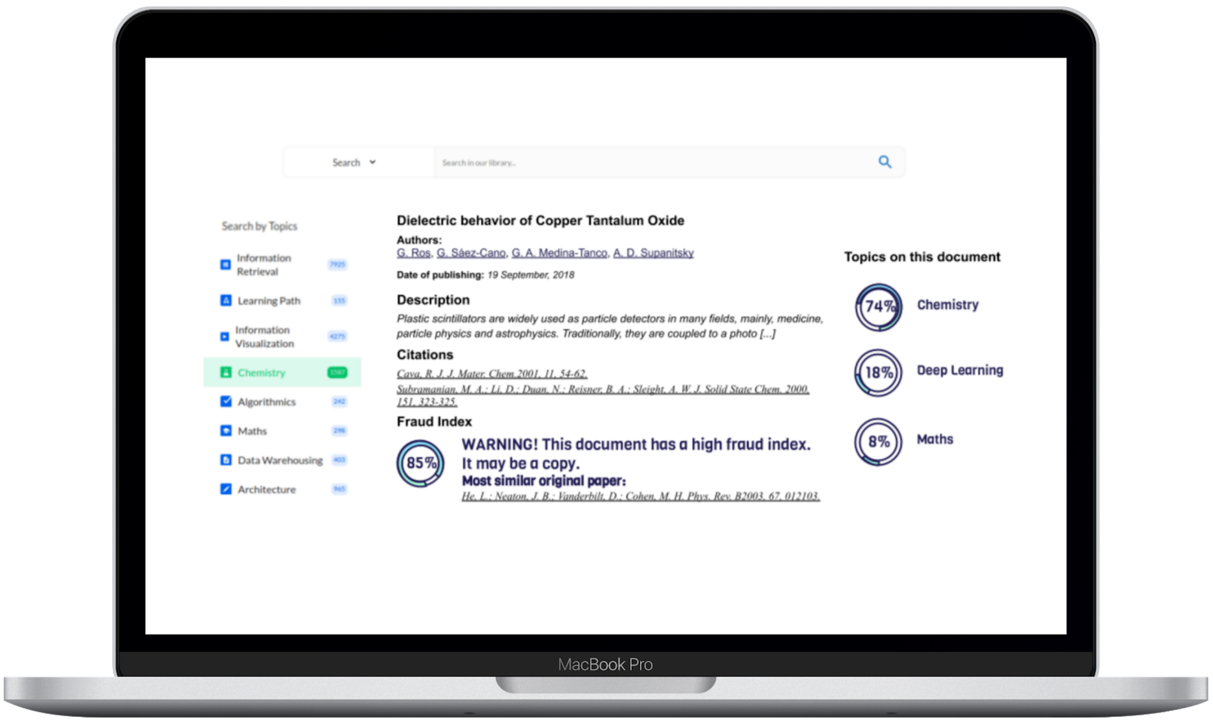
Information Retrieval and Analysis
December 2020

Lab Session 8: System design

PROJECT REPORT



Dielectric behavior of Copper Tantalum Oxide

Authors:
G. Ros, G. Sáez-Cano, G. A. Medina-Tanco, A. D. Supanitsky

Date of publishing: *19 September, 2018*

Description

Plastic scintillators are widely used as particle detectors in many fields, mainly, medicine, particle physics and astrophysics. Traditionally, they are coupled to a photo [...]

Citations

Cava, R. J. J. Mater. Chem.2001, 11, 54-62.

Subramanian, M. A.; Li, D.; Duan, N.; Reisner, B. A.; Sleight, A. W. J. Solid State Chem. 2000, 151, 323-325.

Fraud Index

WARNING! This document has a high fraud index.
It may be a copy.
Most similar original paper:
He, L.; Neaton, J. B.; Vanderbilt, D.; Cohen, M. H. Phys. Rev. B2003, 67, 012103.

Elías Abad Rocamora
Victor Novelle Moriano
Barcelona, UPC - FIB & FME

Introduction

Scientists' prestige nowadays is given by the number of citations and relevance their papers get in the scientific community. Having a high prestige is very important for them as the future work they get, salaries and students that want to be taught by them, depend on it. But in order to get prestige, some authors cheat to increase the number of citations their papers have using different techniques. This can then lead to not so good researchers achieving better positions and opportunities than others who deserve it more.

Another problem that we encounter in the research world is that some relevant papers in a research topic remain unnoticed and don't get as much credit (citations) as other irrelevant ones. This can be because the author or the topic covered in that paper is not too popular, but if a topic becomes popular in the future, we would like to find that paper that was written 50 years ago and covers what I'm studying now.

In order to solve these and any other problems appearing to researchers, we think that an Information retrieval system that analyzes the information available about authors, documents and citations needs to be implemented. We would like to have a system that truly estimates the relevance of a researcher's work without taking into account (or even penalizing) citation boosting. Knowledge discovery is a very important task, relevant information about a topic has to be easily available to apply it in future research, as repeating discoveries should be avoided to better invest researcher's precious time.

In this project, we will define the structure of a system that provides insightful information for our scientists to help them perform better in their work and truly represent their effort and merits.

Functionalities

Taking into account all the mentioned above, we have designed a search system that allows fulfilling the later tasks easily for the user and greatly improves the obtained results/ time spent ratio on his research.

One basic functionality of our search engine consists of retrieving the R most relevant documents for a given topic ($R \in N$). In this way, the user would only need to input the name of the topic that he's interested about and the best document would pop out without performing a time-consuming search. Also, the user could introduce topic-related terms if he wants to consult a branch of a bigger section (*Deep Learning* $\subset$ *Machine Learning e.g).* To perform such a task, the Arxiv database would be consulted.

We also think that another interesting functionality regarding document search would consist of finding similar documents to one given by the user. In this way, the user could extract ideas from papers with a similar approach as the users' document. Moreover, instead of just returning alike papers, the search engine would also provide them ranked, thereby the user can consult the ones that were well appraised among the research community. To perform this activity, the Arxiv database would also be used.
Independently of the type of search performed, each retrieved document would be provided to the user with the same layout.

Document layout:

- **Title,author and date of publishing**
  Essential for document identification.
- **Abstract**
  Allows identifying with a glance if the document is useful for our purposes.
- **Topics**
  List with the topics present on the paper and the distribution of them. Helps in identifying the weight of the different subjects in the document, as well as allows you to perform a topic-based search clicking on the wanted thematic.
- **Citations**
  Allows easy navigation to documents used in the work.
- **Fraud index**
  Provides a value between [0,1] range that indicates the probability that the document is a copy from another paper previously published. If this value is above a threshold, a link to the paper with higher similarity would be provided. Doing this, the user could check by itself the differences by itself and choose properly which document should he study/cite for his work.

As was mentioned in the introduction, citations have a huge impact on authors' prestige, so we think that diverse functionalities regarding this aspect need to be implemented.

A user could be interested in obtaining the ranking of authors in relation to the topic he is working into to further investigate their work and acquire a deeper knowledge. So, an author search functionality should be provided, that introducing a topic by the user, returns the most prestigious scientists that have published about it, and when selecting one, a profile similar to the one Google Scholar provides would be displayed. A “direct search” functionality for authors should also be provided, in order to ease this consult if the user wants to perform it.

This profile would include the most relevant documents of this author, but also some insightful statistics about this author, as:

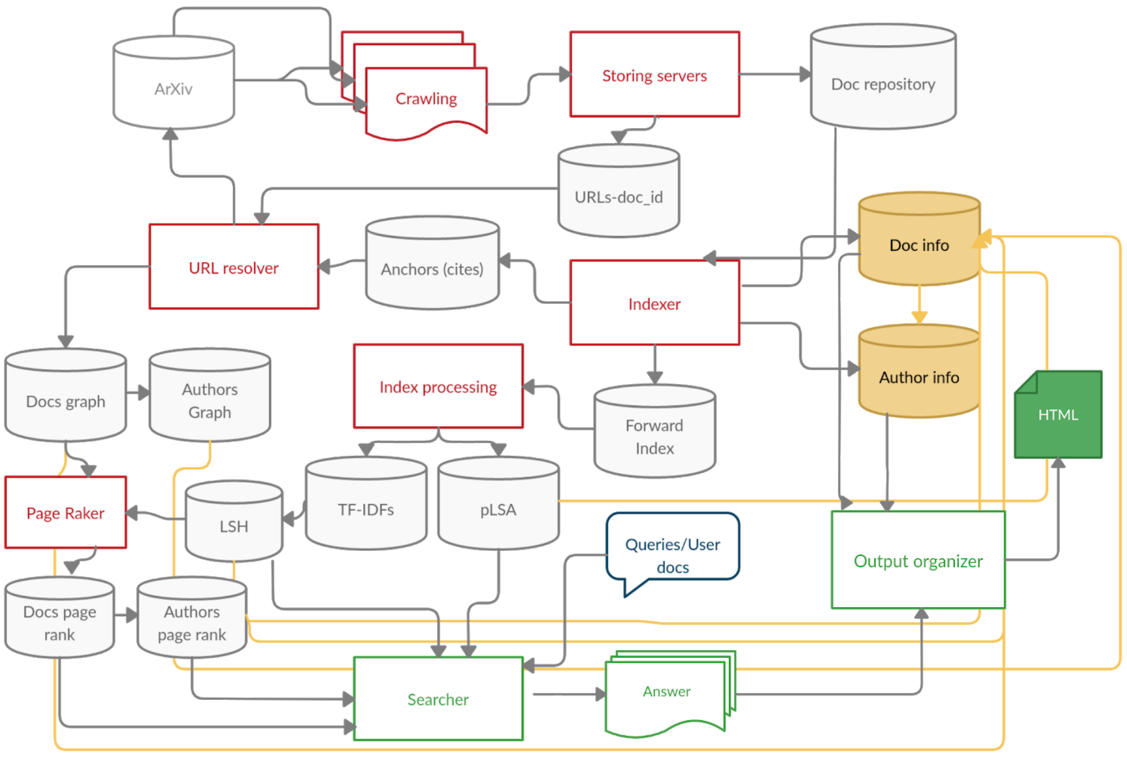
Author layout:

- **Authors he cites the most and the ones that cite him**
  A list of 5-10 authors with the number of cites to them. Another list with authors and the number of times they cite the author in this profile.
  This can help the user detect if this author has agreed to mutual cite with another one.
- **Total number of citations.**
- **Median number of citations**
  Taking the median value of citations and the total number of them can help the user see it this author has only a few important papers and many others that are irrelevant.
- **Topics that he works into.**
  As in documents’ topics, a link to search documents or authors in that topic is provided.

- **Fraud index**
  This fraud index will be a combination of the mean fraud index of all his/her papers, the number self-citations and mutual citations within a group of authors. Moreover, besides just providing this metric, a list with the values taken into account into its computation will be provided. Doing this, the user can find why an author is considered fraudulent instead of just having to assume so based on the number.
  - **Percentage of self-citations**
  - **Percentage of suspicious of being agreed citations**
    Both with respect to the total number of citations he/she gets
  - **Mean fraud index across all his/her papers**

High-level architecture

Now we will focus on the architecture our browser will have internally in order to work properly.



Technical implementation

Having defined the high level architecture of our system, we need to define specifically how everything will work and which techniques will be used for each part.

Crawling

In this process, the Arxiv database is systematically explored and its documents are collected. In order to do so, an URL for each Arxiv topic is established in the “*URL seeds queue”.* Doing this, a depth-first based exploration can be performed for each one of the areas present on the database. As a general rule, the documents that are classified into a certain topic by Arxiv will have connections to documents that are related to the same ambit.

As crawling is a distributed process, approximately all the themes will be equally explored in any moment. This fact could not be achieved by a random initialization of the seed, as we could have the bad luck to select several documents related to the same topic and exploring its theme to a greater extent than the rest.
Indicate that this process should be done following the Arxiv crawling policies and respecting the standard for robot exclusion guidelines. To do so, Arxiv has an up-to-date copy of the corpus oriented to perform crawling tasks, which will be used to perform a monthly extraction without impairing its server capacity.

Indexing

Once the Crawler has returned the set of documents we can proceed to do the **Indexing**. The task of the Indexer will be to parse the text to separate it between the text itself, title, authors, citations, publishing date and abstract. Then, each part of it will be sent to a different mechanism.

After segmenting a document, we will incrementally create, based on cites and authors of a document, graphs between authors and between documents.

The graph for authors will have:
- **V**: a set of vertices (a_i), with one vertex for each author.
- **E**: a set of directed edges ($e_{i,j}$) between vertices. The existence of an edge will be determined by the existence of a cite from one author's document to another author's document.
- **W**: a set of weights ($w_{i,j} \in N$) for each edge representing the total number of citations of an author to another.

The documents graph will be composed of:
- **V**: a set of vertices (v_i), with one vertex for each document.
- **E**: a set of directed edges ($e_{i,j}$) between vertices. The existence of an edge will be determined by the existence of a cite from one document to another one.

Both graphs will be implemented as adjacency lists, the first one containing a list of pairs (doc_id, weight), and the second one containing only the doc_id. Indicate that, differently to the documents graph, the lists store ingoing edges.

Once all the documents have been retrieved, which means that the graphs have been created, we can compute some trustworthy metrics about the authors. The % of self-citations is easily computed dividing the self-citations between the sum of the total citations that author gets.
In relation to the % of suspicious of being agreed citations, it is difficult to establish an optimal decision boundary that determines when the citations are fraudulent or not without testing on real values. We consider that a possible option could be establishing a threshold for the number of mutual citations,considering all the ones above the limit invalid. In this way, we prevent the creation of fraudulent agreements.
This implementation could be easily added also using the authors' graph. Exploring the list for a given author we identify the number of cites that surpass the mentioned threshold. Next, we check if the author also cites him back above the threshold. If this is the case, both

number of cites will be invalidated. Finally, to calculate the percentage a division between the number of invalidated cites and the number of total cites is performed for each author. These metrics will be stored in the author's database to later generate their profiles.

Concurrently to the graph creation, the Indexer will also take care of the different sections it has separated above. On one hand, with the text, the forward index is created, generating the doc_id-term frequency matrix. On the other hand, the title, authors, citations, publishing date and abstract will be stored in the documents information database. In this database, the basic information for each document is stored and, when a document wants to be consulted, the formatter will generate its profile. Notate that the missing metrics in the document profile cannot be computed at the moment and they will be added later on in the process. Another database for storing basic information about authors will be also needed.

Having the forward indexes we can proceed to compute the *TF-IDF* representations of our documents in order to solve queries and compute similarities. This will be done in parallel with the topic modelling of our corpus.

Topic modelling

The type of topic model we have chosen for our system is *pLSA. pLSA* is based on the doc_id-term frequency matrix, by means of a log-likelihood maximization criteria, we decompose this matrix in three matrices:

$$A_{DxT} \approx U_{DxK} \cdot S_{KxK} \cdot V^T{}_{KxT}$$

Where A is the original doc_id-term matrix, U is the matrix representing the probability of a document given a topic $(U = [P(d_i|z_j)]_{(i,j)\in DxK})$, S is a diagonal matrix containing the prior probability of each topic $(S = diag([P(z_i)]_{i=1,\dots,K}))$ and V^T is the matrix representing the probability of a certain term given a topic $(V^T = [P(w_i|z_j)]_{(i,j)\in KxT})$.
In order to compute this representation, we need to define the hyperparameter K, which is the number of topics we want. The choice of this parameter has to be thoughtfully studied, one approach could be to take only the topics for which the prior probability is big enough. Also, a naive choice of K could be the number of research fields in Arxiv.

The assignment of the topics obtained with *pLSA* to real research topics (e.g. Physics, Computer Science, etc.) will be done by getting the terms with the highest probability for each row in V^T and interpreting which area of research could contain that set of words. For example, if the five terms with the highest probability for a certain topic are *computer, algorithm, loop, execution and Dijsktra,* that topic is likely to be Computer Science.
We've chosen pLSA as it is easy to interpret and gives us a probabilistic representation that we can use to compute which topics appear in a certain document and which ones does an author work in $(P(z|d)\ and\ P(z|a) = \frac{1}{|D_a|} \cdot \sum_{d\in a} P(z|d))$ to then insert this data in the Authors and Documents databases.

Topic-based page ranks

As we already know, one of the main problems of traditional page rank is that it is insensitive to the user's needs. To avoid that, we have opted to perform a topic-based page rank. Doing this, the advantages of traditional page rank are maintained, off-line computation and collective reputation, and several optimizations can be performed if a topic-related search is done by the user.

To compute the page rank, the power method for the Google matrix will be applied for each topic on our corpus. This could be performed using the following pseudocode:

```
$P_d$ = dictionary of the pageranks per topic for every document ({d:[$P_{d,k}$]})
for all topics k:
         $G_k$ = Graph containing only the documents with topic k.
         $n_k$ = len($G_k$ ).
         $P_k$ = dict([(v,1/n) for v in $G_k$])
         d = the chosen damping factor, between 0 and 1
         dist = 1 #default, only to enter the loop the first time
         while not dist <= epsilon:
                   $P_k$ new = dict([(v,(1-d)/n) for v in g])
                   for v in $G_k$:
                        L = g[v] # forward adjacency list for node v
                        for j in L:
                             $P_k$ new [j] += d * $P_k$[v]/len(L)
                   #Function that computes euclidean distance between dicts
                   dist = distance($P_k$,$P_k$ new)
                   $P_k$ = $P_k$ new
         for doc,prk in $P_k$.items():
                   $P_d$[doc].append(prk)
```

As it can be seen, this implementation is computationally efficient mainly due to two reasons. On one hand, the G_k will be computed on the fly after checking once the topics for each document, using the U_{DxK} matrix on *pLSA.* On the other hand, a list of outgoing edges is used encoded in dictionaries, allowing us to load each row of the adjacency list separately and iterate through it efficiently.

In order to compute the page rank of the authors, the previously computed page rank for the document is used. To do so, the topic-page rank of a document is divided between all the authors that published it. Doing this, a logical redistribution has been performed so that the total sum remains 1. This operation must be applied for all topics to obtain the list of page ranks per author.

LSH

In order to check if there are documents that are too similar and therefore suspicious of being a copy, we are going to use Locality Sensitive Hashing. This technique will map every document to let's say, a number, taking into account the similarity of the TF-IDF representation of two documents to map them to a closer number, or even to the same one if they have very high similarity.

Based on the result of this mapping, we will iterate over every "bucket" and compute the cosine similarity between every document in the bucket and the ones that have a previous publishing date. It makes no sense to calculate the fraud index of a document with the ones

that were published after it, we suppose that the first one is the original. In order to make this efficiently, we will keep the items in the same bucket sorted by publishing date.
After computing the cosine similarities within the bucket, the fraud index of a document will be the mean value of them. When we have this value we can simply update the document information database and upload this fraud index.

After the computation of the documents' fraud index, we have access to all the necessary metrics to compute the authors' one. We have decided to use an "*inverted harmonic mean".* As we know, the F-measure penalizes more heavily small values than the arithmetic mean. If we apply:

$$1 - \frac{3}{\frac{1}{1-x} + \frac{1}{1-y} + \frac{1}{1-z}}$$

where x = % self-citations, y = % suspicious of being agreed citations and z = average fraud index of documents

Using this, if one of the three metrics has a high value, *1-metric* will have a low one, being strongly penalized by the mean. This effect will be later on inverted by the *1 - mean* computation, obtaining an elevated fraud index. We have decided to not give more weight to any specific metric as it would imply a minor penalization to the other ones, benefiting those who practise these ruses. This metric will also be stored in the authors' database.

Queries

Now that we have indexed, modelled the topics and computed the RageRanks, we can proceed to explain how our queries will be executed. In order to do that, we will define the steps involved in every query defined in the functionalities of our system:

- **Top R most relevant documents/authors per topic:**
  We can solve this query easily just by sorting the pages within a topic based on the Topic PageRank that we have precomputed and get the top R documents/authors. So there is very little online computation for this query and it will be fast to execute.

- **Top R most relevant documents by query:**
  In order to solve this query, we will take advantage of the V^T matrix of *pLSA.* In this matrix, we have the most relevant terms of a topic, so one could compute the cosine similarity of the query with every term vector of the topics and then ponderate the Topic Pagerank of every document by this similarity to get a score:

  $$Score(d_i) = (1 - f_i) \cdot \sum_{j=1}^{K} sim(V^T[j, :], q) \cdot P_{j,d}$$

  Where $P_{j,d}$ is the Pagerank for topic j and document d. And f_i the fraud index of document i. By multiplying by $(1 - f_i)$ we penalize the score of documents with a high fraud index.

  This query can also be resolved very efficiently, as we have already computed $P_{j,d}$ and we only need to compute $sim(V^T[j, :], q)$, which asymptotically takes $O(T)$ and as we have to do this per every topic and document, it will take $O(K \cdot T)$.

A query refinement, via for example Rocchio's rule, could also be done, but we think that the system could work effectively as it is, because if we take a query that is clearly characteristic of a topic, only the similarity with that topic will work well and all the documents that contain that topic will be benefited, so there is no need to add other relevant terms to the query. Also when we have a mix of topics in the query, the documents containing the topics in the query will also be benefited in terms of score.

After computing all the scores, we simply sort decreasingly by Score and return the top R documents.

- **Return documents related to another:**
  We can also solve this task easily by taking documents in close hash mappings to the initial document. A number of maximum "buckets" of distance could be defined and then we could output all the documents in buckets closer or equal to this distance. This is done efficiently as we have already computed LSH mappings for every document.
- **Search an author/document by name:**
  This task only involves a query to the author/document information database, where the name is like the one provided by the user.

For the first three queries, the output will be a list of documents or authors. Our idea is that some sort of abbreviated profile of each document/author will be displayed. For example, in the case of documents, we could display the document's name, the authors, the first 3 lines of the abstract and the topics it contains. In the case of authors, the name of the author, the research topics he/she writes about, the total number of citations and a picture could be displayed. Of course, after clicking on a document or an author, the full profile would be displayed.

Limitations

The limitations of this project are directly correlated with the volume of documents our system will work with. Depending on the number of documents and the average number of different terms per document, we could estimate the size needed for storing our indices, pageranks and topic models, but this is not a big problem as they usually can fit all into a single server or even in memory.
The problem comes when storing the documents themselves, that is the .pdf files. The memory needed to store the files scales very badly with the number of documents, for this reason, we will be needing to scale out and use more than one server to store our corpus. A Consistent Hashing could then be used to replicate the data into different servers.

This helps us in two ways:

- If a server fails, our data will be replicated in other servers where we can still get it.
- If a document becomes very important, we can distribute the load of users wanting to access that topic between all the servers that contain that document.

The drawbacks of this data distribution is that it makes the implementation of our system more complicated and expensive. We will need more money for servers and more time to design the connection between machines.

Regarding the response time,in the best case,which consists in having the original document (or a replica) in the server where the petition was made, it would be almost instantaneous. The time would be spent on the query solving, the generation of the authors/documents profiles and sending the response to the user by the network. This set of operations could be performed in the order of seconds.

However, in the worst case, such an optimistic forecast it’s not so credible. This scenario would happen when the document/s that form part of the answer are not located in the server where the request is being made. These papers should be fetched in other servers, that will also be dealing with other user petitions. This may lead to bottle necks and slower response times. As the average bandwidth ranges between 12 to 25 MBps and most of the *PDFs* occupy less than 10 Mb, we estimate around 1~5 added seconds in the response time.

The user satisfaction with the system could be measured in many different ways. We haven’t yet seen these techniques in this course, but we could keep track of the time a person spends on the results page of a query. If a user stays a long time looking at the results, this will probably mean that he/she was interested in what the system returned. Another metric could be the number of queries the user executes. If a user makes many requests, it might be because he/she likes the system, otherwise, he/she would leave our page and look for another solution.

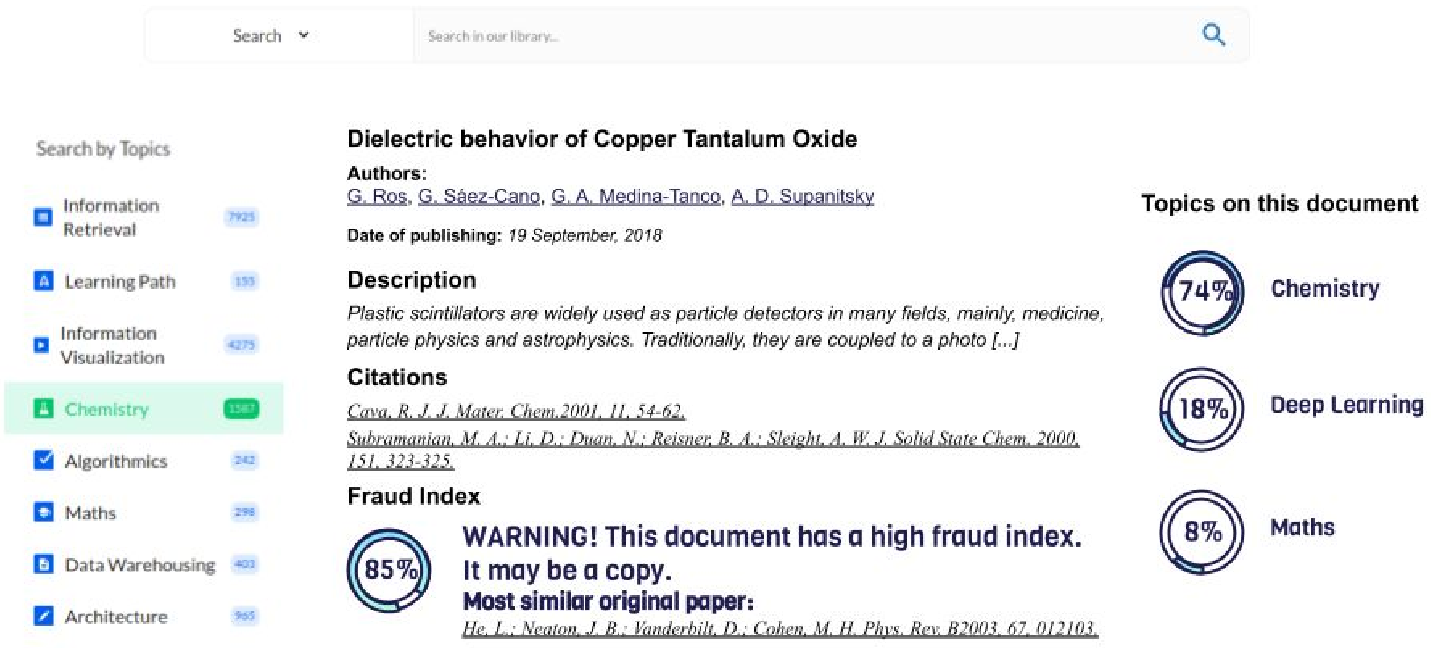
Again, the implementation of these user satisfaction metrics would involve an extra cost in terms of time in the development of the system but would be a valuable tool to evaluate the performance.

Conclusions

We have designed an interesting system that could help researchers look for relevant information for their investigations and establish good performance metrics for themselves and their “competitors”.
The development of this system could take even months to be done, and maybe some of the techniques we’ve proposed don’t provide the performance we desire. But we think that we’ve established a good starting point for a project that could be done in the future.

Annex

Annex 1: Example of how the interface of our web page might look like.



Dielectric behavior of Copper Tantalum Oxide

Authors:
G. Ros, G. Sáez-Cano, G. A. Medina-Tanco, A. D. Supanitsky

Date of publishing: *19 September, 2018*

Description

Plastic scintillators are widely used as particle detectors in many fields, mainly, medicine, particle physics and astrophysics. Traditionally, they are coupled to a photo [...]

Citations

Cava, R. J. J. Mater. Chem.2001, 11, 54-62.

Subramanian, M. A.; Li, D.; Duan, N.; Reisner, B. A.; Sleight, A. W. J. Solid State Chem. 2000, 151, 323-325.

Fraud Index

85%

WARNING! This document has a high fraud index. It may be a copy.

Most similar original paper:

He, L.; Neaton, J. B.; Vanderbilt, D.; Cohen, M. H. Phys. Rev. B2003, 67, 012103.

Topics on this document

74%	Chemistry
18%	Deep Learning
8%	Maths