two different artificial datasets, referable to as **A** and **B** respectively, were examined. While both datasets are valid datasets $\mathcal{D}$ in line with the definitions presented in Section 3, they differ from each other in various ways: One apparent way is the number of entities for the three major categories. In dataset **A**, we have $\langle |C|, |\mathcal{T}|, |\mathcal{P}|\rangle = \langle 292, 170, 60\,000\rangle$, compared to dataset **B**'s $\langle |C|, |\mathcal{T}|, |\mathcal{P}|\rangle = \langle 322, 51, 100\,000\rangle$. But also the sets of attributes of patient entities are quite different: Whereas, in dataset **A**, we

find 2 categorical attributes, i.e. `"name"`, `"sex"`, and 1 numerical attribute, `"age"`, it is a whole 7 categorical attributes, namely `"blood_group"`, `"country_of_residence"`, `"email"`, `"gender"`, `"name"`, `"occupation"`, `"type"`, and 2 numerical attributes, `"age"`, `"birthdate"`, in dataset **B**. Then again, the attribute schemata for therapy and condition entities are identical among the two datasets; for any condition or therapy, there are always the categorical attributes `"name"`, `"type"`, and no other.

One implementation-related detail worth mentioning here, is that in the raw datasets, values for some trial attributes (`"start"`, `"end"`), some patient-condition attributes (`"diagnosed"`, `"cured"`), and patient attributes (`"birthdate"`), were stored as `YYYYMMDD`-formatted date strings. As we wanted to treat them like any other numerical value, we detected such date representations by heuristic criteria like the length of the string being 8, the first two characters being either `19` or `20`, and so on. Then, the detected date strings were converted to numerical values representing the number of days having passed since 1 January 1900. Also notably, with respect to Algorithm 1 for finding further relevant attributes, the case of dataset **B**'s numerical patient attributes `"age"` and `"birthdate"` is the only one, here, where a numerical attribute ever gets excluded due to being highly correlated to another present one.

As already mentioned, both datasets do not contain real data, but are the result of randomised simulations. Dataset **A** was constructed by myself, whereas dataset **B** was provided by the course instructor. Thus, I am not in the position to report on the details of how dataset **B** was constructed, but I can shed some light on how this was done for dataset **A**: After crawling and selecting 'realistic-sounding' names and types for conditions and therapies, a mapping from conditions to sets of therapies was created, indicating which kinds of conditions can ever be treated with trials of which kinds of therapies. Crucially, these sets were designed to differ in size across conditions, i.e. some conditions can be cured only by a very small number of therapies, whereas for other conditions, almost any therapy can be tried. Every condition was also assigned **curability weight**, i.e. a value ranging between 0.1 and 1, which during randomisation was weighted in such a way that trials of conditions with lower curability weights were less likely to terminate successfully. In addition, every condition was assigned a **frequency weight**, designed to be moderately correlated with the corresponding curability weight, and weighted in during the creation of patient-condition entities in a way that would ensure that conditions with a smaller frequency weight would also appear less often in the data. Another included artifical exploitable factor are **sex-specificness** values, again mapped to any condition, indicating whether a condition can occur to both sexes, or either to males or to females only. It was then made sure these values would be restricting randomisation in the expectable way, e.g. that patient entities with ⟨`"sex"`, `"female"`⟩ would never display any conditions marked as `"male"`-specific. Similarly, for each therapy, a general **efficacy weight** was constructed. Then, for any combination of a condition and a therapy supported by the aforementioned condition-to-therapies mapping, a condition-specific efficacy weight for the particular therapy was generated, displaying a value slightly deviating from the general one in a controlled randomised way. Another kind of weight used during dataset generation was a **duration weight** assigned to any therapy and designed to moderately correlate with its respective efficacy weight. As the name suggests, duration weights were used to increase or decrease the average duration trials of any particular therapy would last. Finally, the **age values** of patient entities were used to restrict the average number of conditions an age group would show, motivated by the fact that in the real world we would expect people who have already lived longer to have been sick more often than newborns in their overall lifetime.

Lastly, dataset **A** was provided with three test cases, i.e. an indication of three specific patient entities alongside three respective yet-uncured conditions of theirs, for which the proposed algorithm should thus be able to generate therapy recommendations. Ten such test cases were provided together with dataset **B**.

Table 1: Hard and soft accuracy on auto-selected test cases

Dataset	# Test cases	Method	Hard acc.	Soft acc.
A	49,648	Baseline	.068	.149
		Simple	.414	.755
		Hybrid	.455	.779
B	78,762	Baseline	.019	.059
		Simple	.019	.058
		Hybrid	.020	.059

7 EXPERIMENTAL EVALUATION

In order to evaluate the performance of the proposed recommendation algorithm, both in its full, hybrid as well as in its simple, non-hybrid version, two kinds of evaluations were conducted.

For the first kind of evaluation, we wanted to inspect the **quality** of therapy predictions on a large number of examples. To do that, we randomly split up the dataset at hand (either **A** or **B**) into one portion containing 80% of patient entities, and another portion containing the 20% of patient entities. Then, from the latter, smaller portion of data, those patients were selected who had at least one trial and whose most recent trial had ended with a higher utility, as defined in (18), than 75% of all trials in the overall dataset. These selected, more-utile-than-average trials were subsequently removed from the dataset and used as queries for recommendation. This process was repeated 10 times for each dataset, i.e. amounting to 10 different random splits of data, resulting in 49,648 such auto-selected test cases for dataset **A** and 78,762 ones for **B**. To quantify performance, two accuracy metrics were employed: **Hard accuracy** is the ratio of cases where the first most recommended therapy was actually identical to the one truely administered in the hidden test-case trial. **Soft accuracy** is a more leniant criterion that takes into consideration not only the first most recommended therapy, but the whole top five recommendations, but penalises any case of a true therapy only predicted as 2nd most recommended by scaling their contribution to overall accuracy by 0.8, those predicted only as 3rd by 0.6, as 4th by 0.4, and as 5th by 0.2.

In addition to the two discussed algorithm versions, i.e. **simple** vs. **hybrid** collaborative filtering, a further, primitive **baseline** was also included for evaluation. This baseline method would simply always recommend the five overall most frequent therapies from the dataset, regardless of the current query, allowing for a rough

Table 2: Avg. run time and avg. full execution time

Dataset	# Test cases	Method	Run time	Full time
A	3	Simple	0.46 s	100.69 s
		Hybrid	2.01 s	306.89 s
B	10	Simple	0.74 s	201.96 s
		Hybrid	2.62 s	941.77 s

estimate of what accuracy values should actually be considered 'good enough' in that they reflect a performance above chance.

As the results summarised in Table 1 indicate, for dataset **A**, performances of both the simple and the hybrid approach were quite decent, with accuracies at .414 (hard) / .755 (soft) and .455 (hard) / .779 (soft), respectively. The baseline frequency heuristic only achieves .068 (hard) / .149 (soft) here. But, somewhat remarkably, the picture is a completely different one in case of dataset **B**, where both the simple and the hybrid approach both perform almost exactly at chance, i.e. at around 0.02 (hard) / 0.06 (soft), as does the baseline heuristic. The reasons for this huge performance difference between datasets, in an evaluation design like the present one, should be further investigated. One might suspect a major cause to be found in the differing underlying generation processes behind the two artificial datasets. If, for example, it was the case that dataset **B** had been generated without any meaningful restrictions on the randomisation process, then one could rightfully expect the performance of any algorithm run on it to lie exactly at chance. To put it differently, no algorithm can predict next week's lottery numbers based on the ones from the whole past year.

The second kind of evaluation concerned the **scalability** of the proposed algorithm, indicated by differences in average execution time on the pre-selected test cases provided with datasets **A** and **B**. Here, we distinguish between **run time**, i.e. the time that a query needs to be processed before returning therapy recommendations if all of the required dataset-specific, but query-agnostic pre-computed intermediate results (see Section 5) have already been created and stored as files that just have to be read in, and **full time**, i.e. the time needed for processing if no prior processed data is available and thus all computation steps have to be carried out from scratch.

Knowing that dataset **B** contains roughly 1.7 as many patient entities and about twice as many patient attributes as dataset **A**, it is interesting to see how such differences affect exeuction time. Table 2 summarises the relevant results. First, we can observe that the hybrid method was consistently associated with a higher time cost compared to the simple one, which is not too surprising given that it has to process all entity attributes instead of focusing exclusively on the raw utility values. Interestingly, though, the factor of time cost introduced by the hybrid algorithm compared to the simpe one was even higher in case of dataset **B**, with 941.77 seconds (hybrid) / 201.96 seconds (simple), i.e. a factor of roughly 4.7, as opposed to dataset **A** with 306.89 seconds (hybrid) / 100.69 seconds (simple), i.e. a factor of roughly 3. This can probably be explained by the fact that the multitude of patient entity attributes present in dataset **B** leads to a much more time-consuming computation of cosine distances, as the underlying vector representation are now scaled up in dimensions much more drastically than it is the case for dataset **A** with its relatively few patient entity attributes.

Indeed, the major bottleneck of the proposed hybrid recommendation algorithm seems to be the phase when patient vectors are assigned to their closest respective cluster medoids (gotten from FasterCLARA), precisely due to the increased computation effort that calculating of $|\mathcal{P}| \cdot |\mathcal{M}|$ cosine-distance values among heavily multidimensional vectors entails. As a future direction, it might be interesting to try an alternative and simpler distance metric, like Jaccard distance, which would also allow for computing similarity-preserving hashed representations of vectors, thus potentially making distance calculation more efficient even with large numbers of included entity attributes, although with some trade-off in quality.

REFERENCES

[1] Gediminas Adomavicius and Alexander Tuzhilin. 2005. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering* 17, 6 (2005), 734–749.

[2] Levent Ertöz, Michael Steinbach, and Vipin Kumar. 2003. Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data. In *Proceedings of the 2003 SIAM international conference on data mining*. SIAM, 47–58.

[3] Leonard Kaufman and Peter J Rousseeuw. 1990. Clustering large applications (program CLARA). *Finding groups in data: an introduction to cluster analysis* 344 (1990), 126–163.

[4] Leonard Kaufman and Peter J Rousseeuw. 1990. Partitioning around medoids (program PAM). *Finding groups in data: an introduction to cluster analysis* 344 (1990), 68–125.

[5] Erich Schubert and Peter J Rousseeuw. 2021. Fast and eager k-medoids clustering: O (k) runtime improvement of the PAM, CLARA, and CLARANS algorithms. *Information Systems* 101 (2021), 101804.

[6] Xiaoyuan Su and Taghi M Khoshgoftaar. 2009. A survey of collaborative filtering techniques. *Advances in artificial intelligence* 2009 (2009).