**WEB AND SOCIAL INFORMATION EXTRACTION PROJECT Report**

lUCA mOSCHELLA, 1594551 AND Federica Spini, 1588482

Summary

[Introduction 3](#_Toc531191593)

[Technologies 3](#_Toc531191594)

[Dataset handling 3](#_Toc531191595)

[Modelling 3](#_Toc531191596)

[Acquisition 4](#_Toc531191597)

[Analytics 4](#_Toc531191598)

[Semantic processing 5](#_Toc531191599)

[Babelnet 5](#_Toc531191600)

[Babelfy 5](#_Toc531191601)

[Extraction of categories and domains 6](#_Toc531191602)

[WikiMID 6](#_Toc531191603)

[Tweet contents 6](#_Toc531191604)

[Items 6](#_Toc531191605)

[Limitations 6](#_Toc531191606)

[Twitter Information Extraction (task 4) 7](#_Toc531191607)

[What we downloaded with Twitter4j 7](#_Toc531191608)

[How we extracted the features 7](#_Toc531191609)

[Representation with latent categories 7](#_Toc531191610)

[Wikipedia pages 7](#_Toc531191611)

[Tweets 8](#_Toc531191612)

[Users (task 1) 8](#_Toc531191613)

[Why this formula? 9](#_Toc531191614)

[Curse of dimensionality 9](#_Toc531191615)

[Notes about the efficiency 9](#_Toc531191616)

[Adjacency matrix exponentiation 10](#_Toc531191617)

[Clusterization (task 2, 4) 10](#_Toc531191618)

[Mini Batch K-Means (inserisci riferimento) 10](#_Toc531191619)

[Clusterization Parameters 11](#_Toc531191620)

[Evaluation (task 3) 12](#_Toc531191621)

[Calinski-Harabaz Index (inserisci reference) 12](#_Toc531191622)

[Davies-Bouldin Index 12](#_Toc531191623)

[Results 13](#_Toc531191624)

[Change in C 16](#_Toc531191625)

[Change in gamma 17](#_Toc531191626)

[Random forest 17](#_Toc531191627)

[Change in max\_depth 17](#_Toc531191628)

[Change in n\_estimators 17](#_Toc531191629)

[Conclusions 18](#_Toc531191630)

# Introduction

DOBBIAMO FINIRLA DI SCRIVERE SUL GOOGLE DOC, POI COPIAMO E INCOLLIAMO

## Technologies

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

* **Java**: to read the datasets, merge the information from the different sources, and generate a unique dataset
* **Python**: to perform algebraic computations on sparse data, cluster and recommend items.
* **Twitter4j**: to download information from Twitter
* **Babelnet**: to find the categories and the domains associated to a given Wikipedia page
* **Babelfy**: to disambiguate the text in the tweets
* **Scikit-learn**: to normalize, reduce the dimensionality, cluster and evaluate the clusters
* **Scipy**/**numpy**: to represent sparse matrices and perform vectorized computations
* **fastutils**: 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.

INSERISCI IMMAGINE

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.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | 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 |

INSERISCI ESEMPIO DI CARATTERE NON UTF8 IN S2\*

# Semantic processing

Scrivi introduzione

## Babelnet

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. inserisci reference

## Babelfy

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). inserisci reference

## 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](http://en.wikipedia.org/wiki/Category:1940_births)“, “[English rock singers](http://en.wikipedia.org/wiki/Category:English_rock_singers)” and “[Murdered male actors](http://en.wikipedia.org/wiki/Category: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 (INSERISCI REFERENCE DI BABELNET). 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 created 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 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,   means that the user has times the category in his tweets.
* Let be the personal page matrix. It contains a row for each user and a column for each existing category. So, means that the user has times the category 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.
* Let be the liked item matrix. It contains a row for each user and a column for each existing category. So that   means that the user has times the category in the Wikipedia pages he likes.
* Let 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,  says if the user follows or not the user .

Now we can define as a matrix that contains a row for each user and a column for each existing category, each row will be the vector representation of the user. A row represents the categories associated to all the various objects related to a user: so   means that the characteristic has importance for the user .

SUPERMAXIFORMULONA

Where:

* alpha is the weight (or the importance) given to the categories of the tweets
* beta is the weight (or the importance) given to the categories of the personal Wikipedia pages
* gamma is the weight (or the importance) given to the categories of the liked Wikipedia pages
* Capital delta iis the weight (or the importance) given to the categories of the following. Note that Capital delta depends on the friendship level. SPIEGARE LA COSA DELL’ESPONENTE: NON LO SCRIVO QUI PERCHE’ C’E’ BISOGNODI SCRIVERE LE FORMULE SPECIFICARE CHE: quelle dei following avranno alpha', beta' e gamma'

### Why this formula?

Spiegazione da scrivere ancora

## 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 \* 190k matrix, we reduced its dimensionality to 500k \* 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 (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.

ALGORITMO PER ABBASSARE LA DIMENSIONALITA’ DEI DATI, Spiega come funziona TruncatedSVD …? non so se serve, forse un copia incolla dell’abstract della documentazione di sklearn

## Notes about the efficiency

We are dealing with big matrices. The adjacency matrix has dimension 500k\*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\*190k, each one of them would require 353 GB 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 where the row represents the 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 exactly 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 (inserisci riferimento)

The MiniBatchKMeans is a variant of the KMeans 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, similar to 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.

MiniBatchKMeans converges faster than KMeans, 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.

>>>>inserisci immagine da doc sklearn

### Clusterization Parameters

The following table describes the various parameters that we have to 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, s MiniBatchKMeans and the table contains also the hypermarameter 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 contain 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 , where at distance I the weight is |
| 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 characteristics 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 (inserisci reference)

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:

>>> formula

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.

- ref alla documentazione dei due

### Davies-Bouldin Index

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 Ci for i = 1..k and its most similar one Cj. In the context of this index, similarity is defined as a measure Rij that trades off:

* si, the average distance between each point of cluster i and the centroid of that cluster, also known as cluster diameter
* dij, the distance between cluster centroids i and j

A simple choice to construct Rij so that it is nonnegative and symmetric is:

>>> formula

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.

#### Calinsky

|  |  |  |  |
| --- | --- | --- | --- |
| **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

|  |  |  |  |
| --- | --- | --- | --- |
| **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 |

# Raccomandation system (task 5)

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 raccomandation 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: the distance of page i >>>from m is defined as the min/mean/max distance of the distances (i, j) for every page j that he likes.

>>>> formula

## 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 similar to 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 those information, contained in S22!

Representing each object in function of its categories makes possible to obtain a general-purpose recommendation system that is able to 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\*m) while the user-item approach takes O(m).

## Results

Analyse the recommendation system is more difficult than evaluate the clusterization system. The reason is that now we don't have 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 of all, we have looked for users with preferences well defined. It was not easy, because the most of our users like a lot of items that in general are really famous. Because we don't know a lot of items and too famous items are not very significative (Rihanna's page appear in 92



User in S22 with his preferences

We can note that the represented user likes rock and metal bands and some videogames inherent item. Let's see what our various tests' results recommend to him:

Altri user interessanti sono: 101532373 (gli piace una marea di roba, ma un sacco di cose sono italiane). A 104239528 piacciono cose italiane, di cui molte inerenti a politica e attualità. 101935414 apprezza roba giapponese.

### Change in C

For each type of kernel, I've tried a c value among 0.001, 0.01, 0.1, 1(default), 10 and 100. Those are the tables that show how the value changes when C changes:

From those graphs maybe, it is clear that the better kernel is the 'rbf'. This is not completely true, because the training of 'poly' and 'linear' SVM employs a very large amount of time. For this reason, I stopped the training after one million of iteration and probably it is the cause of their terrible results. So, these results are not "so real": with my computer the 'linear' kernel employs about 5 minutes with one million of iteration obtaining the showed results. Always the linear kernel, with a c value of 0,1 employs 25 minutes to train without any bound of time or of number of iterations. Tre result is that the average precision is 0,84 instead of 0,38 , the average recall is 0,84 instead of 0,23 , and the average of f-measure value is 0,83 instead of 0,14. For the 'poly' kernel I've trained it without a bound with the minimum degree (set at 2, because if it would be 1 it is linear) for more than two hours and it didn't converge, so probably it is completely unsuitable to the considered data.

Apart from unreliable results of 'poly' and 'linear' kernels, the important matter here is making a couple of consideration about the value C: in the 'rbf' kernel the better one is the default (c=1) that is a "middle way" c-value. For the sigmoid it seems better to use a little value, and this probably happen because with a large c the model wants "very tight" bounds; this is a problem because this favours the overfitting. Considering that I've **chosen c=1 with 'rbf' kernel**, **c=0,001 with 'linear'** and **c=0,01 with 'sigmoid'**. I didn't use poly anymore because foe the dataset dimension is too slow.

### Change in gamma

For each type of kernel I've tried a c value among (1e2, 1, 1e-2, 1e-3, 1e-4, 1e-5). The obtained results show that:

* For the 'rbf' kernel used with c=1 the variation of gamma value doesn't make any change in the result, unless it is a value minor than 0,001: in this case the smaller is the value, the worst are the performances. Precisely, with gamma= 0,0001 all the evaluation measure get worse by 1% and with gamma=0,00001 they get worse by 2%.
* Both for the 'linear' kernel and for the 'sigmoid' kernel there isn't any variation with the change of gamma value

## Random forest

The random forest is composed by decision trees, so some of the parameter I've considered are the same. Precisely I've tried at first to understand which type of the preprocessing is the better one, then I've tuned, as for the decision tree, the **max\_depth** of the trees and the **class\_weight**.

Moreover, this time there are multiples decision trees so another important hyperparameter to consider is **n\_estimators**, that is how many decision tree do we use to build the model.

The results are in the following tables and each type of preprocessing parameter it's highlighted the better result in green.

### Change in max\_depth

Variation of the values with max\_depth = {5, 10, 15, 30, 45, 60}. I've used here the dataset completely preprocessed with exception of capital-gain and capital-loss features. Note that the better value is 10, that allows to reach the better result until now. Using a **class\_weight** value ={0:3, 1:7} almost all the performance measures get worse about of 1%. Using a complete preprocessing procedure all the values get worse about of the 2-3% and the better value for max\_depth is 15. Without any preprocessing performance are almost like with the complete preprocessing and the better value for max\_depth is 10. It is logica if we think about it, that the better value of max\_depth in the random forests coincides with the optimum for the decision tree, because a random forest is composed by different decision trees.

### Change in n\_estimators

Variation of the values with n\_estimators = {2, 6, 10, 15, 20, 25, 40, 100}: I've tried all those values on the better result of the previous section, but both precision, recall and f-measures still among 84-86%. This probably is due to the fact that the decision trees are all similar one to the other.

# Conclusions

This project has been an opportuny for me to study and use different Machine Learning algorithm looking for their different effects. Sometime is happened the opposite respect to what I expected, but the useful thing is that when there is something strange I've to explain why it happened. The results are however quite good because the dataset respect some of the feature is maybe not big enough. Capital-in and capital-gain are, in my opinion, two feature that are difficult to deal with. It has also been important to use an instance if it has a missing value: at the very beginning I discarded them, but results were strongly worse.

Of all the attempt I've described the better one surely is the random forest used with max\_depth=10: I'm satisfied of the precision and recall values I've obtained, which are 0,86 and 0,87. The goodness of this result however is mostly due to the preprocessing phase that have improved the results largely. The tuning of the various hyperparameters allowed to improve only a few the quality of the model.