Universidad Carlos III de Madrid

SpringerLink Discipline Prediction

Machine learning applications

Final Project

Authors:

Daniel de las Cuevas Turel - 100406666

Claudio Sotillos Peceroso - 100409401

Pablo Reyes Martín - 100409333

Bosco de Enrique Romeu - 100406718

1. First task: NLP and Topic Modelling

Introduction

For this first task we had to select a dataset and implement a full text processing pipeline, extraction of topics and vector representation using LDA, and finally the calculation of semantic distances using Gephi.

Regarding the dataset selection, we opted for the *American publishing company of academic journals and books, SpringerLink*. In the springer webpage we could find articles from a wide range of disciplines. We focused on ten disciplines to limit the number of possible articles: “Computer Science”, “Philosophy”, “Mathematics”, “Psychology”, “Biomedicine”, “Criminology and Crime Justice”, “Geography”, “Education”, “Physics” and “Economics”. From these disciplines we selected 2,000 instances per theme, obtaining a total of 20,000. This number was reduced since some articles did not have an abstract.

Corpus acquisition

The challenge here was scraping from the web all the information we needed to store it in our database. We did 2 phases of web scraping to obtain the features we needed to perform the thematic analysis.

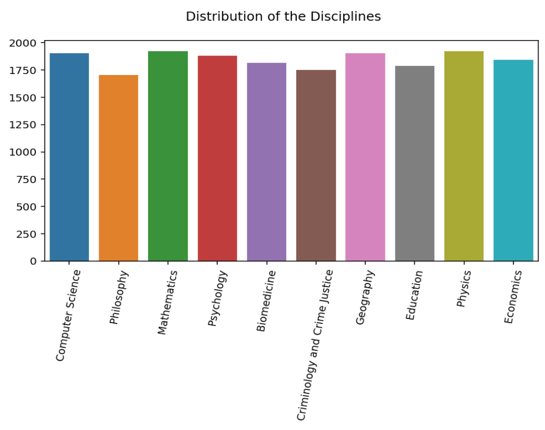
First, we performed an initial ‘outer’ extraction obtaining the articles’ titles and urls, iterating through each discipline’s url. Then, we did an ‘inner’ web scraping where we went through the urls’ columns, and we obtained the remaining features, which were inside of our own article page. These features were: abstract, publication date, authors, journal, accesses, keywords and target (discipline the article belongs to). The first process took 1 hour to complete and the second one 4 hours.

Finally, once we generated a pandas dataframe with all the variables, we deleted those rows which did not contain an abstract, since these would be useless for the topic modeling task, and we saved the clean dataset using pickle format.

Data processing and analysis

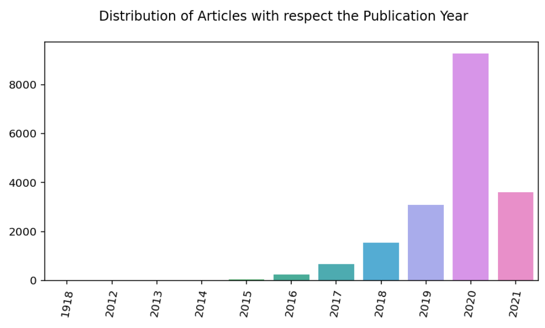
We started off by doing a data analysis of the documents. The mean abstract length was around 182 words, and the number of documents 18,447.

We plotted the distribution of disciplines counting the number of occurrences of each one. As we can see, the missing abstracts were present homogeneously (the remaining until 2,000).

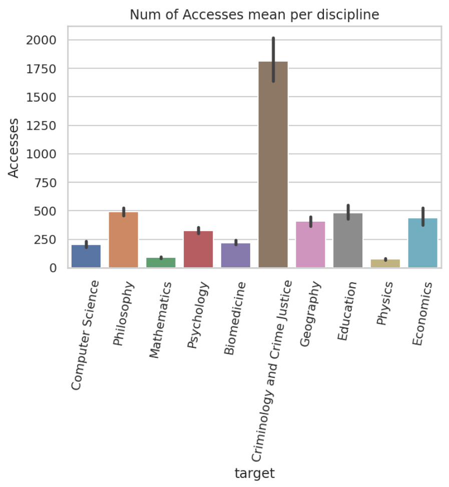


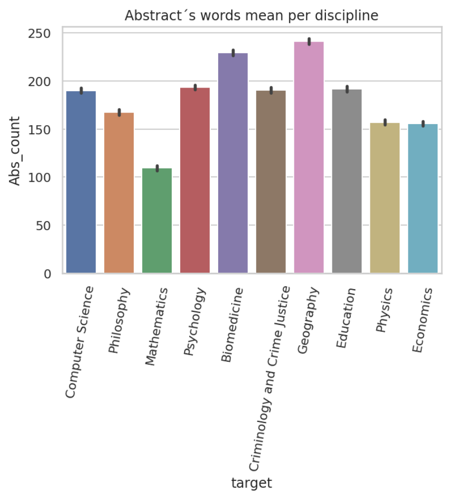
We removed the column ‘urls’, since it would not have been useful in the future. In addition, we converted the ‘accesses’ column into integer values. We did so by applying a function using lambda, where we checked if the type of the variable is a string, and we transformed these strings into integers by splitting the value where the number ended.

Furthermore, we extracted the year and month from the column ‘publication date’ and saved it into two new columns. We also showed the distribution of articles with respect to the publication years. As we can see, 2020 is the most common year for these articles, but there is at least one published in 1918!.



Finally, we checked the mean number of views and abstract words per discipline. Note that in the first case, criminology and crime justice has significantly more clicks than any other theme.





Corpus preprocessing

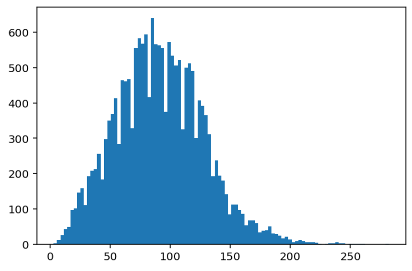
Regarding corpus preprocessing, we coded to different pipelines: one for the abstract and another one for the titles.

The one for the titles is simple. It performs tokenization, lemmatization, Part of Speech tagging (checks if the title is either a noun or a proper noun) and removes the stopwords: those that appear very often in all contexts and are not relevant. The one for the abstracts is similar to the previous one, but in this case it also keeps only alphanumeric tokens, and the valid PoS are now verbs, nouns, adjectives and proper nouns. This is due to the fact that abstracts contain semantically different information than titles, so they have to be processed in a slightly different way.

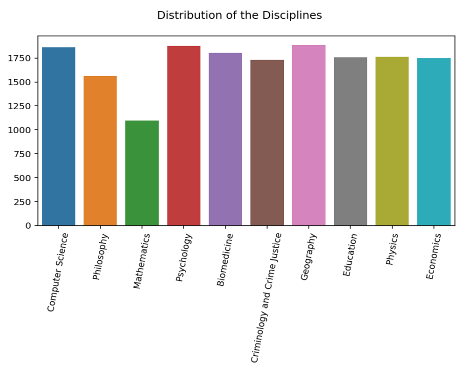
After the first stage, we added the column "lemmas" to our data frame and stored it in an xlsx format as a checkpoint. This saved us some time between phases, so that we did not have to run all the code every time.

Coming up next, we decided to do N-gram detection. The reason was that we had an intuition that this type of words would be frequent in the contexts we were analyzing. We also believed it was helpful to characterize the discipline of a certain topic. We tried the parameters we saw in class: a min count of 2 and a threshold of 20, and found that the job being done seemed reasonable. The algorithm detected pairs like "white-gaussian" or "false-alarm", which are typically together in these scientific areas. There were 207,175 n grams found, which constitute about 11% of the words in the corpus. This finding justifies the use of this technique, so we decided to keep the transformation and continue.

Now we decided to limit the number of projects depending on the number of lemmas found in each one. We can visualize the distribution of the number of lemmas per project before cutting.



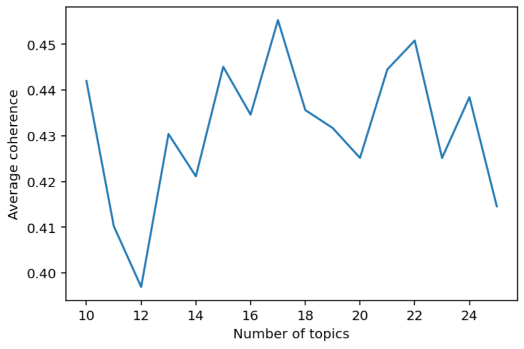
Before we saw that mathematics had a shorter number of abstract words than the rest of them. Based on this information, we selected 40 as the lower bound. After this selection, the number of projects was reduced to 17,090 and the distribution of projects per discipline was similar to before, so we did not change the initial dynamics of our selection, which was good.



The next step was transforming the corpus to a bag of words representation. For that, we first selected each element of the corpus, splitted them, and we built the dictionary. Then, we wanted to reduce the size of the dictionary and remove the unmeaningful words. Hence, we cleaned the dictionary using the filter extremes function. We kept the default values, which would keep the terms that appear at least in 5 documents, and that do not appear in more than 50% of the documents. In our case, this would mean that words with higher appearances than 8545 will be deleted. This happens with the word study (8722 occurrences). With this filtering we reduced the dictionary from 85,858 to 19,389 terms. Finally, we iterated through every doc in the corpus, and we built the corpus bow using the dictionary.

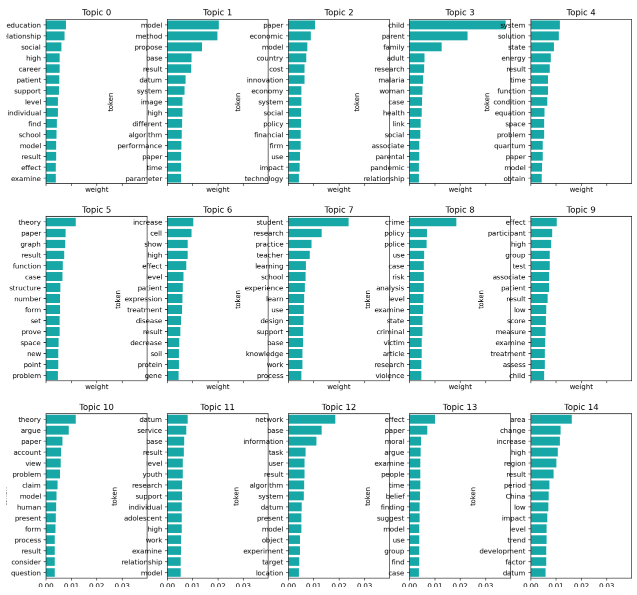
Initial topic modeling and vocabulary cleaning

After the previous stage, we decided to try to design an initial topic modeling using our intuition to see what words could be removed from the corpus to clean the vocabulary, and to have a first glance at the topics generated by LDA. We started with 15 topics and built the topic-token matrix for every topic. Then, we represented the weights of each token for every topic in 15 joint barplots. We can already see that the results make some sense, but there are words which are too general so we need to remove them.



For general word removal we also reused the list given in one of the class labs, checking if there was any word that we wanted to keep. In addition, we manually defined some other ones, like ‘research’, ‘high’, ‘low’, ‘time’, and so on. These words should not have a determining semantic effect in the topic modeling.

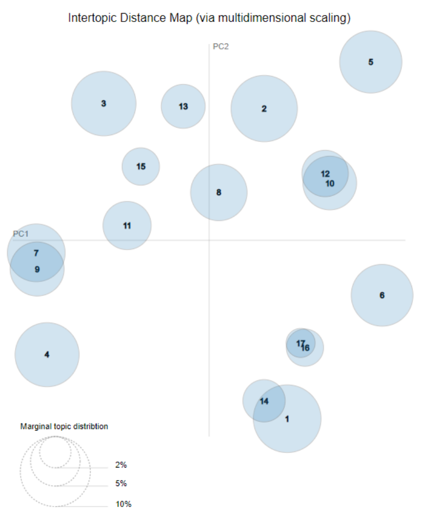
After removing specific stop words we got 16,632 documents. Thus, the dictionary needed to be re-constructed and re-filtered. Hence, we repeated the same process than before, with the new clean dataset. We first built the dictionary from the corpus based on the new clean lemmas, and then we filtered the extremes with the same parameters than before. This filtering resulted in a dictionary with 18,996 terms. Finally, we saved this last dataframe and clean dictionary, which would be used to build the final corpus bag of words in the final topic modeling.



Topic modeling

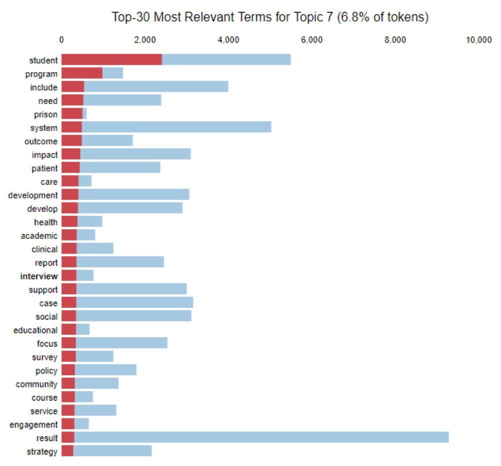
Now we have the dataframe and dictionary ready to build the final topic modeling. The only thing we have to do is choose a number of topics that will be used. For that decision, we have the coherence evaluation that can give us an idea of the optimal result. However, the best pick will always be subjective to the author. We tested for a range between 10 and 25 topics, since in our first experiment we noticed that a number higher than 30 dropped the coherence by a lot, and the same happened with less than 5. This makes sense, since we need each topic to be as semantically meaningful and specific as possible. The optimal value was 17.

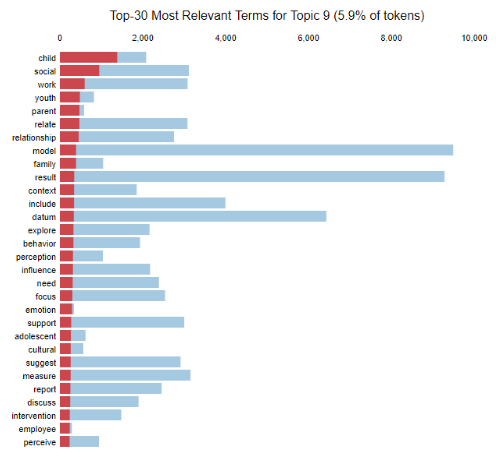
Another thing to take into account is that our articles come from 10 disciplines, so having a model with exactly 10 topics would be pointless.

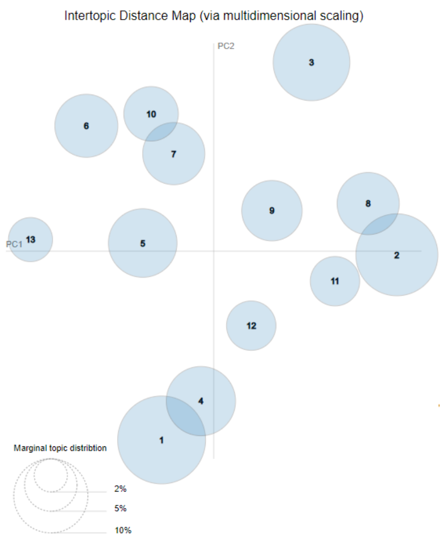


We then used pyLDAvis to plot the intertopic distance map between topics. We started with 17 to see how the topics would be placed in the 2-dimensional principal components space. As we can see on the right, some topics are enclosed by others. This happens because there are more topics than disciplines, so they can overlap. The difficulty here is that with a higher number of topics it is easier to recognize the discipline it belongs to, but they are redundant. On the other hand, with a lower number of topics each one becomes more general, so it is harder to classify them, but they are more separated from each other.

Let's see an example. If we check the map, topics 7 and 9 share a big part of its meaning. If we check the most relevant terms for each one, we can see that both share some kind of social meaning (student-child), but both have different overall interpretations.



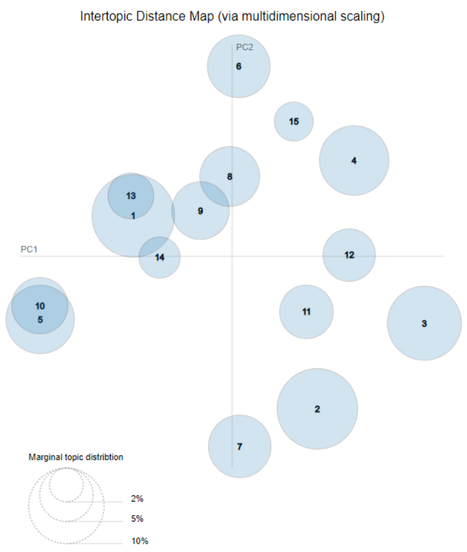




When using 13 topics (plot on the right), we see what we explained before: the topics are less likely to be overlapped, and this translates into less redundancy in the main words that characterize each one.

After seeing the pros and cons of the two previous approaches, we decided the best option would be to go for a number in the middle, namely 15. We were aware some topics would overlap but we wanted to be able to extract the main idea of each one easily.

Naming the topics



Once we had obtained the final topic model, we had to name each one accordingly. During this process, apart from finding redundant topics, we noticed some of them were garbage ones. They did not give a clear meaning of what they were representing. An example of this topic included the following terms: ‘patient’, ‘datum’, ‘loss’, ‘security’, ‘compare’, ‘result’, ‘self-compassion’, ‘risk’, ‘network’... As we can observe, they mix different terms and we cannot classify it.

After naming all the topics we built the final dataframe which we will use to do a semantic similarity analysis and to compute the edge and node files for Gephi.

Analyzing semantic similarity

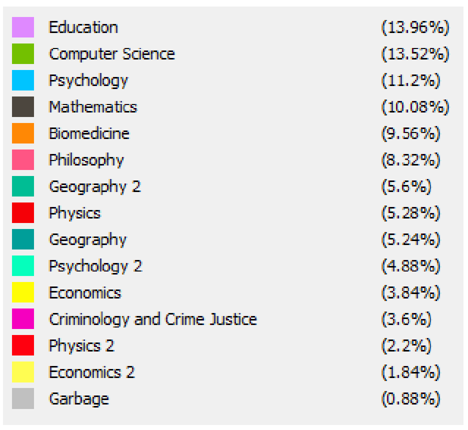
The next step was building the similarity matrix and reducing its dimensionality by keeping only non-zero components and its triangular part (since the matrix is symmetric). The ratio of non-zero components was 0.86 and the estimated number of links in the full corpus was around 350,100.

Gephi

We built the nodes graph of 2500 nodes using an id. We thought of having 100 links/weights per node. For 2500 nodes we needed around 250000 links in the sampled corpus. We adjusted the “thr” value to 0.66 in order to obtain 253124 links, which are close enough.

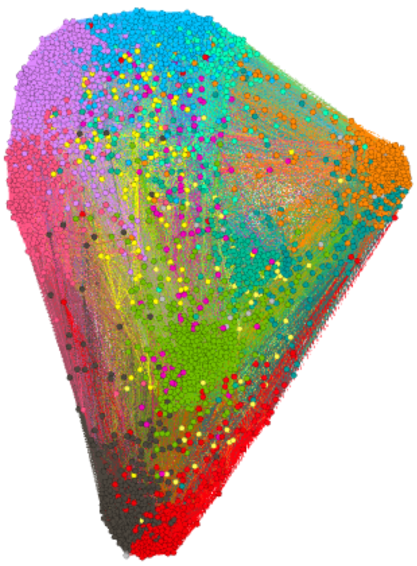
Once saved the sample of nodes and their respective edges let's start with the Graph deployment.

We will generate a fully undirected graph (both nodes and edges are undirected). Having inserted the nodes and the edges in Gephi, the first thing that we do is to color the nodes as shown in the image. Those topics with similar names (i.e. Physics and Physics 2) are colored with similar colors. After coloring, we increased the size of the nodes to 40 in order to distinguish them better.



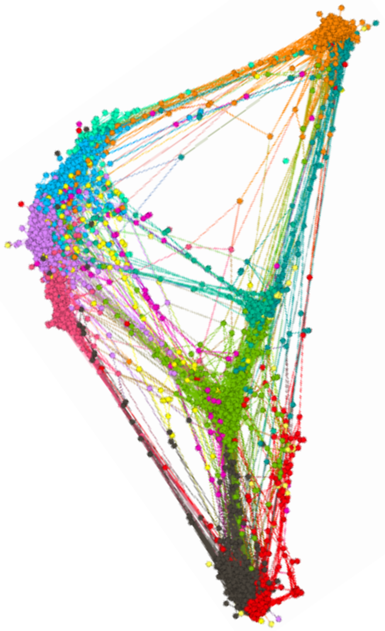
🟪	Education	(13.96%)
🟩	Computer Science	(13.52%)
🟦	Psychology	(11.2%)
⬛	Mathematics	(10.08%)
🟧	Biomedicine	(9.56%)
🟥	Philosophy	(8.32%)
🟩	Geography 2	(5.6%)
🟥	Physics	(5.28%)
🟩	Geography	(5.24%)
🟩	Psychology 2	(4.88%)
🟨	Economics	(3.84%)
🟪	Criminology and Crime Justice	(3.6%)
🟥	Physics 2	(2.2%)
🟨	Economics 2	(1.84%)
⬜	Garbage	(0.88%)

After this, in the layout section we run the ForceAtlas 2 approach. What happens with this approach is that nodes repulse each other,
like magnets of opposite poles, while edges attract the nodes. When the graph balances, we obtain a very reasonable representation. For instance, we can observe that Physics, Mathematics and Computer Science are pretty close together, meaning that they have much more in common than with Psychology (that is on the opposite pole.
The graph on the left shows the stabilized ForceAtlas 2 representation.



To finish up, we will do some filtering in order to have a clearer graph.

First, we will make a partition in the attribute “main_topic_15” in order to eliminate the Garbage topic. Inside this filter, we have implemented an “Edge Weight” filter, with which we only keep those nodes connected with weights between 0.85 and 0.99. Due to how this representation works, when we remove the links/weights lower than 0.85, the nodes which are not connected disappear since they are repelled. This second graph is the one obtained after performing the mentioned filters.



2. Second Task: Classification

During this second task we have implemented different strategies using the information extracted from our corpus during the language processing and the topic modelling stages to try to predict the discipline to which each document belongs to. We have followed two main courses of action to solve the classification problem, first using a *discriminative* vector representation and second a semantic embedding for the documents: TF-IDF and Word2Vec.

Term Frequency - Inverse Document Frequency (TF -IDF)

Firstly, since our documents belong to 10 different disciplines, and so we have lots of terms that are domain specific, we decided to use TF-IDF over BoW to embed *discriminative information* in our initial data.

Singular Value Decomposition

Furthermore, we decided to compare the performance of TF-IDF on it's own, but also, due to the high dimensionality of the data (our vocabulary contains 18 996 words), TF-IDF combined with Singular Value Decomposition (SVD). The reason why we selected SVD as dimensionality reduction technique over other methods was to exploit the correlations between semantically related terms SVD is able to capture.

We decided to truncate the SVD based on our sample information instead of using cross validation techniques: the computational time would've increased exponentially and we don't believe that tuning this parameter would've provided considerably better results. Using our entire corpus we decided to fix the number of components to 4500 (around 25% of the initial dimensions) which explained just above 80% of the overall variance.

Classifiers

We believe that, to solve the classification problem with this first approach, it was interesting to compare four techniques that model the boundary between classes (SVM and LDA) but also the density distribution of the data (KNN and Naïve Bayes). Due to the characteristics of TF-IDF, we expect the discriminative models to outperform the generative models. Here are the parameters we have decided to fine tune for each classifier:

- Naïve Bayes: Whether we learn the prior distribution of the classes or not and whether we need to tackle the problem of zero probability using Laplace smoothing.

- KNN: The number of neighbours and whether to use distance weighted contributions of the neighbours.
- Linear SVM: The penalty (l1 or l2 regularization to check if it would be adequate to embed feature selection in our classifier) and the loss function.
- LDA: The solver.

To cross validate all these parameters we have used the sklearn module *GridSearchCV* with five folds (distribution of 0.6, 0.2, 0.2 for training, test and validation). Also, note that it is not possible to apply Naïve Bayes with the TF-IDF + SVD approach (since the assumption of N. Bayes is that the features follow a multinomial distribution and SVD produces negative value features) nor to apply LDA with TF-IDF since it cannot handle sparse data.

Results and Conclusions

Some of the parameters tuned by sklearn are worth mentioning. First, regarding N. Bayes the best option was not to fit the prior distribution of the classes (hence a uniform is assumed); although we could actually expect it as we have, except for mathematics, almost the same number of documents per discipline. Second, the loss selected for the SVM training was the modified huber loss; it is a function that is less sensitive to outliers so we will need to address the problem of novelty detection further on. As for the performance, there are also some interesting things to say.

Classification error	TF - IDF Representation			TF - IDF + SVD		
	N. Bayes	**KNN**	**SVM**	**KNN**	**SVM**	**LDA**
Train	0.93	1	0.93	1	0.91	0.98
Valid	0.83	0.78	0.83	0.77	0.83	0.82
Test	0.83	0.78	0.84	0.78	0.84	0.83

The first thing that is obvious after observing the performances, is that, unfortunately for us, applying the SVD transformation has no effect on the overall results (only notable difference is that SVM is slightly less overfitted using SVD). Furthermore, the KNN algorithm is not generalizing correctly (model is clearly overfitted); the reasons are very clear to us, KNN does not handle well data with a high number of features but also considering that we have less than 20 000 instances in our training data and either 4500 (with SVD) or 18 996 features, a non-linear model is not the best way to go.

On the positive side, our intuition was correct, discriminative models are able to use the TF-IDF representation to model pretty correctly the class boundaries; these models obtained the best performances although we would retain SVM as it is the model less affected by overfitting. Naïve Bayes obtained good results too, but we feel that the uniform prior may have a lot to do with the outcome.

Word2Vec in classification

In our second analysis axis we decided to take advantage of the Word2Vec model to get semantic embedding of our documents. Next, we decided to use, opposed to our first study approach, a non-linear dimensionality reduction technique, t-Distributed stochastic neighbour embedding, to attempt the classification in three different ways; using exclusively the tSNE representation, using the tSNE and metadata (preprocessed) and finally combining Word2Vec and Fisher discriminant Analysis (FDA).

Word2Vec representation of documents

To obtain the embeddings of our documents we designed two different pipelines, one where we summed all the word embeddings and a second one where we averaged them. As we showed above in task 1, not all disciplines have similar length abstracts so we felt that comparing both of these approaches was relevant. We expect the average approach to perform better as we believe the difference in abstract length could partially discriminate between the different disciplines.

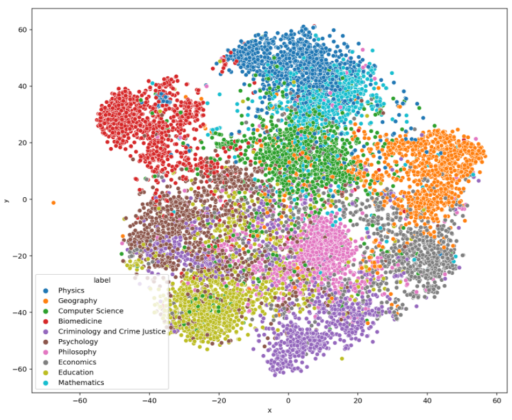
We applied these pipelines with three different data; first with the documents abstracts but also with the title and the keywords. In the context of developing a pipeline with both tasks nested it would be time friendly to skip the preprocessing of the abstracts so if good solutions were to be obtained with the keywords or the title, they would be very useful.

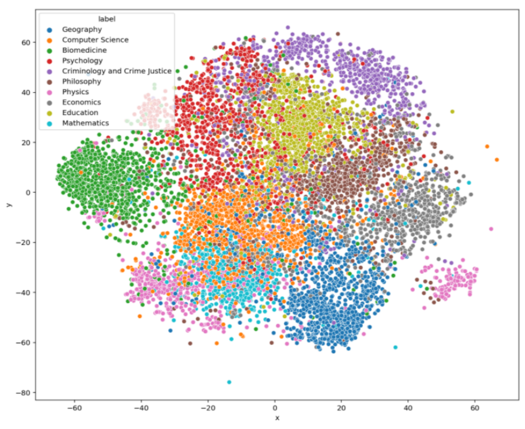
To apply the Word2Vec model we have used a pretrained network from Gensim.

Word2tSNE

The six different (scaled) representations obtained in the last section were used as input for the tSNE algorithm. Regarding the parameters of this model, we couldn't cross validate them as it is highly time consuming. Therefore we decided to analyze different values for the number of components and the perplexity, and we concluded that the best way to go was to use two components (also for visualization purposes) and 30 *neighbours*. Lower values for the perplexity resulted in poor local density estimations.

After visualizing all the different combinations, as expected, the mean Word2Vec representation combined with tSNE showed better results; disciplines were further apart and there was less noise in the different clusters (data points belonging to other disciplines). Among these three 2D representations, the best one was the one obtained with the abstract (left) and while the result using keywords was surprisingly acceptable (right), the representation using the title was very disappointing.





Both of these visualizations are very informative, the left one because of the clearly recognizable clusters (the only cluster that seems to be all over the place is the one associated with Criminology and Crime Justice); but we especially liked the one on the right as both physics and mathematics documents are mixed with other disciplines and this is very coherent as they both contribute (of course) to many research areas. However, for obvious reasons we decided to continue our study using the mean Word2Vec + 2D tSNE approach (left).

Classification using tSNE representation

We have used the tSNE representation for classification with two different approaches, first by analyzing the classifiers performance with just the two tSNE components and second by adding new features with document metadata. For the second approach we have applied different preprocessing techniques to the extra features. Because of the non-linear projection obtained with tSNE we have selected two non-linear classifiers; RBF - kernel SVM and KNN.

Preprocessing:

Among our metadata we have the following features; year of publication, number of lemmas in the abstract, number of accesses and number of topics. Before doing anything, because the number of documents in our corpus published in 2020 are dominant, we decided to eliminate this information; we want the classifier to generalize equally for documents published at any time.

Furthermore, after scaling numerical variables and applying a uniform transformation to the number of accesses (extremely right-skewed), we used minimum redundancy Maximum Relevance (mrMR with correlation as redundancy measure and Mutual Information as relevance measure) to decide what features to keep for our study. The number of topics had a mrMR of almost zero so we removed it; our final data was then the two tSNE projections, transformed number of accesses and number of lemmas.

The last important preprocessing step we performed was novelty detection, motivated by both the tSNE visualization (where we can see some noise) and the results from the first classification (distance selected by SVM and KNN's poor performance). The most appropriate technique here was to use a Gaussian Mixture Model as we have 10 different clusters and we wish to get rid of those data points that are clearly outside their true cluster. We did an initial *gridSearch* with KNN and SVM to check what contamination ratio we should select; KNN shed a light on the fact that we needed to select the highest possible value (in our case 0.10) as the performance , but most importantly the optimal number of neighbours, increased with the this parameter.

Classifiers and results:

As stated before the two classifiers we have selected are RBF - kernel SVM and KNN. The parameters we have tuned for the SVM are the regularization parameter C and the curvature of the kernel gamma. As for the KNN we study the performance of different numbers of neighbours, weighted contributions and the distance metric. All of these parameters are cross validated using *GridSearchCV* again, although for the SVM we studied the relationship between gamma and C by setting a default value for each of them and cross validating the other.

After cross validating the parameters, the distance metric selected for KNN (using only the tSNE projections, not the metadata) was chebyshev; which shows how KNN exploits the two projections of the tSNE algorithm as chebyshev distance is computed by studying *marginal* distances in the two different directions. Here are the results obtained with all the different configurations.

train/test Accuracy	**tSNE**	**tSNE + metadata**
RBF SVM	0.77 / 0.74	0.80 / 0.71
KNN	1.0 / 0.75	1.0 / 0.73

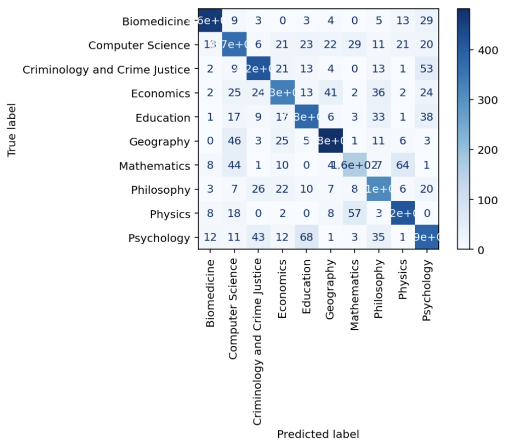
Mainly because of the novelty detection performed with GMM, we were expecting the tSNE + metadata configuration to perform better, but no improvement is visible. Using extra features other than the tSNE projections leads to a slightly more overfited SVM and to a KNN performing equally bad as without the metadata. On the bright side, tSNE combined with RBF - kernel SVM seems to generalize the data pretty well.

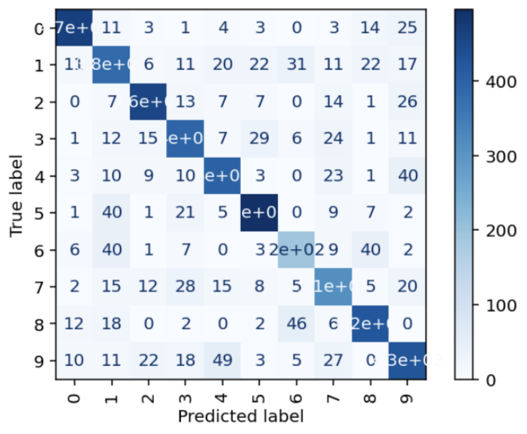
Classification using Word2Vec and Fisher Discriminant Analysis

Throughout the study we have repeatedly observed that discriminative classifiers outperformed other types, so as a conclusion to our work we have decided to try to combine SVM (which is the classifier that performed better) with a dimensionality reduction technique that could help boost its performance: FDA. By minimizing the variance within the classes and maximizing the distance between the classes we expect the accuracy to increase.

A very known pipeline in Machine Learning is using PCA to remove correlation in our data and subsequently applying FDA. Here we have decided not to use PCA as the correlation inside the Word2Vec embeddings is minimum. As closure to our task 2 notebook we have implemented the given pipeline to show how the results are indeed very similar to those we obtain in this section as PCA is not able to remove strong correlations (nonexistent), but we won't be discussing them in the report.

As expected, the accuracy obtained with FDA and SVM is better; the algorithm is able to fit more precisely the boundaries taking advantage of the FDA projection resulting in a train accuracy of 0.81 and test accuracy of 0.79. To visualize this improvement we compare the confusion matrices obtained with SVM + tSNE (left) and SVM + FDA (right).





Using these matrices we can see how FDA does an excellent job as the accuracy increased due to the fact that similar disciplines as Mathematics and Computer Science or Psychology and

Education are better separated opposed to tSNE; where the classifier had trouble distinguishing documents belonging to these disciplines.

Conclusions

After this extensive study, we have collected a lot of information on our data and what tools are better suited to complete certain ML tasks. The main conclusion is that, to complete the classification task, we need to apply techniques oriented to exploiting the boundaries of the classes to help the subsequent (discriminative) classifier; combined with novelty detection techniques as the noise in our data is considerable. In this context the best approach we have found is combining Word2Vec with FDA and SVM.

Finally, although the TF-IDF approach returned higher accuracies than the Word2Vec approach, it was at the cost of having overfitted models; the results obtained with Word2Vec were more robust. This, as we have visualized in the last section with the confusion matrices, is probably due to the ability of Word2Vec to discriminate between disciplines that are more closely related by extracting the semantic information of each of the documents, while TF-IDF, although is capable of discriminating very divergent disciplines using very different terms, has more difficulties discriminating between similar disciplines that frequently use similar (if not the same) terms.

Code user's manual

This file contains three python notebooks and three folders plus the report.

**Python notebooks brief summary:

- Web_scraping: It contains the code and explanations of our web scraping procedure. ¡WARNING! If you are willing to execute the code of this notebook, be aware that it will take a long time in executing (it took us around 5 hours, even with "ray framework" parallelization).

- Task 1: In it, we fulfilled the first task. It consists of an initial preprocessing and analysis of our data set, the text processing pipeline, the LDA topic modeling and how we obtained the Nodes/Edges CSV files for our Graph Analysis.

- Task 2: For task 2 we have decided to compare the Abstract vectorization approaches, TF-IDF and Word2Vec with different classifiers. ¡WARNING! If you are willing to execute the code of this notebook, it will take a lot of time. Concretely, the cells that take more time in executing are those related to the TSNE representation.

In order to run the code inside these notebooks you just have to change your Drive path.
Substitute it with the path where you have stored the final project file.

```
Just change this:  |----------------------------------------------------|
                   v                                                    v
   local_folder = '/content/drive/MyDrive/ML Applications/Final Project/'
```

**Description of the folders:

- Non-Clean Datasets: In it, we have stored the Data sets obtained with web scraping. The "firstphase.pickle" is the one which belongs to the outer web scraping (columns: [title,url, discipline name]). The "Dataset.pickle" is the one which belongs to the inner web scraping

  [titles,Abstract,KeyWords,Authors,Journal,PublicationDate,Accesses,urls,target]).

  The "Dataset_clean.pickle" is the same as the previous one, but with some cleaning modifications.

- Task 1 Checkpoints: It contains the definitive Data set after the initial cleaning of the initial web scraping Data set. It also contains the several checkpoints performed along the task 1 notebook.

- Task 2 Checkpoints: The same as task 1 checkpoints but non-structured.

References

Some of our code was obtained from the following sources:

- This course's notebooks (for task 1 and 2)
- Pablo Martinez Olmos' course in Neural Networks (for task 2: probabilistic PCA)

Other acknowledgements:

- Data extracted from: https://www.springer.com/