PROJECT REPORT

WEB AND SOCIAL INFORMATION EXTRACTION

LUCA MOSCHELLA - 1594551 *FEDERICA SPINI* - 1588482

SUMMARY

INTRODUCTION

This document is the report of our Web and Social Information Extraction project. This section is composed by an explanation of the structure of the document and a brief description of the technologies we used.

DOCUMENT'S STRUCTURE

The first chapter of this document explains how the dataset data has been handled to be used for our purposes. We'll explain how the dataset is modelled in Java, how we acquired the various data and we'll make some analytics about what is contained in the dataset.

The second chapter deals with the assignment of categories to the users, that is a part of our first task. To do that we'll talk of how we semantically elaborated, thank to BabelNet and Babelfy, both Wikipedia pages and tweets in order to obtain a correct categorization.

The third chapter is about our fourth task, that is the extraction from Twitter of some information about the users of the s21 document, a process that we've done also for the users in s22 to have more information about them. In particular, we'll explain both how we downloaded the list of the friend and the list of the last tweets by the users' ids.

In the fourth chapter is about how we completed the first task, that is, how we represented objects using categories. Precisely, we didn't assign category, but a set of values that represent them to the users. The number of categories we found is enormous, but we decided to use all of them and embed them. In this way we considered both more common categories and the very rare ones, without discard anything. The method to embed this information is invented by us and exploits the information extracted by tweets, Wikipedia pages liked by the users, the Wikipedia pages which represent the users and also the users' friend categories.

The fifth Chapter is about the second, the third and the fourth tasks. Indeed, we've clustered together both the users from WikiMID and from s21 and s22 altogether, cause we previously built a common representation of them. So, we'll explain here the clusterization method we used (that is Mini-Batch K-Means), the parameters of the procedure with their meaning and the values we tried. Finally, always in this chapter, we evaluated the obtained clusters' quality using Calinski-Harabaz and Davies-Bouldin indexes.

The last chapter is about the fifth task and explain the approaches we used for the recommendation system. We made recommendation exploiting the latent categories representation of the users, trying to match them with the Wikipedia pages from s23, but also simply comparing the item from s23 with the ones contained in s22.

At the end we'll do some consideration about what we have done and possible improvements.

TECHNOLOGIES

We decided to use different technologies depending on the task we had to solve.

- **Java** [1]: to read the datasets, merge the information from the different sources, and generate a unique dataset
- **Python** [2]: to perform algebraic computations on sparse data, cluster and recommend items.
- **Twitter4j** [3]: to download information from Twitter
- **BabelNet** [4]: to find the categories and the domains associated to a given Wikipedia page
- **Babelfy** [5]: to disambiguate the text in the tweets
- **Scikit-learn** [6]: to normalize, reduce the dimensionality, cluster and evaluate the clusters
- **SciPy** [7]/**NumPy** [8]: to represent sparse matrices and perform vectorized computations
- **Fastutils** [9]: provides primitive collections in Java, avoiding the auto boxing and un-boxing of primitive types.

We didn’t use Lucene since we don’t perform any text search.

DATASET HANDLING

The first problem that we had to face was how to represent the data in a way that it could be easily queried without being slow. We found that Lucene was not adequate, since its main goal is to index text to perform fast textual searches. In our approach we do not perform any textual search. We could have used a database such as MongoDB, but we found out that modelling the relationships directly in Java, using primitive collections (provided by the fastutils library, developed in the University of Milan), yielded good enough performance for our purposes, and provided a very straightforward access to the information needed.

In the following sections, and in this report in general, we tried to abstract from the details of the implementation where possible, for this purpose there is the code documentation.

So, let's shortly see how we represented WikiMID, s21, s22 and s23. In other words, how we model our data.

MODELLING

The dataset model contains information about users, Wikipedia pages, interests and tweets. This dataset will be expanded with the information obtained from BabelNet, Babelfy and the twitter API, and we will talk about how this has been done in the following sections.

A quick overview of the main objects and their purpose:

- **UserModel:** it represents Twitter users. This java model keeps trace of the known following and followers of users, of his tweets, and of the Wikipedia pages he likes. Moreover, a given user could be a public figure and in this case, we define him "famous", and we keep track of the corresponding Wikipedia page that talks about him.
- **WikiPageModel:** it represents any Wikipedia page that is inherent to our Twitter content.
- **InterestModel:** it represents the interests of the Twitter users, so something that corresponds to a Wikipedia page, that we associate to each one of these objects.
- **TweetModel:** it represents the tweets written by our users. To each tweet the author and the corresponding interest are associated.

All these objects are collected together into an object of the class **Dataset**.

This model has been made serializable, so that it could be saved in cache, without the need to process the original datasets each time.

ACQUISITION

We've been given several dataset files. Three of them give information that are "independent" from the other ones. Indeed S21, S22 and S23 require operations ad hoc for them. On the other side we have four different files that compose the WikiMID dataset, that have to be integrated together.

We built a dataset model for each one of the given datasets and one that merges together all the datasets. The union of all the datasets will be the most useful model, since there is some intersection with the information available in the different datasets.

ANALYTICS

The following table shows how much information is contained in the different dataset files. WikiMID is the largest one and contains all type of information we previously described. Instead, S21 and S22 have just users and Wikipedia pages, so the statistics we computed on them aren't interesting: we can just see that the Wikipedia pages in S22 correspond

to about 47 000 categories that are a subset of the pages in WikiMID. Indeed, the number of pages contained in WikiMID and in Total is the same.

About Total dataset: it is the union of the information contained in WikiMID, S21 and S22. It is not a simple union of the objects but contains also the relationships among the users in the three different files, so that if a user in S21 follows a user in WikiMID, the Total dataset takes it into account. Moreover, in the Total dataset there is the information downloaded from Twitter, so that the categories of the disambiguated tweets are associated to the corresponding users.

		WIKIMID	S21	S22	TOTAL
USERS	Number of common users	443346	1500	500	445343
	Number of famous users	58789	0	0	58789
	Total number of users	502135	1500	0	504132
TWEETS STATS	Total number of tweets	1758685	0	0	1758685
	Greatest number of tweets	13462 for 1 user	0 for 1500 users	0 for 500 users	13462 for 1 user
	Smallest number of tweets	0 for 214839 users	0 for 1500 users	0 for 500 users	0 for 216836 users
	Median number of tweets	1 for 161520 users	0 for 1500 users	0 for 500 users	1 for 161520 users
	Mean of number of tweets	3,5024	0	0	3,4885
	Number of tweets per user variance	1360,5557	0	0	1355,2146
	Number of tweets standard deviation	36,8857	0	0	36,8132
FRIENDSHIP STATS	Total number of friendships	66129104	0	0	66486104
	Greatest number of friendships	106757 per 1 user	0 for 1500 users	0 for 500 users	107263 per 1 user
	Smallest number of friendships	0 per 39452	0 for 1500 users	0 for 500 users	0 per 39313
	Median number of friendships	39 per 2703 users	0 for 1500 users	0 for 500 users	39 per 2728 users
	Mean of number of friendships	131,6959	0	0	131,8823
	Number of friendships per user variance	453893,3085	0	0	456384,8489
	Number of friendship standard deviation	673,716	0	0	675,5626
FOLLOWING STATS	Total number of followings (incoming edges)	33067306	0	0	33245823
	Greatest number of followings	106757 per 1 user	0 for 1500 users	0 for 500 users	107263 per 1 user
	Smallest number of followings	0 per 443346 users	0 for 1500 users	0 for 500 users	0 per 427424 users
	Median number of followings	0 per 443346 users	0 for 1500 users	0 for 500 users	0 per 427424 users
	Mean of number of followings	65,8534	0	0	65,9467
	Number of followings per user variance	447614,1076	0	0	450112,9157
	Number of followings standard deviation	669,0397	0	0	670,9046
FOLLOWER STATS	Total number of followers (outcoming edges)	33067306	0	0	33245823
	Greatest number of followers	2388 per 1 user	0 for 1500 users	0 for 500 users	2388 per 1 user
	Smallest number of followers	0 per 96853	0 for 1500 users	0 for 500 users	0 per 96975 users
	Median number of followers	22 per 4190 users	0 for 1500 users	0 for 500 users	22 per 4211 users
	Mean of number of followers	65,8534	0	0	65,9467
	Number of followers per user variance	13887,9	0	0	13890,8805
	Number of followers standard deviation	117,8469	0	0	117,8596
Total number of interests		282303	0	0	282303
Total number of Wikipedia pages		143257	0	14657	143257
Total number of categories		169611	0	47355	195697
Total number of domain		34	0	34	34

SEMANTIC PROCESSING

In this chapter we'll explain how we processed our information from the semantic point of view. First of all, we'll see BabelNet and Babelfy, that we used to associate categories to the Wikipedia pages and tweets' texts. Then we'll explain more in detail how we used them on the dataset files.

BABELNET [4]

BabelNet is both a multilingual encyclopaedic dictionary, with lexicographic and encyclopaedic coverage of terms, and a semantic network which connects concepts and named entities in a very large network of semantic relations, made up of about 15 million entries, called Babel synsets. Each Babel synset represents a given meaning and contains all the synonyms which express that meaning in a range of different languages.

BABELFY [5]

Babelfy is a unified, multilingual, graph-based approach to Entity Linking and Word Sense Disambiguation based on a loose identification of candidate meanings coupled with a densest subgraph heuristic which selects high-coherence semantic interpretations.

A language-agnostic setting is available. In this setting, Babelfy considers all 271 languages without assuming or trying to infer the language of the input text giving the further possibility of annotating text written in multiple languages.

We think that the agnostic setting is the best possible approach in trying to disambiguate the tweets, since it is common writing tweets in multiple language (e.g. in English and in Italian, depending on the context).

EXTRACTION OF CATEGORIES AND DOMAINS

We used these semantic resources to extract categories and domains from the available information. The number of domains is very small (34), the number of categories is very high (BabelNet obtains most of them through the processing of the Wikipedia's categories). So, there is a great difference between categories and domains: the categories are very different, they can be more or less generic (for example a singer can be in the categories "1940 births", "English rock singers" and "Murdered male actors"), while the domains are very few (every type of entertainment activity is simply included into "Media" domain).

WIKIMID

In the WikiMID dataset there are Wikipedia pages expressed as the interest in a tweet or as the personal page of a famous user. It turns out that it is possible to associate to each Wikipedia page a Synset. Moreover, we know that in BabelNet each Synset may be in one or more *domains* and in one or more *categories*.

So, we are able to obtain the categories, or domains, of each tweet and the categories, or domains, of the page of famous users.

This information will be used to create an embedding of the users and the pages.

TWEET CONTENTS

We have been given users IDs and asked to download their information through the Twitter API. We downloaded some tweets for each user.

We have been able to disambiguate the downloaded tweets using Babelfy in AGNOSTIC mode, obtaining Synsets, whichever was the language of the user/tweet.

So, yet again, we're able to obtain the categories, or domains, of each Tweet.

Actually, we didn't disambiguate each tweet. It turns out that merging together in a single string the tweets of a given user (forcing each tweet in a new paragraph) gives more context to Babelfy, which can better disambiguate the words.

ITEMS

Obviously, we have been able to associate categories/domains even to the items in the dataset S22 and S23, since they are Wikipedia pages too.

LIMITATIONS

This approach however has some negative aspects: not all the Wikipedia pages have an associated synset, and not all synsets have associated categories.

Moreover, a problem with the encoding of non-ascii characters in the S22 and S23 datasets caused the loss of some information, since we haven't been able to find the synset of some pages that actually have a synset in BabelNet (around 50 pages).

More insight on the data is available in the analytics section.

TWITTER INFORMATION EXTRACTION (TASK 4)

One of our tasks (the fourth) was to use the Twitter API to download information for the Twitter users of the dataset S21. In this section we explain which type of information we have chosen to download, and how we have done it.

WHAT WE DOWNLOADED WITH TWITTER4J

The first thing we've found useful has been the *following relationships* of the users of S21. Twitter4j allows to download the complete list of the twitter ids of the **following** of a user. Almost all the users of S21 have some *follow-out* friends that already are in WikiMID, so they're a precious information, so they will be easier to cluster. Moreover, often these users that already are in WikiMID represent famous people, so they can be directly associated with a personal page.

The other Twitter information we decided to download are the **tweets** of the users. Indeed, thanks to BabelNet API we are able to associate a tweet with a series of categories and this is precious to merge the S21's users in clusters with the WikiMID users. To do it we had to deal with two problems:

- Some of the users present in the dataset have been deleted from Twitter, their profile is private, or they are suspended. So, their information is not available. These exceptions have been captured, but we ignored the corresponding users.

- The time these operations require is long due to the limits of the free Twitter API. In particular, the requests for a user's following can be done every 15 minutes, so for 1500 users we need more than a day.

Downloading the tweets' texts thanks to the Twitter API and using Babelfy on them, allowed us to obtain a set of synsets [4]. Because of a synset is associated to a Wikipedia page, we extracted the categories from the corresponding pages as previously explained. These new categories are stored in the model of the author of the tweet and we'll use them as tweets categories to assign some categories also to the users.

HOW WE EXTRACTED THE FEATURES

The classes in our code that have to manage this part are the ones in the package **TwitterOperation**, with the extractor that makes the correct Twitter4j requests and the two type of responses, one for the friends of the users, one for the tweets. To have more details about the implementations see the documentation of the code.

REPRESENTATION WITH LATENT CATEGORIES

We used BabelNet to extract the categories/domains, and we represented every object in the dataset in the space of the categories, merging together all the available information. The way to do that is explained in the following sections. The number of dimensions, that is the number of categories obtained from the data, is about $190'000$.

We reduced the dimensionality of the space to 300, normalizing before and after the dimensionality reduction. In this way we didn't deal with real categories, but with "latent" categories.

WIKIPEDIA PAGES

Each Wikipedia page is represented as a vector of numbers: one point in the space of the categories. Note that this vector will be very sparse, because a Wikipedia page has only a bunch of all the $190'000$ categories.

TWEETS

In the WikiMID dataset each tweet is associated to only a Wikipedia page. So, each tweet can be represented as a point in the space of categories too.

Note that the tweets disambiguated through Babelfy don't fall in this category, since they are merged together before the disambiguation.

USERS (TASK 1)

To make a correct representation of the users, we've decided to use BabelNet. Moreover, as we have already said:

- How to associate categories to the Wikipedia pages
- How to associate categories to tweets

These two things are really important for the method we developed to represent users as vectors of categories.

In this section we will see how we embedded all the information available for a given user in a vector. Now the challenging part of the chapter comes.

The vector representation of a given user must embed the information about:

- The categories of the tweets that the user wrote
- The categories of the Wikipedia pages that the user likes
- The categories of the Wikipedia page that represents a user. As we said, some of our users are public figures and the categories of the pages that talk about them are important, but this is a different concept compared to the pages they like as Twitter users
- The categories of the users following: if a user admires and follows someone, we expect that he's interested also in his categories (it's a recursive formulation)

Moreover, we thought that these various characteristics are all important, but not in the same way. Given a user, the Wikipedia pages he likes seemed to us a stronger evidence of his interests with respect to his tweets.

Another example is, for famous people, let's say a singer, a personal page surely represents him, and we want to take it into account. However, it contains the categories of the "public figure" of the singer. If the singer is also an amateur winemaker, even if nobody knows about it, he as a user may appreciate pages wine related and makes tweets about this topic. For this reason, we gave different weights to the categories of different types (as for the categories of the personal pages which have less importance than the liked Wikipedia pages).

Now let's see in detail how we embedded this information together. We defined three matrices:

- Let T be the tweet matrix. It contains a row for each user and a column for each existing category. A row contains the categories associated to all the tweets of a user. So, $T_{i,j} = x$ means that the user i has x times the category j in his tweets.

$$\boldsymbol{T}_{i,j} = \begin{cases} 0, & \text{if the category } j \text{ does not appear in the tweets of user } i \\ x, & \text{if the category } j \text{ appears } x \text{ times in the tweets of user } i \end{cases}$$

- Let P be the personal page matrix. It contains a row for each user and a column for each existing category. So, $P_{i,j} = x$ means that the user i has x times the category j in his personal page. Note that each value can be only 0 or 1, because a user can have at most only one page. Moreover, consider the fact that most of the entries of this table are made of zeros, because only a little percentage of the users has a personal page, and each page doesn't have a lot of categories.

$$\boldsymbol{P}_{i,j} = \begin{cases} 0, & \text{if the category } j \text{ does not appear in the personal page of user } i \\ 1, & \text{if the category } j \text{ appears in the personal page of user } i \end{cases}$$

- Let L be the liked item matrix. It contains a row for each user and a column for each existing category. So that $L_{i,j} = x$ means that the user i has x times the category j in the Wikipedia pages he likes.

$$\boldsymbol{L}_{i,j} = \begin{cases} 0, & \text{if the category } j \text{ does not appear in the pages liked by user } i \\ x, & \text{if the category } j \text{ appears x times in the pages liked by user } i \end{cases}$$

- Let F be the friendship matrix, it is a boolean adjacency matrix. It contains a row for each user and a column for each user too. A row represents the friendship relation, so that the user that corresponds to a row is a follower of the users corresponding to the columns set to one. So, $F_{i,j} = x$ says if the user i follows or not the user j.

$$\boldsymbol{F}_{i,j} = \begin{cases} 1, & \text{if user } i \text{ follows the user } j \\ 0, & \text{otherwise} \end{cases}$$

Now we can define M as a matrix that contains a row for each user and a column for each existing category, each i^{th} row will be the vector representation of the i^{th} user. A row represents the categories associated to all the various objects related to a user: so $M_{i,j} = x$ means that the characteristic j has importance x for the user i. Note that the k^{th} power of and adjacency matrix gives the friends at distance k.

$$\boldsymbol{M} = \alpha\boldsymbol{T} + \beta\boldsymbol{P} + \gamma\boldsymbol{L} + \sum_{d=i}^{+\infty} \Delta^d (\alpha' \boldsymbol{F}^d \boldsymbol{T} + \beta' \boldsymbol{F}^d \boldsymbol{P} + \gamma' \boldsymbol{F}^d \boldsymbol{L})$$

Where:

- α is the weight (or the importance) given to the categories of the tweets, α' refers to the friend's categories
- β is the weight (or the importance) given to the categories of the personal Wikipedia pages, β' refers to the friend's personal pages
- γ is the weight (or the importance) given to the categories of the liked Wikipedia pages, γ' refers to the friend's liked pages
- Δ is the weight (or the importance) given to the categories of the following. Note that Δ depends on the connectivity of the friendships graph. Since $\Delta \in [0, 1]$ the operation Δ^d forces an exponential decay of the information as it travels from friend to friend, i.e. to a given user the information given by another user at distance k is exponentially less relevant as k increases.

CURSE OF DIMENSIONALITY

Until now, in describing how we've dealt with the problem of assign categories to the users, we've talked of matrices whose dimensions often depends on the categories number.

Once we obtained M, a $500k \times 190k$ matrix, we reduced its dimensionality to $500k \times m$.

To reduce the dimensionality, we defined the following pipeline in sklearn:

- Normalize the rows such that the norm is one
- Reduce the dimensionality with TruncatedSVD [10] (maintain only the m most important singular values)
- Normalize again the rows such that the norm is one

The definition of such pipeline, instead of applying manually each fit and transformation, will help when dealing with test data. It will be enough to retrieve the already fit pipeline and just transform the new data.

WHY THIS REPRESENTATION?

EMBEDDINGS

In machine learning almost every system is built upon a fundamental concept: the embeddings, a computer-friendly representation of objects (i.e. a dense vector of numbers). The embeddings should depict the objects as closely as possible: two embeddings should be similar if and only if the objects that they represent are similar. It is hard to define what *similar* means in complex real-world contexts and directly find embeddings that satisfy the required properties.

Thus, the first problem that generally must be solved is *how to represent objects.*

We embedded in a single vector all the known information about a single object, moreover, that information is expressed in term of semantic categories. This is an extremely general approach: almost every object can be described by a set of semantic categories, and thus it can be associated to an embedding with this approach. This idea will lead us to a general-purpose recommendation system.

We wanted to embed in a single vector all the information relative to a given single user i, let's call this information I_i. So, this information includes his tweets, possibly the items that he likes and his personal page. Moreover, since these sources of knowledge could have different importance, we decided to weight them accordingly.

Moreover, we have another important information: the structure of the graph of the friendships. We exploited this graph, enforcing the idea that in the information I_i there are components of information defined as $f(k, I_j)$ for each j^{th} user at distance k from i. Note that the weight of the components should decrease exponentially as the distances increase, and, that the parts that make up the information should be weighted differently if they are referring to the *proprietary* or to a *friend*, since they have different roles (e.g. my personal page is not so relevant in describing what I like, the personal page of a famous singer that I'm following is very relevant in doing so). Hence the proposed formula. We used the interesting property of adjacency matrices: F^d contains in row i the friends of i at distance d. Without this property it would have been much more difficult to be able to perform these computations.

Note that the process, actually, is not iterative: we use the information of the j^{th} user while defining the i^{th} user, but when the embedding of the j^{th} is computed it is *not* used to compute the embedding of the i^{th} user. Basically the i^{th} embedding is built using the information known about the friends, not their embeddings. It would be interesting to try this approach and iterate the process until convergence (in this way we would get a PageRank-like algorithm).

NOTES ABOUT EFFICIENCY

We are dealing with big matrices. The adjacency matrix has dimension $500k \times 500k$, even if it's a boolean matrix, storing it entirely in memory would mean to require 58 GB of RAM (assuming no overhead). The situation is even worse when we consider the other matrices that have dimension $500k \times 190k$, each one of them would require 353GB of RAM (assuming no overhead) since they contain float32 and not booleans.

It's obvious that it is not feasible to store in memory those dense matrices. Fortunately, we can exploit one common property: they are sparse matrices.

We used SciPy. Sparse to work with these matrices, and we were able to fit them, and the computations, in a 32GB machine.

However, using this approach lead to some limitations: not all the functions made available by sklearn accept a sparse matrix, and not all the functions scale well enough memory wise. So, our choices were limited by the available options. One of the main motivations that lead us to continue the project in python and not in java, was that java hasn't a good enough, easy to use, library to handle sparse matrices, algebraic computations and machine learning tools. Instead in python we could leverage SciPy, NumPy and sklearn that, together, met all our needs.

ADJACENCY MATRIX EXPONENTIATION

Must be noted, as already explained, that performing the i-power of the adjacency matrix gives us the nodes at distance i. Given the structure of the graph that one could expect from twitter users (some people have a lot of edges, following the Zipf law) it is straightforward to see that the initially sparse adjacency matrix would become exponentially less sparse at each exponentiation. On our 32GB machine we were able to compute only the square of the adjacency matrix. Probably the operations could be optimized in some way, but we think that the importance of information decreases exponentially as it travels from friend to friend, and we enforced this behaviour setting an exponential decay in the formula. So, what we computed should give us a good enough approximation of an optimal computation (where higher powers of the adjacency matrix are computed).

CLUSTERIZATION (TASK 2, 4)

We have a matrix M where the i^{th} row represents the i^{th} user. The number of rows has been reduced using the Singular Value Decomposition. The values have been normalized before and after the decomposition (so, the ranking given by the cosine distance and the Euclidean distance will be the same)

This new matrix, while still being quite big, is much more manageable: it is possible to feed it to a clustering algorithm to perform the clusterization.

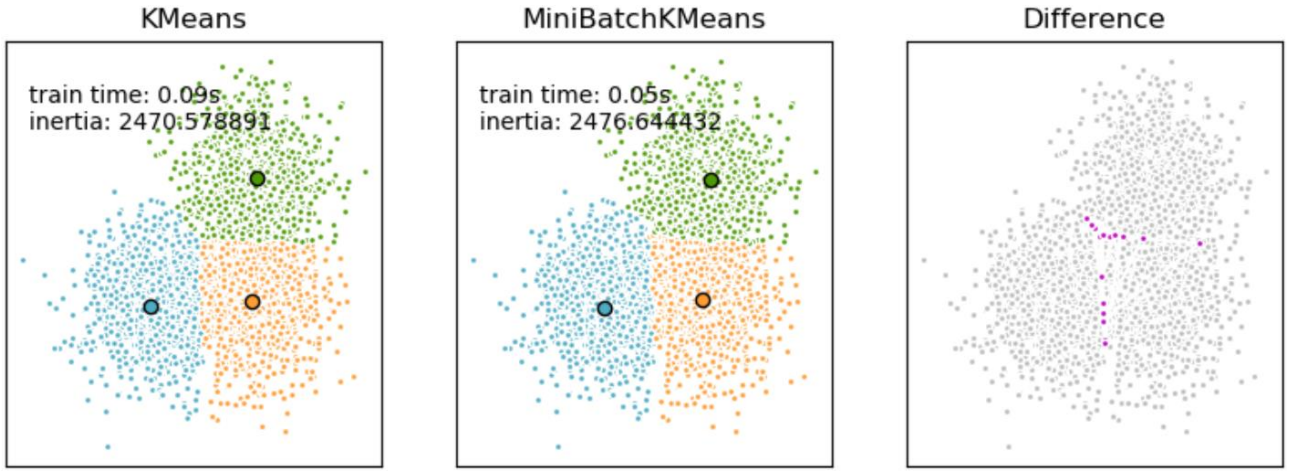
To note that in this matrix all the users from all the dataset's files are present. It has been done to simplify the algebraic computations previously explained. However, it could have been possible to cluster the users in S21 afterward: it would have been enough to compute the representation of each user, it can be done repeating the same procedure, then selecting only the correspondent rows in the final matrix and using the already fit clusters to predict the correct cluster for each user.

MINI BATCH K-MEANS [11]

The MiniBatch K-Means is a variant of the K-Means algorithm which uses mini-batches to reduce the computational time, while still attempting to optimise the same objective function. Mini batches are subsets of the input data, randomly sampled in each training iteration. These mini-batches drastically reduce the amount of computation required to converge to a local solution. In contrast to other algorithms, that reduce the convergence time of k-means, mini-batch k-means produces results that are generally only slightly worse than the standard algorithm.

The algorithm iterates between two major steps, like vanilla k-means. In the first step, samples are drawn randomly from the dataset, to form a mini-batch. These are then assigned to the nearest centroid. In the second step, the centroids are updated. In contrast to k-means, this is done on a per-sample basis. For each sample in the mini-batch, the assigned centroid is updated by taking the streaming average of the sample and all previous samples assigned to that centroid. This has the effect of decreasing the rate of change for a centroid over time. These steps are performed until convergence or a predetermined number of iterations is reached.

Minibatch K-Means converges faster than K-Means, but the quality of the results is reduced. In practice this difference in quality can be quite small, as shown in the example and cited reference.



CLUSTERIZATION PARAMETERS

The following table describes the various parameters that we must choose to complete all the clusterization procedure on the users of both WikiMID and S21, completing the first and the fourth tasks. The used algorithm, as we already said, is MiniBatch K-Means and the table contains also the hyperparameter to run it.

Pay attention to the "value" column of the table: it indicates which values has been used for each parameter. Sometimes a set of values is contained, and it means that we have tried different values.

Probably a more complete tuning of all these parameters can lead to better results, but it requires time and because of we already have a good evaluation, we preferred to focus on other aspects of the project.

FEATURE NAME	VALUE	DESCRIPTION
Dataset	The union of WikiMID, S21 and S22	The dataset from which we have extracted information for cluster
Dimension	Complete	If this value is "complete" we've used all the data from the dataset. If it is "small" we've limited the read values, to increase the speed of the clusterization
Cluster_over	Categories	The clusterization can be based on categories values or on domain value
Max_user_distance	2	The neighbour of at most distance=2, considering only outcoming edges, affect the clusterization of the users
Tweet_importance	0.3	The weight of the user's tweets in clustering him
Personal_page_importance	0.15	The weight of the user's personal Wikipedia page in clustering him
Liked_item_importance	0.55	The weight of the Wikipedia pages liked by the user in clustering him
Rate_of_decay	{0.5, 0.15}	The weight of the user's following at $distance = i$, where at distance I the weight is $(rate\_of\_decay)^i$
Follow_out_tweet_importance	0.15	The weight of the user's followings tweets in clustering him
Follow_out_personal_page_importance	0.55	The weight of the user's followings personal Wikipedia page in clustering him
Follow_out_liked_items_importance	0.3	The weight of the user's followings liked Wikipedia pages in clustering him
Reducer	Truncated_svd	The type of reducer that is used
Matrix_dimensionality	{100, 300}	The dimensionality of the characteristic's representation.
Clusterer	Minibatch_kmeans	The cluster strategy used
N_clusters	{50, 100, 150, 300, 350, 400, 450, 500, 550, 600, 750, 800}	The number of clusters that are created
Max_iter	1000000	The maximum number of iterations
Batch_size	5000	The amount of data that are processed each iteration
Max_no_improvement	10000	The maximum number of iteration that can be executed without an improvement
Init_size	50000	The number of the initial group of user that are clustered

N_init	100	The maximum number of users combination that are considered as starting centroids
Reassignment_ratio	1e-06	The ratio of the allowed deviation from the centroids of the previous centroids

EVALUATION (TASK 3)

In order to measure the quality of the clusters, in absence of a ground truth (i.e. the correct clusterization for at least a subset of the data), we had to rely on unsupervised measures.

Two of the most common such measures are the Calinski-Harabaz Index and the Davies-Bouldin Score.

CALINSKI-HARABAZ INDEX [12]

Also known as the Variance Ratio Criterion, a higher Calinski-Harabaz score relates to a model with better defined clusters.

Fixed k, the score s is given as the ratio of the between-clusters dispersion mean and the within-cluster dispersion.

To note that the score is higher when clusters are dense and well separated, which relates to a standard concept of a cluster. Moreover, the computation is fast: an essential requisite when dealing with a lot of data on a single machine with limited time.

DAVIES-BOULDIN INDEX [13]

A lower Davies-Bouldin Index relates to a model with a better separation between clusters, where the minimum value is zero.

The index is defined as the average similarity between each cluster C_i for $i = 1..k$ and its most similar one C_j.

The usage of centroid distance limits the distance metric to the Euclidean space, that it isn't a problem in our case since we are dealing with a Euclidean space.

RESULTS

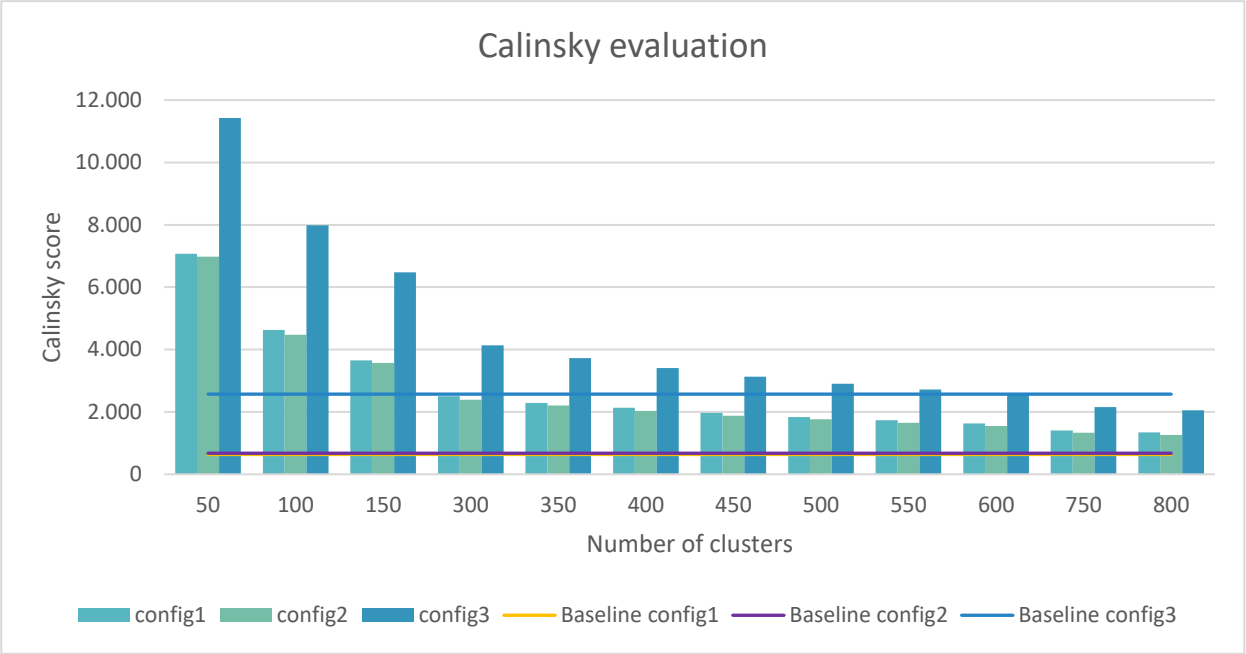
We have run the cluster procedure with three different configurations of the parameters:

- Matrix_dimensionality = 300 and rate_of_dacay = 0.5: let's call this *config1*
- Matrix_dimensionality = 300 and rate_of_dacay = 0.15: let's call this *config2*
- Matrix_dimensionality = 100 and rate_of_dacay = 0.15: let's call this *config3*

Every time we tried all the number_of_clusters values that are indicated in the parameters table. So, thank to the two algorithms we have explained, it is possible to visualize the variation of the obtained clusters quality.

CALINSKI-HARABAZ INDEX EVALUATION

In the chart below, we can see the Calinski-Harabaz score obtained with each cluster configuration, changing the number of clusters that are obtained. Note that using the third configuration, that has a lowest dimensionality of the categories' representation, performances increase. Moreover, the less is the number of the clusters, the higher is the quality obtained by them. We can see also that config3, with more than 600 cluster, goes under the baseline in its quality measure.

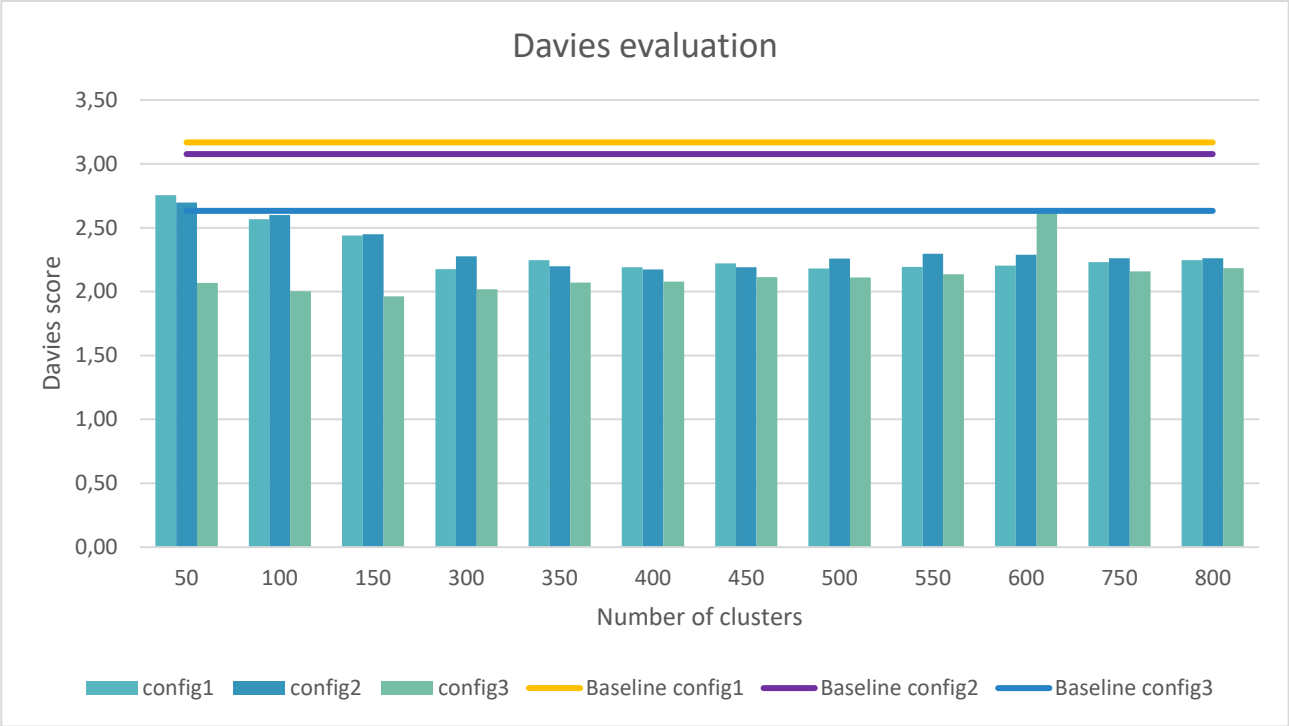


The following table shows in detail the scores produced by each configuration, with all the numbers of clusters tried. Note that the more a box is red the worst the score. On the other hand, the more a box is green, the better the score.

NUMBER OF CLUSTERS	config1	config2	config3
50	7.070,2326	6.976,5030	11.428,4984
100	4.624,0151	4.477,0698	7.984,5053
150	3.657,4498	3.571,6759	6.480,4472
300	2.505,0718	2.394,1535	4.131,1040
350	2.284,1925	2.205,5930	3.726,8157
400	2.133,4000	2.028,9906	3.403,0349
450	1.968,6313	1.873,5034	3.125,7205
500	1.839,3874	1.762,0726	2.900,3937
550	1.734,5088	1.655,8799	2.716,4693
600	1.633,3835	1.551,5312	2.570,6077
750	1.403,9857	1.330,2392	2.156,4655
800	1.339,5846	1.265,7260	2.051,6902
Baseline	636,8426	675,6398	2.570,6077

DAVIES-BOULDIN INDEX EVALUATION

In the chart below, we can see the Davies' measure scores obtained with each cluster configuration, changing the number of clusters. Note that using the third configuration, as for the Calinski-Harabaz, has a lowest dimensionality of the categories' representation, performances increase. Indeed, this time, lower scores are better. Note that the best scores are obtained with around 150 clusters for config3, 400 clusters for config1 and config2, while using Calinski-Harabaz measure the best choice is always number_of_clusters = 50.



The following table shows in detail the scores produced by each configuration, with all the numbers of clusters tried. Note that the more a box is red the worst the score. On the other hand, the more a box is green, the better the score.

NUMBER OF CLUSTERS	config1	config2	config3
50	2,7548	2,6982	2,0678
100	2,5660	2,5991	2,0039
150	2,4396	2,4492	1,9624
300	2,1770	2,2751	2,0190
350	2,2469	2,1981	2,0704
400	2,1912	2,1729	2,0785
450	2,2206	2,1906	2,1123
500	2,1816	2,2587	2,1106
550	2,1936	2,2973	2,1348
600	2,2033	2,2878	2,6319
750	2,2299	2,2618	2,1578
800	2,2459	2,2600	2,1839
Baseline	3,1679	3,0764	2,6319

RECOMMENDATION SYSTEM (TASK 5)

The approach used to build the recommender system is very general: it makes possible to recommend, obtaining a score, whichever item that can be expressed in terms of BabelNet categories, without any further training (e.g. any page of Wikipedia, any synset in BabelNet that has categories associated).

To wrap up: we have the vector representation of users and pages in a k-dimensional space of latent categories, where k is much less than the number of categories.

In most of the recommendation systems we have that the representation of users and items are expressed in two different spaces, so one can think of computing only the user-user similarity (the so-called user-based recommendation system) and item-item similarity (the so-called item-based recommendation system).

In our case it's different: since both the users and the items lie in the same space, we can compute the user-item similarity directly, yielding another type of recommender system.

As a measure of distance, we decided to use the cosine distance.

REPRESENTATION WITH LATENT CATEGORIES

How to represent the item, i.e. the Wikipedia pages, in the same space of the users' representations? It is possible to build an analogue matrix to the one built for the users.

We decided to keep the construction of this matrix simple, but it would have been interesting to explore the possibility to introduce the information of pages near i in the representation of the page i, like we did for the users.

So, to represent a page i, we simply take its vector in the space of the categories and reduce its dimensionality. It is important to note that in order to obtain results comparable with the users' representations, the same transformation must be applied. Since the implementation of TruncatedSVD involves some random component, the same pipeline (fit on the users' data) must be used.

ITEM-BASED RECOMMENDATION SYSTEM

We know the preference of a given user, and we are asked to find which ones, from a selection of pages, he may like.

Since we have a nice vector representation for the pages, it is easy to give a ranking to the possible pages.

If a user likes n pages, and we must choose which one from m given pages we should recommend, a function to reduce the distance of each page to the others must be chosen.

We decided to implement three different naive possibilities:

- Suggest the pages that have the smallest minimum distance to one of the pages liked by the user. In this way it can also happen that if a user likes mostly pages about a topic and a different page, about something else, a page very similar to the unusual one may be suggested.
- Suggest the pages that have the smallest medium distance to all the pages liked by the user. In general, this seems the more balanced method, because the more the user likes pages about a given topic, the more this topic will be considered in the mean.
- Suggest the pages that have the smallest maximum distance from all the pages that are liked by the user. But in this way pages that in general are nearer to all the pages are preferred to the ones that can be very far from some liked pages, but also very near to some other pages.

USER-ITEM RECOMMENDATION SYSTEM

Since both the users and the pages are in the same space, there is a much simpler approach that one can think of: just recommend the pages that are most like the user.

We are not directly using the information about the pages that the user likes. Actually, we could have built the same recommendation system even without that information, contained in S22!

Representing each object in function of its categories makes possible to obtain a general-purpose recommendation system that can recommend an extremely wide range of items, without directly knowing what items the user likes.

Note that in this case the difference in speed may be irrelevant, but we think that if a user likes a lot of pages and we must suggest something chosen from a huge set the difference in performances could be relevant. Indeed, if the user likes r pages, and the pages to suggest must be chosen from m pages: the item-based recommendation (implemented in the naive way, without preprocessing) takes $O(r \cdot m)$ while the user-item approach takes $O(m)$.

RESULTS

Analysing the recommendation system is more difficult than evaluate the clusterization system. The reason is that now we haven't a measure or a groundtruth, so the only thing we can do is to give a look and try to understand if the recommended items are coherent with users' tastes.

First, we have looked for users with preferences well defined. It was not easy, because most of our users like a lot of items that in general are famous and we don't know a lot of items or too famous items are not very significative (Rihanna's page appears in 92 users' preferences).

USERS

The first user we considered is the one with id = 1057516135. He likes rock and metal bands, and something about videogames. We'll call him *user1*.

The second and the third users we consider clearly are Italian. One of them, 104239528, likes also a lot of international public figures and doesn't have very clear tastes. It can be interesting compare him to another user, 101532373, that is more focused only on Italian items.

We'll call them *user2* and *user3*.

Finally, we considered the user 101935414, that likes a lot of Japanese items and probably he's Japanese too. We'll call him *user4*.

1057516135	WIKI:EN:Funko
1057516135	WIKI:EN:Overwatch
1057516135	WIKI:EN:Simon's_Cat
1057516135	WIKI:EN:Muy_Interesante
1057516135	WIKI:EN:Amazon.com
1057516135	WIKI:EN:Alex_Hirsch
1057516135	WIKI:EN:Jack_Flash
1057516135	WIKI:EN:Son_of_God
1057516135	WIKI:EN:EMP_Merchandising
1057516135	WIKI:EN:AC/DC
1057516135	WIKI:EN:ZZ_Top
1057516135	WIKI:EN:Lynyrd_Skynyrd
1057516135	WIKI:EN:Led_Zeppelin
1057516135	WIKI:EN:The_Who
1057516135	WIKI:EN:The_Doors
1057516135	WIKI:EN:Tony_Iommi
1057516135	WIKI:EN:Led_Zeppelin
1057516135	WIKI:EN:Pink_Floyd
1057516135	WIKI:EN:Five_Finger_Death_Punch
1057516135	WIKI:EN:Intimissimi
1057516135	WIKI:EN:Iron_Maiden
1057516135	WIKI:EN:Ozzy_Osbourne
1057516135	WIKI:EN:Blizzard_Entertainment
1057516135	WIKI:EN:Metallica
1057516135	WIKI:EN:Steven_Tyler
1057516135	WIKI:EN:The_Rolling_Stones
1057516135	WIKI:EN:Aerosmith

The complete list of the items that user1 likes. We can see that he follows a lot of bands.

101935414	WIKI:EN:Rinko_Kikuchi
101935414	WIKI:EN:Maria_Sharapova
101935414	WIKI:EN:Yoshitomo_Nara
101935414	WIKI:EN:Daoko
101935414	WIKI:EN:Jamie
101935414	WIKI:EN:Nina_Kraviz
101935414	WIKI:EN:Teru
101935414	WIKI:EN:Richie
101935414	WIKI:EN:Instagram
101935414	WIKI:EN:Lady_Gaga
101935414	WIKI:EN:Koji_Uehara
101935414	WIKI:EN:SoundCloud
101935414	WIKI:EN:4hero
101935414	WIKI:EN:Allyson_Felix
101935414	WIKI:EN:Mixcloud
101935414	WIKI:EN:Mixlr
101935414	WIKI:EN:Gackt
101935414	WIKI:EN:The_Japan_Times
101935414	WIKI:EN:YouTube
101935414	WIKI:EN:Naoko_Yamazaki
101935414	WIKI:EN:Ji-hae
101935414	WIKI:EN:Ninja_Tune
101935414	WIKI:EN:Yukihiro_Takahashi
101935414	WIKI:EN:Mirai_Nagasu

A subset of the items that are liked by user4. He likes also generic famous items, like Lady Gaga or Instagram, but he clearly is Japanese.

101532373	WIKI:EN:President_of_Italy
101532373	WIKI:EN:Mental_Floss
101532373	WIKI:EN:Siae
101532373	WIKI:EN:RTL_102.5
101532373	WIKI:EN:Alessandra_Amoroso
101532373	WIKI:EN:Francesca_Michielin
101532373	WIKI:EN:Brooklyn_Bowl
101532373	WIKI:EN:Danilo_Petrucci
101532373	WIKI:EN:Simona_Ventura
101532373	WIKI:EN:Francesca_Archibugi
101532373	WIKI:EN:Daniel_Ek
101532373	WIKI:EN:Adnkronos
101532373	WIKI:EN:Emma_Marrone
101532373	WIKI:EN:Luca_Argentero
101532373	WIKI:EN:Pierfrancesco_Favino
101532373	WIKI:EN:Fabio_De_Luigi
101532373	WIKI:EN:Federica_Pellegrini
101532373	WIKI:EN:Fabio_Aru
101532373	WIKI:EN:Vincenzo_Nibali
101532373	WIKI:EN:Sky_TG24
101532373	WIKI:EN:TgCom24
101532373	WIKI:EN:Andrea_Ranocchia
101532373	WIKI:EN:Nicola_Formichetti
101532373	WIKI:EN:Upworthy
101532373	WIKI:EN:BuzzFeed
101532373	WIKI:EN:Giulio_Base
101532373	WIKI:EN:Stephan_Moccio
101532373	WIKI:EN:Stromae
101532373	WIKI:EN:Napster
101532373	WIKI:EN:Marco_Werman
101532373	WIKI:EN:Luca_Parmitano
101532373	WIKI:EN:Joe_Bastianich
101532373	WIKI:EN:John_Lurie
101532373	WIKI:EN:RAI1

A set of the items liked by user3: we can see that he's focused mostly on Italian items.

104239528	WIKI:EN:Paola_Turci
104239528	WIKI:EN:Tiziano_Ferro
104239528	WIKI:EN:Massive_Attack
104239528	WIKI:EN:Giles_Duley
104239528	WIKI:EN:Sheryl_Sandberg
104239528	WIKI:EN:Mark_Zuckerberg
104239528	WIKI:EN:Benji
104239528	WIKI:EN:Federica_Pellegrini
104239528	WIKI:EN:Tania_Cagnotto
104239528	WIKI:EN:Kent_Moran
104239528	WIKI:EN:Ben_Affleck
104239528	WIKI:EN:Rosario_Dawson
104239528	WIKI:EN:The_Intercept
104239528	WIKI:EN:Jeremy_Scahill
104239528	WIKI:EN:Glenn_Greenwald
104239528	WIKI:EN:CBS_News
104239528	WIKI:EN:Sky_News
104239528	WIKI:EN:Al_Jazeera_English
104239528	WIKI:EN:NBC_News
104239528	WIKI:EN:The_Wall_Street_Journal
104239528	WIKI:EN:CNN
104239528	WIKI:EN:World_Food_Programme
104239528	WIKI:EN:United_Nations_High_Commissioner_for_Refugees
104239528	WIKI:EN:Amnesty_International
104239528	WIKI:EN:Human_Rights_Watch
104239528	WIKI:EN:Reuters
104239528	WIKI:EN:WikiLeaks
104239528	WIKI:EN:Sadiq_Khan
104239528	WIKI:EN:Subsonica
104239528	WIKI:EN:Pope_Francis
104239528	WIKI:EN:Edward_Snowden
104239528	WIKI:EN:Mark_Ruffalo
104239528	WIKI:EN:Donna_Moderna

A set of the items liked by user2: we can see that he appreciates a lot of Italian items, but also a lot of international items.

Now we'll see the results of the recommendation systems for these four users. Remember that the task is to recommend 3 of 6 given items for each user.

HOW TO READ THE RESULTS TABLES

The recommendation methods we previously explained are used with the same configurations that we used in the clustering phase. We'll continue to call them *config1*, *config2* and *config3*.

For each configuration we have considered:

- The result of the user-item recommendation system previously described in this chapter
- The result of the item-based recommendation system previously described in this chapter, considering that the systems recommends:
    - The items that have the minimum of the maximum distances
    - The items that have the minimum of the mean distances
    - The items that have the minimum of the minimum distances

The following tables give the scores that each method assigns to the different items for each configuration. The three blue items of each row are the ones recommended to the user by the corresponding method.

RESULTS FOR USER 1 (1057516135)

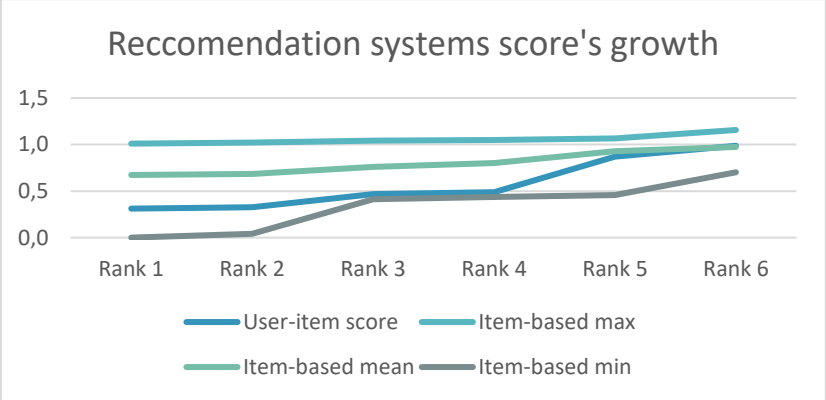
		Susie Cagle	Kenny Wayne Shepherd	Sean St Ledger	Red Hot Chili Peppers	The Beatles	Boyce Avenue
CONFIG1	User-item score	0.17039000	0.42281508	0.55811148	0.79953247	0.80423969	0.98632597
	Item-based max	1.02227079	1.04455196	1.00550186	1.02582943	1.08477604	1.05253601
	Item-based mean	0.93184477	0.93384039	0.95738822	0.77144658	0.77143234	0.99759137
	Item-based min	0.39566808	0.50040501	0.67964434	0.32947206	0.22832912	0.86714977
CONFIG2	User-item score	0.47340416	0.64141923	0.71653068	0.49541503	0.43813228	0.97936707
	Item-based max	1.01227343	1.05486238	1.00701546	1.00465250	1.10733079	1.05692613
	Item-based mean	0.93390357	0.94430005	0.95792269	0.78188002	0.76120674	0.99560552
	Item-based min	0.40223854	0.53543806	0.67957991	0.35301893	0.23438137	0.84882944
CONFIG3	User-item score	0.41939151	0.39414393	0.64881563	0.33956754	0.33316922	1.00717747
	Item-based max	1.00987470	1.0809626	1.16147994	1.02257919	1.06451034	1.24705362
	Item-based mean	0.90722399	0.88051331	0.94964432	0.62747299	0.63524425	1.03683924
	Item-based min	0.25452697	0.22963517	0.57523727	0.17631912	0.05439776	0.81202471

The first positive result of our recommendation system is that the great part of the configurations recommends Red Hot Chilli Pepper and The Beatles. These surely are the most coherent items for user1, considering that he's passionate about rock music. Sean St Ledger is a football player so doesn't have to be recommended to user1 and Boyce Avenue is a band that doesn't meet user1's tastes. It's interesting that user1 probably likes both Susie Cagle and Wayne Shepherd. Susie Cagle is a journalist and cartoonist and user1 is interested in animation (he likes Simon's Cat). Wayne Shepherd is a rock/blues guitarist and obviously user1 likes music. At a first look, the great part of the configurations suggests Susie Cagle to user1, so she seems more coherent, but we suspect that maybe Wayne Shepherd is better. Remember that config3 is the one which consider a characteristic's representation with dimensionality = 100, and that in the clusterization phase it led to a better result. We can suppose that also in this case it can be better and it is interesting to see that it modifies the results here too.

It's interesting to note the effects of the different strategies in the item-based approach. The *min* distance tries to recommend an item such that the user likes at least one item very similar to it. Meanwhile the *mean* distance gives a

more comprehensive view, recommending items that, in average, are similar to the ones that the user likes. The *max* distance, instead, seems to be the least appropriate, since it tries to suggest the item that doesn't yield a big distance with respect to any of the items that he likes.

However, these strategies lead to a very different distribution of scores:



Note that this chart is based on the user1 results table's config1 rows but represents a general pattern.

Finally, we can say that in general the user-item score seems to slightly favour items that represent people. This is explainable thank to the fact that we generate the representation of users using the knowledge derived from the Twitter graph, so it heavily involves the information of friends. Instead to generate the representation of the items we do not explicitly use the friends' information, we use only the already fit model to reduce the dimensionality.

RESULTS FOR USER 2 (104239528)

		Luciano Ligabue	Peter Diamandis	Sabina Guzzanti	Jonathan Meeks	Ivan Basso	Lovato
CONFIG1	User-item score	0.21562594	0.23696577	0.25874429	0.32971787	0.39319777	0.99606978
	Item-based max	1.03166687	1.046202182	1.10180389	1.01299917	1.04970848	1.01704287
	Item-based mean	0.65086758	0.66881108	0.66881841	0.70861601	0.72243505	0.99946391
	Item-based min	0.00240564	0.05709582	0.11523568	0.025071084	0.13041406	0.98465245
CONFIG2	User-item score	0.22302818	0.25074499	0.24854296	0.34684580	0.38999408	0.99827867
	Item-based max	1.05057406	1.03537511	1.06715750	1.01425302	1.039211273	1.01470232
	Item-based mean	0.63441950	0.66184371	0.65402621	0.70127230	0.71086639	0.99972659
	Item-based min	0.00159138	0.05304151	0.13264536	0.01960587	0.11038821	0.96493607
CONFIG3	User-item score	0.06860822	0.09963476	0.08521842	0.18547588	0.09613633	0.99904584
	Item-based max	1.02420628	1.03034806	1.02311146	1.04618513	1.01643586	1.05173861
	Item-based mean	0.47687402	0.50318753	0.49102938	0.55235600	0.49474266	0.99860179
	Item-based min	0.00019896	0.03451925	0.02566015	0.01010727	0.00268155	0.80939096

These are the results for the first user that probably is Italian. We already said that he likes also a lot of international items and concept and in the s22 file he has 200 preferences, from politics to entertainment figures. The result is that for him recommending items is a mess: he should like the Italian singer Luciano Ligabue (our only certainty), but he maybe likes Peter Diamandis (a famous entrepreneur), Sabina Guzzanti (an Italian politician and actress), Jonathan Meeks (an American Football player), Ivan Basso (an Italian cyclist) or Lovato (an Italian gas company). As we said for user1 probably the best results are the one of config3. We can confirm this theory because they are more coherent between strategies, than the ones of config1 and config2. However, let's see, in the next table, the difference with another Italian user that is focused on Italian items.

RESULTS FOR USER 3 (101532373)

		Francesco Facchinetti	Vanessa Mesquita	Micheal Franti	Julie Gonzalo	LCD Soundsystem	Michelin Guide
CONFIG1	User-item score	0.31219780	0.32624858	0.46797347	0.48979091	0.86958670	0.98736423
	Item-based max	1.01071619	1.02065658	1.04122328	1.04917728	1.06530714	1.15731239
	Item-based mean	0.67365473	0.68513345	0.75941383	0.80149823	0.92900764	0.97462397
	Item-based min	0.00038087	0.04246902	0.41394716	0.45911628	0.43726527	0.70254635
CONFIG2	User-item score	0.31937891	0.32664591	0.48155611	0.53578507	0.84857785	0.98381215
	Item-based max	1.01307153	1.03077936	1.06059670	1.03591990	1.05304896	1.10467374
	Item-based mean	0.67262995	0.67713385	0.75889861	0.79583966	0.93628478	0.97442007
	Item-based min	0.00025969	0.03669416	0.41654574	0.43427741	0.54710268	0.63797879
CONFIG3	User-item score	0.12710583	0.15415841	0.31326764	0.47501844	0.78134959	0.95927464
	Item-based max	1.03802096	1.07080721	1.07253897	1.04766476	1.04610264	1.08729732
	Item-based mean	0.49164122	0.51100057	0.61691224	0.71829754	0.87142097	0.91518914
	Item-based min	1.80602073e-05	0.01278769	0.30587744	0.41679912	0.27778279	0.22437548

In this table we can immediately see the difference with the previous one. Indeed, user3 likes only about 100 items and focuses on Italian and musical ones. This is surely the reason to recommend Micheal Franti (an American rapper) and Francesco Facchinetti (an Italian TV presenter). Vanessa Mesquita is a Brazilian Model and maybe she is suggested because there are common characteristics with user3's liked item: it's not so clear at a first look, it's difficult to understand why this item has been recommended but surely latent connections are present. It seems correct to us to not suggest the Argentine-American actress Julie Gonzalo, the LCD Soundsystem rock band (this user likes music, but he's focused on pop genre) and the Michelin Guide, because this user doesn't like anything related to food.

RESULTS FOR USER 4 (101935414)

		Peter Barakan	Gilles Peterson	Shinici Osawa	Kyle Lohse	Bonnie Bernstein	Kevin Davies
CONFIG1	User-item score	0.18335354	0.28289526	0.31770640	0.44326877	0.55662775	0.62633281
	Item-based max	1.02904319	1.00326359	1.00247037	1.05889630	1.02348864	1.00541985
	Item-based mean	0.68510347	0.72126948	0.73663341	0.77673810	0.82293516	0.86050999
	Item-based min	0.12079596	0.26013034	0.29372304	0.33003520	0.34226554	0.62006115
CONFIG2	User-item score	0.20544975	0.33083713	0.36038476	0.46614426	0.56329762	0.65747344
	Item-based max	1.03652811	1.01947152	1.01102817	1.07803082	1.00260615	1.00794053
	Item-based mean	0.67104762	0.72105151	0.73521077	0.77549606	0.81082469	0.86116474
	Item-based min	0.09001410	0.25340670	0.28627026	0.32753765	0.32849371	0.61797916
CONFIG3	User-item score	0.12612426	0.11803799	0.11837840	0.14473962	0.29156279	0.59117245
	Item-based max	1.01171267	1.00667226	1.00169003	1.00861442	1.04054832	1.01898217
	Item-based mean	0.56093895	0.55169630	0.55066066	0.57054179	0.64652693	0.80114173
	Item-based min	0.01693272	0.01173305	0.02248156	0.036414623	0.13861727	0.53733444

We can't recognize exactly what user3 likes, but most of his preferences are Japan-related. So, it is good that Shinici Osawa (a Japanese musician) is the only item that is always recommended. Another positive thing is that, as for user1 and user3, the items to recommend seems clearly Peter Barakan (an English-born Japanese DJ) and Gilles Peterson (another DJ). So we can even understand that this user probably likes electronic music (all his recommended items are

DJs) and it is curious to note how much his results are focused around a topic considering that he has only 27 preferences. Kyle Lohse (a football player), Bonnie Bernstein (a sport journalist) and Kevin Devies (a football player) are not suggested. So, it seems to work in the correct way.

CONCLUSIONS

The performances are encouraging, and we think that with a more extensive parameter tuning and some tweaks to the procedure they could be even better.

The recommendation system is particularly good, since this system can recommend whichever object expressed in terms of BabelNet categories, yielding good performances. We used two different approaches with slightly different goals, while the item-based approach is more suitable to recommend items, the user-item approach is more suitable to suggest other users.

The possibility to recommend generic objects without directly knowing the preferences of a given user seems to be achievable using the user-item approach, since the user preferences are automatically inferred from the other information available.

FUTURE WORKS

It may be interesting to deepen some aspects of the work. The first thing that should be done, with the availability of a more powerful machine and more time, is to perform a deeper tuning of the parameters that couldn't be done due to physical constraints. In particular, it may be interesting to increase the maximum friend distance that the information can travel. Now it is limited to two, maybe with bigger values the performances may increase.

It could be interesting to generate the representation of the Wikipedia pages in a smarter way, at the moment it contains only the page's categories: as with the users, we could embed in the representation the categories of *similar* pages.

Finally, the method that we used to compute the embeddings could be compared to a PageRank-like algorithm, that may perform better, i.e. iteratively update the embeddings representation in function of other embeddings, until convergence.

RIFERIMENTI

[1] «Java,» [Online]. Available: https://www.java.com/it/download/.

[2] «Python,» [Online]. Available: https://www.python.org/.

[3] Twitter4j. [Online]. Available: http://twitter4j.org/en/index.html.

[4] «BabelNet,» [Online]. Available: http://live.babelnet.org/about.

[5] «Babelfy,» [Online]. Available: http://babelfy.org/about.

[6] «Scikit-learn,» [Online]. Available: https://scikit-learn.org/stable/.

[7] «SciPy,» [Online]. Available: https://www.scipy.org/.

[8] «NumPy,» [Online]. Available: http://www.numpy.org/.

[9] fastutils. [Online]. Available: http://fastutil.di.unimi.it/.

[10] «TruncatedSVD,» [Online]. Available: https://scikit-learn.org/stable/modules/decomposition.html#lsa.

[11] «MiniBatch K-Means,» [Online]. Available: https://scikit-learn.org/stable/modules/clustering.html#mini-batch-kmeans.

[12] «Calinski-Harabaz,» [Online]. Available: https://scikit-learn.org/stable/modules/clustering.html#calinski-harabaz-index.

[13] «Davies-Bouldin,» [Online]. Available: https://scikit-learn.org/stable/modules/clustering.html#davies-bouldin-index.