ULB UNIVERSITÉ LIBRE DE BRUXELLES

INFO-H515 - Big Data : Distributed Data Management and Scalable Analytics - 202122

RecSys 2022 Challenge - Fashion recommendations

David Perez Edwige Loems

Sid Ahmed Bouzouidja

Group 2

Course teachers : Gianluca Bontempi

Dimitris Sacharidis

Assistants : Théo Verhelst

Daniele Lunghi

Antonios Kontaxakis

June 2022

Contents

1 Introduction and overview of the data **1**

1.1 Objective of the project 1

1.2 Overview of the data 1

2 Pipeline for data preprocessing - Task 1 **3**

3 Scalable features selection algorithm - Task 2 **9**

3.1 Creation of the input matrix X and output vector Y 10

3.2 Ranking algorithm 11

3.3 Forward features selection 11

3.4 Implementation of the features selection methods and features selected 12

4 Predictions - Task 3 **14**

4.1 Explanation of the approach 14

4.2 Description of the algorithm 15

4.2.1 Computation of the pair RDD with the id of each session each item appears in 15

4.2.2 KNN algorithm 16

4.3 Evaluation of a prediction 17

4.4 Optimisation of the KNN search's parameters 17

4.5 Assessment of feature selection 17

5 Conclusion **19**

Bibliography 20

1 Introduction and overview of the data

The online video of our presentation can be found in the following onedrive folder :
https://universitelibrebruxelles-my.sharepoint.com/:f:/g/personal/edwige_loems_ulb_be/EoaNF3rb4H9Ml7gI5UOS6RsBrNHR8QSvvifzSnGTYGRfeA?e=a4aBbC.

1.1 Objective of the project

The project revolves around participation in the RecSys 2022 challenge. This year's RecSys challenge focuses on fashion recommendations for online shopping. The challenge's objective is to predict which item will be bought at the end of a session, given user sessions, purchase data, and content data about items.

1.2 Overview of the data

According to the RecSys challenge 2022 website [**1**], the dataset contains 1.1 million online retail sessions in the fashion domain, sampled from 18 months. All the sessions are purchasing sessions that resulted in at least one item purchased. The dataset is split by date into a training set and a test set, as visualized in figure 1.1. The training set is composed of the first 17 months period, and the test set is the last month's sessions. The test set is split randomly into a leaderboard test set and a final test set containing 50 thousand sessions each. The training set contains 1 million sessions.

Our objective is to provide a top 100 ranking of the items likely to be bought for each query session. During an online retail session, the recommendations are shown to the user at various points in their sessions. For that reason, the length of each query session is diminished by 0 to 50%.

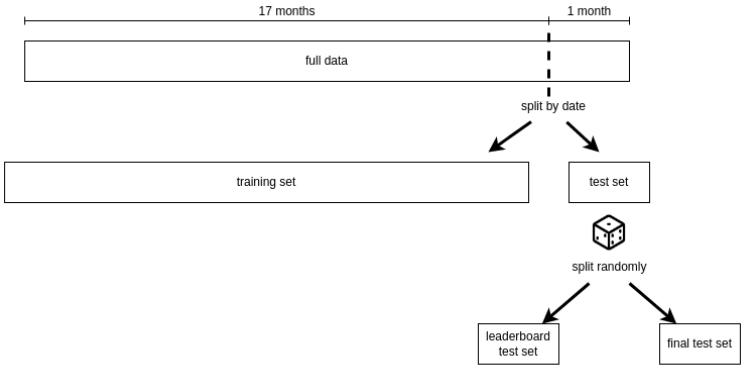

Figure 1.1: Splitting the data into a training set and a test set [**1**].

The dataset is composed of 6 CSV files described below. The RecSys Challenge website also mentions information about the sessions, the features category, and their values. The sessions are anonymous. If a session resulted in multiple purchases, only one will be selected randomly (the RecSys challenge website mention that the size of the dataset should be sufficiently large to compensate for it). Many sessions are short. Some items may not share many features category id if they are different types of items, and items will have a different number of assignments depending on how complicated they are. An item might also have multiple values for the same category (ex: second color).

1. The ***train purchases*** contain the purchases that happened at the end of the session.
2. The ***train session*** lists the items viewed in a session, up and not including the item bought on that session.
3. The ***Item features*** contain the feature category and the corresponding value for each item.
4. ***Candidate Items:*** the set of items purchased in the test month.
5. ***Test leaderboard sessions:*** input session for prediction for the leaderboard.
6. ***Test final sessions:*** input session for the for the challenge ranking.

The initial ***features_category_id*** will be referred as the features $\boldsymbol{f}$ corresponding to the features describing the items. Later on, additional features describing the sessions will be engineered; the session feature deriving from the item feature f_i will be written F_i.

2

Pipeline for data preprocessing - Task 1

This section explains the series of transformations applied to our pair RDDs and the feature engineering. The architecture of task 1 is illustrated in figure 2.1. Our objective is to produce a pair RDD with the ***session_id*** as key and a list containing the value of the engineered features as value. This pair RDD will be used for the feature selection.

To optimize our work, the choice was made to sample our training set and first work on a reduced dataset to minimize the running time of our notebook. The dataset reduces by undersampling corresponds to the first two out of the 17 months. Unfortunately, due to a lack of resources, the dataset used for the feature selection and model prediction was reduce as well. The selected features and obtained predictions might have been more accurate using the whole dataset. The dataset used for the feature selection methods and the training of the predictive model contains a random sample of 92044 sessions. The dataset used for the evaluation of the predictive model contains 1070 sessions sampled from the most recent sessions.

After exploring the information gathered from the Recsys website [**1**], we generated the three initial RDDs from the training set. The project was done in a Spark session inside the cluster. Spark is a fast, in-memory, distributed data processing engine. An RDD (Resilient Distributed Dataset) is Spark's abstraction for representing a very large dataset. The significant advantage of using Spark is that it automatically distributes the data contained in RDDs across the cluster and parallelizes the operations performed on them. Four major transformations and set operations were done on the RDDs. The transformations consist of creating new RDDs from existing ones, and the set operations are performed on two RDDs and produce one resulting RDD.

- The ***filter transformation:*** Takes in a function and returns an RDD formed by selecting those elements which pair the filter function.
- The ***map transformation:*** Takes in a function and passes each element in the input RDD through the function, resulting in the function being the new value of each element in the resulting RDD.
- The ***flatMap transformation:*** Takes each element from an existing RDD and can produce 0, 1, or many outputs for each element.

- ***Union (set operation) :*** returns data from both input RDDs.

The initials RDDs generated are pair RDDs. The pair RDDs are a particular type of RDD that can store key-value pairs. Most of the functions of the RDDs are appliable to the pair RDDs. Some functions are only applicable to pair RDDs. The main functions used are listed below.

- The ***mapValues function*** is applied to each key-value pair and converts the values based on the mapValues function without changing the keys.
- The filter transformation: Takes in a function and returns a Pair RDD formed by selecting those elements which pass the filter function.
- The ***reduceByKey transformation*** allows us to aggregate statistics across all elements with the same key. The reduceByKey function runs several parallels reduce operations, one for each key in the dataset, where each operation combines values with the same key. It returns a new RDD consisting of each key and the reduced value for that key.
- The ***groupByKey transformation*** group the values for each key in the RDD into a single sequence.
- The ***sortByKey transformation*** sort a pair RDD by the ordering defined on the key, and the sortBy function sort the pair RDD by the value.

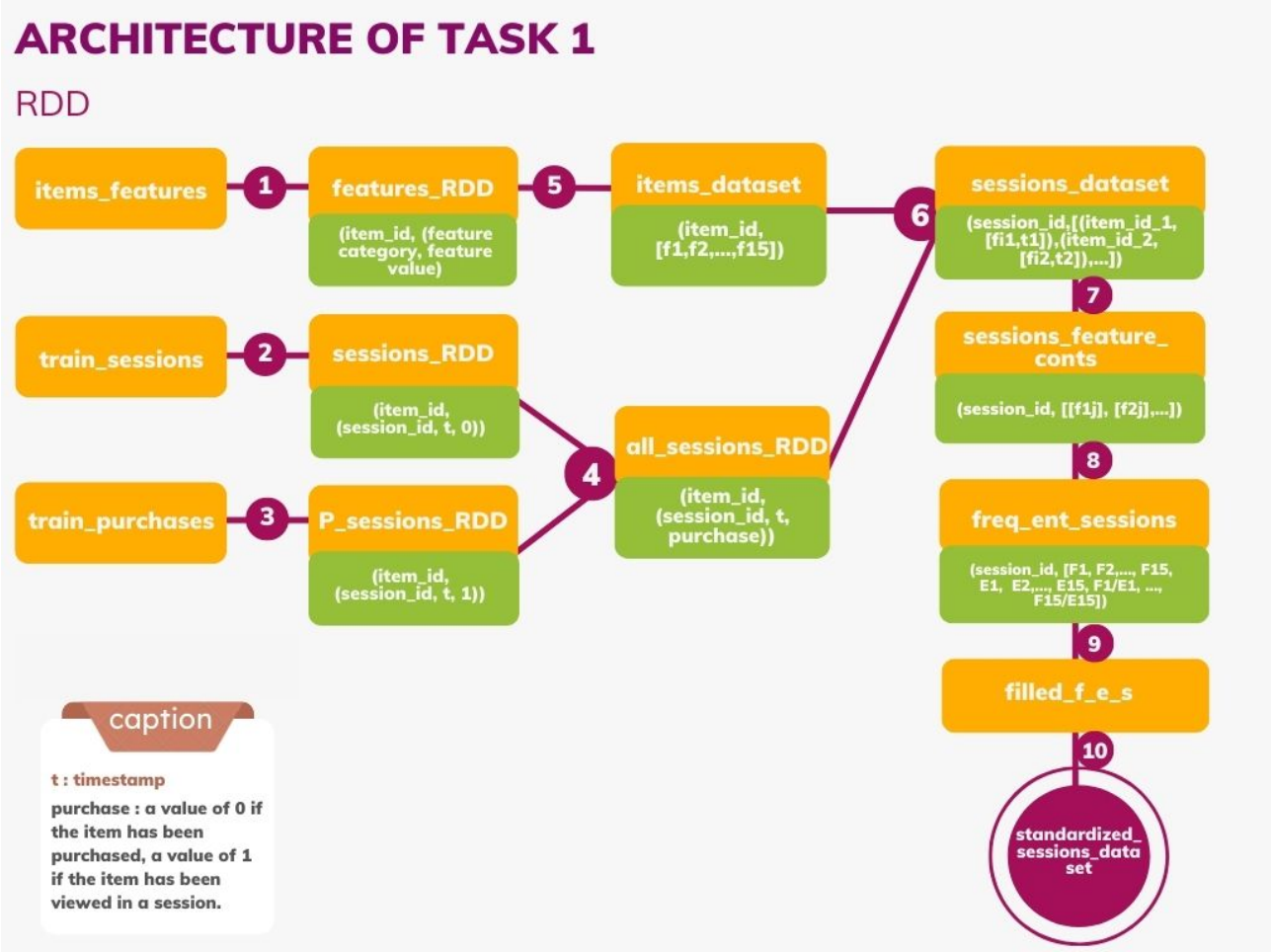

Figure 2.1: Architecture of task 1

The three first pair RDDs are represented in figure 2.1 after the numbers 1, 2, and 3. The ***features_RDD*** was generated from ***items_features***. The RDD contains the item id as key and tuple (***feature_category_id***, ***feature_value_value_id***) as value. The ***sessions_RDD*** was generated from the dataset ***train_sessions***. It contains the ***item_id*** as key and the tuple ***(session_id, timestamp, 0)*** as value. A value of 0 was added for each item present in the file ***train_session*** to indicate that the item viewed has not been purchased. The ***P_sessions_RDD*** was generated from the dataset ***train_purchases***. It contains the ***item_id as key*** and the tuple ***(session_id, timestamp, 1)*** as value. A value of 1 was added for each item present in the file ***train_purchases*** to indicate that the item was purchased.

The timestamps initially take the following format: ***yyyy/MM/dd HH:mm:ss***. Each timestamp has been converted to a numeric value to facilitate our further work. The first and the last timestamps were printed to verify that the period between the first session and the last was 17 months.

We used the groupByKey function on the ***features_RDD*** and ***sessions_RDD datasets*** to further explore the data. This allowed us to obtain the count of unique items present in each RDD. To retrieve that information, we used an action that transforms an RDD into a typical programming language value. The number of unique items contained on the ***sessions_RDD*** is lower than the number of unique items contained on the

features_RDD. This is because the ***items_features*** dataset contains a description of all the items and the ***train_sessions*** dataset only contains the items view in each session.

Comparing the number of rows in the ***train_sessions*** dataset and the number of unique occurrences of ***item_id*** in the ***sessions_RDD***, it is observed that some items appear consulted more than once in the same session. Some items can have multiple values for the same feature, as explained before. In that case, the items appear to be viewed multiple times in the same session, with a different value for the corresponding feature. To solve that issue, the key of the ***session_RDD*** dataset was modified to contain both the session and the item id. The function reduceByKey was used to group together the multiple occurrences of an item in a session. Only the first occurrence is kept because it contains the primary features of the item.

Transformation 4 corresponds to the union of the ***session_RDD*** and the ***P_sessions_RDD***. This transformation allowed us to treat together with the information about the viewed and purchased items in a session.

The ***feature_RDD*** was used to count the number of unique feature categories. In figure 2.2, It is observed that there is 73 unique feature category. As described on the RecSys website, many features only describe part of our items. With the help of figure 2.2, we have set the minimum threshold of items that a feature must describe at a value corresponding to half of our items.

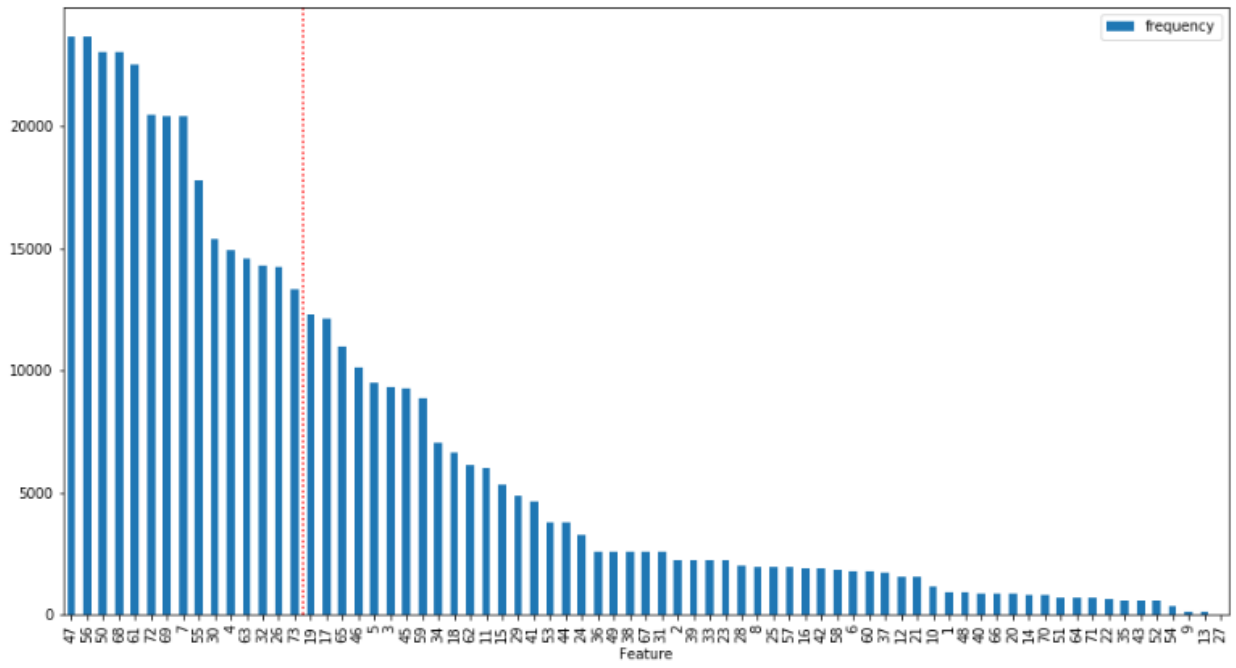

Figure 2.2: Plot of the features sorted by the number of objects for which the features have a defined value

In transformation 5, a map, and a reduce transformation was used to reshape ***features_RDD***. The previously set threshold was also used to decrease the number of feature categories to 15 (feature category id : 4, 7, 26, 30, 32, 47, 50, 55, 56, 61, 63, 68, 69, 72, 73). It creates a new RDD ***items_dataset*** that has the structure

(item_id, [f1,...f15]). The number of keys corresponds to the number of unique item_id. The values correspond to a tuple with the value of each of the 15 remaining feature categories. If a feature does not have a value for a feature category, the value is set to 0. It has been previously verified that 0 is not used as a value in any feature category.

The ***items_dataset*** and ***all_sessions_RDD*** are joined in transformation 6 to create the RDD ***sessions_dataset***. Each key is a ***session_id***, and the value is a list of lists, each containing the ***item_id*** of the viewed or purchased item and a list with the value of the features and timestamp.

Transformation 7 creates the ***RDD sessions_features_counts***. For each session, all the values for each feature of the viewed purchased item are set in an array. The timestamp was removed at this step of the data preprocessing as it is not used either as a feature or for feature engineering.

With transformation 8, we aim to produce a pair RDD with the ***session_id*** as key and a tuple containing the value of the features as value ***(freq_ent_sessions)***. Each session is mapped to an array of the session's features.

A list of engineered variables now defines the value of our RDD. Our dataset is described by 45 features that can be divided into three category : the session features (Fi), Shannon entropy (Ei) of the distribution of fi and, the Session features divided by Shannon entropy + constant ($\frac{Fi}{Ei+c}$).

- ***F1,...F15 (Session features:)*** For each feature category, the value of the most represented feature among the items consulted in the session (most frequent value for each item feature) is assigned. If the most represented feature is null, the value of the following most represented feature among the items consulted in the session is assigned. If all the features ***f*** inside a session are null, F takes as value the general average of the corresponding feature. If there is an equality in the most represented feature, the value selected is chosen randomly between the most represented features. It could be considered to select the most represented feature viewed last.

- ***E1,...E15 (Shannon entropy):*** The entropy enables to measure the uniformity (or the disparity) of the different values that a feature takes over the different items in a session. Entropy is chosen as a feature because it seems more appropriate for categorical variables, for which the values of the features have no quantitative significance With continuous variables, it would have been preferred to use the variance.

  The entropy E_i of a session is defined by $H = -\sum_{i=1}^{M} P_i \, log_2 P_i$. The entropy E_i is the Shannon entropy of the distribution of f_i. The entropy will have a value of 0 if all the items in a session have the same value for the feature. The entropy will be maximal if all the session items take the values of the feature represented uniformly. $\boldsymbol{P_i}$ represents the relative frequency of occurrence of the value i for the

corresponding feature, and M is the total number of values that the feature can take in the session. Table 2.1 represents a fictional dataset of 1 session, three items, and one feature category. In this case, the most represented feature is two, and the feature session F1 equals 2 (F1=2). With the relative frequency of occurrence of the three values for the feature, the entropy E1 of the distribution of f1 can be computed. In this case, E1 will be equal to $H(X) = -[1/3 * log_2(1/3) + 2/3 * log_2(2/3)]$

List of all the items in the session 1	Feature category1, value for each item	Relative frequency
item 1	4	$\frac{1}{3}$
item 2	2	$\frac{1}{3}$
item 3	2	$\frac{1}{3}$
	$F_1 = 2$	$E_1 = 0.91$

Table 2.1: Example of the data present in a fictitious session 1 containing 3 items

- ***F1/(E1+1),...F15/(E15+1) (Session features divided by Shannon entropy + constant) :*** 15 new features are, generated by dividing each session feature (F_i) by the entropy of the distribution of its corresponding feature (fi). The motivation for this feature is that it might contribute some relevent information to combine these features in this non linear way, where the feature F_i is given more importance in a session if most of the items in the session have the same value for that feature, and inversely, features that take a lot of values in the session will have a higher entropy and thus be given a smaller ponderation.

Transformation 9 replaces, for each session, the features that are null with the means of the corresponding feature. Finally, transformation 10 corresponds to the standardization of the features (Z-Score normalization). Each feature (x) is transformed by subtracting the mean(μ) and dividing by the standard deviation (σ). The standardized feature take the value $x' = (x - \mu)/\sigma$

The Spark RDD containing the dataset for further analysis is obtained. Our variables are resized and are comparable on a common scale.

3

Scalable features selection algorithm - Task 2

This part will discuss how we select a relevant set of features from the feature engineering step using the MapReduce technique.

An issue that arose when discussing the way to compute the feature selection was related to the dimensionality of our data. It seemed at first that the input matrix X should represent the features describing each session : each line corresponds to a session and each column one of the 45 features to evaluate. However, it is not as clear what the output should be. Logically the output should represent the item purchased, as it is what is supposed to be predicted, but each item is represented by a vector of dimention n, n being the number of features describing a item. In that case, the output vector y is an array of dimension #sessions×#item_features.

However that dimensionality does not work for the feature selection algorithms that we intended to use : an output vector of dimension #session is required. A solution that was considered for the problem was to use an input matrix X where the number of line is no longer the number of sessions but instead the number of events (for each session there is a line for each item either consulted or purchased), and the output would be an array with 0s for the items consulted and 1s for the items purchased. However in this case all the features would have had to be modified and the result unsure.

Another approach was found more convincing, and was hence selected. The motivation for this approach is knowing that the method chosen in task 3 (the implementation of a prediction model) is a method of collaborative filtering, where the assumption made is that the item purchased is determined based on the purchase history of similar sessions. Thus, the way a session is positionned in a features space is not as crucial to evaluate as the way the distance between two sessions in the feature space is computed. For that reason, it was chosen to randomly match pairs of sessions together, and compute the (absolute value of the) difference between each of their features. In that case, the X matrix becomes a matrix where each line is a pair of session, and each column the difference between their respective features. As the output, the euclidian distance of the purchased items of each session in the item feature space was chosen. Item feature space in this case means the space of the 15 items features selected during task 1.

When using the output and input variables of both methods, we observed that for the mRMR implementation, the correlation score between the features x and the output y was significantly lower when we used the input and output variables of the first method. For that matter, we chose our first approach to feature selection.

As described in the guidelines, we implemented a ranking algorithm and a forward selection in a distributed way.

3.1 Creation of the input matrix X and output vector Y

In the first method, we create pairs of sessions. The objective is to know the difference between the session's features and the distances between the session's purchased items. We will first create a pair ***RDD X_AND_Y*** that contains all the required data to create the input matrix X and the output vector Y.

We will first modify the RDD ***items_dataset***. The key will remain the same (item id of the items viewed and purchased in every session). The value of the RDD is composed of the fifteen item features (f) of the corresponding item. A value of 0 was previously attributed to the features that did not have a value for the item. In this step, the null values have been replaced with the mean of the corresponding feature for every session. In the same way as for the session features, we also realized a standardization of the item features (Z-Score normalization). Each feature (x) is transformed by subtracting the mean(μ) and dividing by the standard deviation (σ). The standardized feature take the value $x' = (x - \mu)/\sigma$
We joined this RDD and the ***P_sessions_RDD*** to select the purchased items for each session.

This RDD was finally joined to the ***standardardized_sessions_dataset RDD***. We obtain a pair RDD ***sessions_features_with_purchased_items*** where the key is the session id and the value contains a list of the 15 features (f) and the 15 features purchased item of the session. To create the list of pairs of sessions, we added an index to the RDD where there are two sessions per index value.

The indexed RDD has been used to create two RDD with a tuple containing the session id of the pair of sessions that share the same index as key. The first RDD is the ***sessions_pairs_difference***. The value is the difference between the pair's first session and second session features. The second RDD is called ***purchased_items_distances***. The value is the euclidean distance between the features of the object purchased in the pair's first and second session. The first RDD corresponds to our input matrix X and the second RDD corresponds to our output vector Y. To ensure that the order of the rows in X and Y is equivalent, we joined the RDDs and used the map function to create the X and Y arrays.

To implement the two feature selection methods, we used broadcast variables to keep a read-only variable cached on each machine rather than shipping a copy of it with tasks. All broadcast variables will be kept at

all the worker nodes for use in one or more Spark operations. We broadcasted the target variable Y, and we saved the values of the predictive variable X in a broadcast variable ***broadcast_selected_features_values***.

3.2 Ranking algorithm

The ranking algorithm chosen in this project is the mRmR (minimum Redundancy Maximal Relevance) method. It is a forward selection method that selects at each step the variable with the highest degree of relevance and the lowest redundancy with the already selected features [**2**]. The pseudocode of the algorithm is illustrated in figure 3.1.

Algorithm 1 mRMR: original algorithm

```
INPUT: candidates, numFeaturesWanted
// candidates is the set of initial features
// numFeaturesWanted is the number of selected features
OUTPUT: selectedFeatures // The set of selected features.
for feature fi in candidates do
   relevance = mutualInfo(fi, class);
   redundancy = 0;
   for feature fj in candidates do
      redundancy += mutualInfo(fi, fj);
   end for
   mrmrValues[fi] = relevance - redundancy;
end for
selectedFeatures = sort(mrmrValues).take(numFeaturesWanted);
```

Figure 3.1: mRMR pseudocode [**3**].

3.3 Forward features selection

The forward feature selection is a wrapper method (assess subsets of variables according to their usefulness to a given predictor). In the forward feature selection method, the procedure starts with no variables and progressively incorporates features. The first input selected is the one that allows the lowest generalization error. The second input selected is the one that, together with the first, has the lowest error, and so on, until no further improvement is made or the required number of features is attained [**2**]. The pseudocode corresponding to the general algorithm for feature selection is illustrated in figure 3.2.

3. GENERAL ALGORITHM FOR FEATURE SELECTION

The basic feature selection algorithm is shown in the following.

Input:
S - data sample with features X,|X| = n
J - evaluation measure to be maximized
GS – successor generation operator

Output:
Solution – (weighted) feature subset
L := Start_Point(X);
Solution := { best of L according to J };

repeat
L := Search_Strategy (L,GS(J),X);
X' := {best of L according to J };
if J(X')≥J(Solution) or (J(X')=J(Solution) and |X'| < |Solution|) then Solution :=X';

until Stop(J,L).

Figure 3.2: Forward feature selection pseudocode [**5**].

3.4 Implementation of the features selection methods and features selected

Both those algorithms are iterative, in which a feature is added to the list of selected features at each iteration. A feature is added during an iteration by computing a score for each feature and selecting the feature with the highest score.

In the case of the mRMR, the score is computed with the function ***get_mrmr_score_spark*** by computing the difference between a relevancy score and a redundancy score. In our case, the relevancy was the mean of the scores of a 3 fold crossvalidation of a linear regression model trained on a candidate feature. The redundancy was the average correlation between the candidate and selected features. The function ***scipy.stats.pearsonr*** was used to compute the Pearson correlation coefficient [**4**]. The selected features are ordered (ranked) according to their score.

In the case of the forward selection, the score is the crossvalidation score of a linear regression model trained on the candidate feature combined with the set of features already selected.

The part of this algorithm that can be parallelized is the computation of the score of each feature. A boolean variable ***choose_mrmr*** determines whether the mRMR or forward features selection is computed. The subset of selected feature values is retrieved and cast as an array for both methods. At each iteration, the feature with the highest score is added to the selected features (and removed from the remaining ones). For the mRMR method, the already selected features are priorly removed. Only the remaining feature mapped using the function ***get_mrmr_score_spark***.

The dataset used for the feature selection methods and the training of the predictive model contains a random sample of 92044 sessions. The features selected with the mRMR method are listed in table 3.1.

selected features index	7	30	5	26	5	13	8	1	18	37
selected feature	F8	$\frac{F1}{E1}$	F6	E27	F4	F14	F9	F2	E19	$\frac{F8}{E8}$

Table 3.1: mRMR features selection

The features selected with the forward features selection method are listed in table 3.2.

selected features index	7	3	11	42	32	34	37	44	9	36
selected features	F8	F4	F12	$\frac{F13}{E13}$	$\frac{F3}{E3}$	$\frac{F5}{E5}$	$\frac{F8}{E8}$	$\frac{F15}{E15}$	F10	$\frac{F7}{E7}$.

Table 3.2: Forward features selection

4

Predictions - Task 3

4.1 Explanation of the approach

The general strategy used to produce the predictions asked by the challenge is to use a collaborative filtering model, where we make the hypothesis that similar sessions are likely to end in the purchase of the same item. Hence, when looking at a session and attributing a score to an item to characterise the likelihood that this item will be purchased in that session, the score attributed to the item is going to depend on the amount of sessions similar to the session of interest where this item was consulted.

To define this score, we followed the method of Jannach and Ludewig in their article " When Recurrent Neural Networks meet the Neighborhood for Session-Based Recommendation."[**6**], in the section *2.1.2 Session-based kNN*. In this section, as its name suggests, a k-nearest-neighbors algorithm is used to determinate the sessions in the neighborhood of the session for which the scores are computed. In this algorithm, a measure of similarity is used to quantify the similarity between 2 sessions s_1 and s_2 : $sim(s_1, s_2)$.

Once N_s, the k sessions in the neighborhood of a sessions s are determined, and their similarity with s computed, the score of a recommendable item i is computed based on this expression :

$$score_{KNN}(i,s) = \sum_{n \in N_s} sim(s,n) \times 1_n(i) \tag{4.1}$$

$1_n(i)$ is a coefficient equal to 1 if n contains i and 0 otherwise. In this project, it was chosen to interpret "contains i" as meaning that the item i was either consulted or purchased during this session, as we were worried that it would have been too limiting otherwise and might have resulted in false negatives – with more time, it could have been interesting to try it this way anyway, or replace $1_n(i)$ by a coefficient that would take a different value weather the item was purchased or merely looked at during session n.

The article mentions optimizing their algorithm by creating an in-memory index data structure (cache) on startup. Specifically, for each item an index is created that points to the sessions in which the item appears. This method was used in this project as well.

There is however a way in which the approach taken in this project diverges slightly from [**6**] : there was a concern that in the context of using a very large number (up to 1 million) training sessions, it might be costly to compute an exact kNN, which requires to compute the similarity of s with every other sessions and ranking these similarities to select the k highest. This concern is a common one, which is why alternative methods to exact kNN, like approximate nearest neighbor searches – and among these algorithms, locality sensitive hashing (LSH) – were developped [**7**].

As LSH methods were also a topic discussed in the INFO-H515 class, it was chosen to use such an algorithm to compute the nearest neighbor search in this project. In particular the PySpark.ml package has a *BucketedRandomProjectionLSHModel* class, which is what was used here. The Bucketed Random Projection is an LSH family for Euclidean distance [**8**]. In this algorithm, the hashing function consist in projecting the feature vector on a random vector $\boldsymbol{v}$:

$$h(\boldsymbol{x}) = \left\lfloor \frac{\boldsymbol{x} \cdot \boldsymbol{v}}{r} \right\rfloor \tag{4.2}$$

Here, r is a parameter called the bucket length, and it controls the number – and size – of buckets. A larger r results in more features beeing hashed into the same bucket, which increases the number of true and false positives [**8**].

Computing the nearest neighbor search in this case consist in calling the *approxNearestNeighbors* function on the *BucketedRandomProjectionLSHModel* model – fitted on the training dataset. This function not only returns the approximate k nearest neighbors of the session s, but the euclidian distance of this session with the k neighbors as well. This distance was used in order to compute a similarity function – the one used in equation 4.1. The similarity function chosen is

$$sim(s,n) = \frac{1}{dist(s,n)+1} \tag{4.3}$$

where the +1 in the denominator enables to avoid any division by 0, and to insure a similarity always comprised between 0 and 1.

4.2 Description of the algorithm

4.2.1 Computation of the pair RDD with the id of each session each item appears in

The computation for a given session s of the score of each item is made from a applying a mapping function on the pair RDD ***items_sess_table***, an RDD in which the keys are the item_id of each item and the value is a list of the session_ids of all the sessions the item appears in (either consulted or purchased).

This RDD is computed from ***sessions_list***, an RDD in which the keys are the session_id and the values are a list of the item_id of all the items present in the session. The steps to go from ***sessions_list*** to

items_sess_table are represented in figure 4.1 (the map operations made to change the order or configuations of key-value pairs in the RDDs are not explicitely represented but are implied).

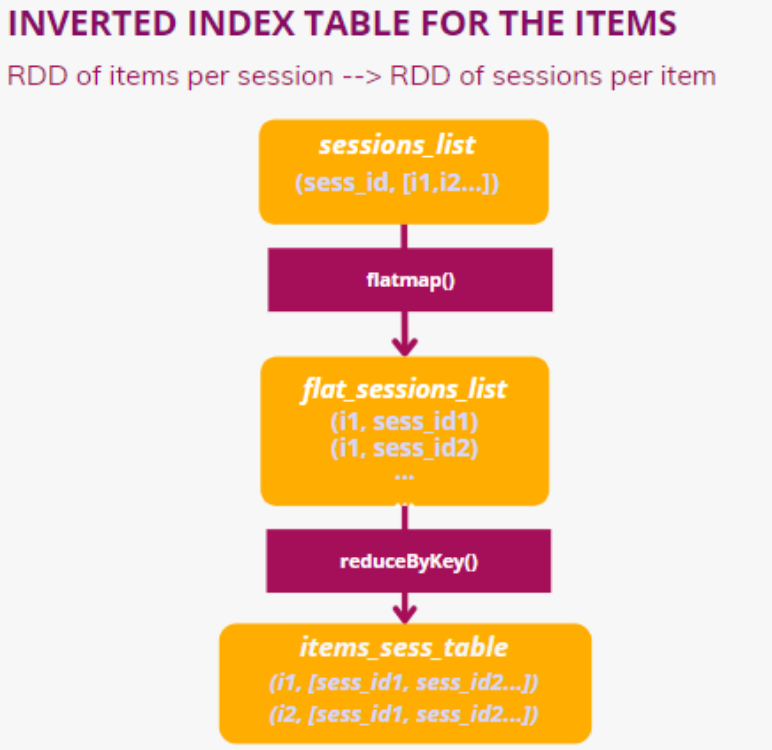

Figure 4.1: Steps to get the table which, for each item, points to the session in which the item appears

4.2.2 KNN algorithm

To compute the top 100 items most likely to be bought following a given session *s*, the following steps are taken :

- The k nearest neighbors are (approximately) computed using the bucketed random projections algorithm
- The similarity between *s* and the neighbors is computed
- The list of the neighbors and their associated similarity score is broadcasted
- The score associated with each possible item is computed. The way this is done is by using the RDD ***items_sess_table*** (of which the computation is described in the previous section) and a map function is applied in order to compute the score of each item (each item coresponding to a line of the RDD). This way, the score of the different items are computed in a distributed way.
- This step results in a new RDD : ***items_scored***, where each key is an item id and the value is the score associated with the item. The last steps are to order this list of item according to their score, and take the first 100 items.

This operation is run in a loop for each session for which we wish to compute the 100 most likely purchased items. The results can be written into a text file.

4.3 Evaluation of a prediction

The results of the predictions of the algorithm for a set of sessions is stored in a new RDD : ***sessions_top100***. To measure the quality of the predictions, it was decided to follow the idea of the evaluation by the RecSys challenge, which is to attribute a score on a prediction depending on the ranking of the actual purchased item in the top 100 produced. An idea could be to use as a score the average ranking of the purchased item in the list of the top 100 predicted purchased items. However the cased where the purchased item is not in the top 100 predictions has to be taken into account. The solution that was found was to compute a score using the expression below :

$$score = \begin{cases} 0 & \text{if the purchased item is not in the top 100} \\ \frac{1}{rank} & \text{otherwise} \end{cases} \tag{4.4}$$

where $rank$ is the ranking of the actual purchased item in the prediction. This expression gives a score for a single session, and with a validation set of sessions, the average of this score (tot_score in the jupyter notebook) can be computed. In this case the inverse of this score ($\frac{1}{tot\_score} = av\_rank$ in the notebook) can provide a measure equivalent to an average ranking, with a penalty for every session where the top 100 predictions does not include the actual purchased item.

4.4 Optimisation of the KNN search's parameters

The prediction algorithm relies the bucketed random projection algorithm to estimate the neighbors of a given session. Consequently there are 2 parameters that could be adjusted to optimize the prediction accuracy : K, the number of neighbors considered, and r, the bucket length parameter.

For a given configuration of these parameters the prediction algorithm can be evaluated for example by computing the average of the prediction score for a set of test sessions. Hence, a grid search can be done on a set of potential values of K and r, in order to find their optimal values. Unfortunately, due to a lack of time and ressources, this was not done in this project and K and R were both arbitrarily set.

4.5 Assessment of feature selection

In the project, 2 feature selection algorithms were programmed, and hence resulted in the choice of 2 potential subsets of optimal features. In order to compare these sets of features, the prediction algorithm was run twice on a set of 1000 test sessions, each time using a set of features provided by each algorithm.

The final score obtained for the mRmR features was 24.2, meaning that the purchased item is on average ranked at this place in the top 100 predictions. The score obtained with the features selected with forward selection is 15.7.

We conclude that in these conditions (with a limited amount of training and validation data, and without optimisation of the nearest neighbor search parameters), the results obtained with forward selection are significantly superior than the ones obtained with mRmR selected features.

5 Conclusion

In this project, a recommender system pipeline was implemented in a scalable way, in order to participate to the RecSys 2020 challenge.

In a first phase, the input data of the challenge was processed and a set of 45 features were desinged and engineered. In a second time 2 feature selection algorithms were implemented, and in a third section, a model generating the predictions asked by the challenge was implemented, using the collaborative filtering approach.

Finally, the developped model was used in order to generate predictions on a validation data, using the 2 set of features selected by each feature selection algorithm, and a measure of prediction accuracy was used in order to compare the results.

Bibliography

[1] "RecSys Challenge 2022", last consulted : June 1st 20200, url : http://www.recsyschallenge.com/2022

[2] Bontempi, Gianluca. 2021. Statistical Fondation of Machine Learning, Second Edition Handbook. ULB, Université Libre de Bruxelles, pp306-315.

[3] Ramírez-Gallego, S., Lastra, I., Martínez-Rego, D., Bolón-Canedo, V., Benítez, J.M., Herrera, F. and Alonso-Betanzos, A. 2017. Fast-mRMR: Fast Minimum Redundancy Maximum Relevance Algorithm for High-Dimensional Big Data. Int. J. Intell. Syst., 32: 134-152. https://doi.org/10.1002/int.21833

[4] Pauli Virtanen et al. (2020) SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17(3), 261-272.

[5] L.Ladha et al. 2011. Feature Selection Method and algorithms. International Journal on Computer Science and Engineering (IJCSE)

[6] Dietmar Jannach and Malte Ludewig. 2017. When Recurrent Neural Networks meet the Neighborhood for Session-Based Recommendation. In Proceedings of the Eleventh ACM Conference on Recommender Systems (RecSys '17). Association for Computing Machinery, New York, NY, USA, 306–310. https://doi.org/10.1145/3109859.3109872

[7] Jingdong Wang, Heng Tao Shen, Jingkuan Song, Jianqiu Ji. 2014. Hashing for Similarity Search: A Survey. in ArXiv. https://arxiv.org/abs/1408.2927

[8] "Extracting, transforming and selecting features", the MLlib guide of PySpark 2.3.1. Last consulted June 1st, 2022. https://spark.apache.org/docs/2.2.3/ml-features.html#bucketed-random-projection-for-euclidean-distance