

**Machine Learning and Content Analytics**

**Project**

**Team: Data-19**

Pantelakis Georgios F2822011

Papadakis Athanasios Marios F2822012

Porikis Christos F2822014

September 2021

Contents

[1. Introduction 3](#_Toc81769691)

[2. Mission 4](#_Toc81769692)

[3. Data 5](#_Toc81769693)

[Data Collection 5](#_Toc81769694)

[Explorative Analysis 5](#_Toc81769695)

[Data preprocessing 7](#_Toc81769696)

[4. Methodology 8](#_Toc81769697)

[Baseline 8](#_Toc81769698)

[Feed Forward Model 9](#_Toc81769699)

[Convolutional Neural Network Model (CNN) 10](#_Toc81769700)

[FastText 11](#_Toc81769701)

[Transformer 11](#_Toc81769702)

[K-Means Model 11](#_Toc81769703)

[5. Results 13](#_Toc81769704)

[Feed Forward Model 13](#_Toc81769705)

[CNN 15](#_Toc81769706)

[FastText 17](#_Toc81769707)

[Transformer 18](#_Toc81769708)

[K-Means Model 19](#_Toc81769709)

[Clustering only DE from the abstract 19](#_Toc81769710)

[Clustering with 2 features (sentences, EU Call) 19](#_Toc81769711)

[Clustering with 3 features (sentences, EU Call, argument embeddings) 20](#_Toc81769712)

[Conclusion 21](#_Toc81769713)

[6. Members/Roles 21](#_Toc81769714)

[Backgrounds 21](#_Toc81769715)

[7. Time Plan 22](#_Toc81769716)

[8. Bibliography 22](#_Toc81769717)

[9. Other Sources 22](#_Toc81769718)

# Introduction

Scientific publications are very important nowadays, since scientists from different domains publish their research, draw conclusions and make claims from their experiments and present results as evidence to support these claims. But there is a problem because each scientific domain has its own vocabulary and maybe difference in the presentation of the results and needs expertise in order to understand the methodology and results of those experiments. Here comes the need to train the machines and be able to understand all those different domains, but the most intriguing part is that although those publications are structured to the human eye, they are unstructured resources from a computer’s point of view.

For this task argumentation mining is used in order to extract the structures and the arguments of the scientific publications with the use of machines. Argumentation mining is a research area within the natural-language processing field. The goal of argument mining is the automatic extraction and identification of argumentative structures from natural language text with the aid of computer programs. Such argumentative structures include the premise, conclusions, the argument scheme and the relationship between the main and subsidiary argument, or the main and counter-argument within discourse.

Even though machines can automate this process in the future, the human factor is still needed in order to create enough data to feed the machines. But still one person is not enough, as some people may have divergent views on the same publication since there is no right and wrong in the process of discovering what is claim or evidence or even if a citation between 2 papers is positive, negative or neutral.

The fact that even people themselves cannot be 100% sure about which sentences is the central claim, what are the supporting statements to this claim and how those arguments are structured in each publication, makes this project interesting and makes room for far more room for improvement in the following years. By solving the problem of automatic detection and extraction of arguments can help in many other domains, such as automatic correction or support written essays or create a chat bot that can dialogue with human beings in non-predefined chats.

# Mission

This project’s focus is to solve the aforementioned problem and speed up the whole process with little need for human intervention. For instance, there are medical papers that do not show enough evidence to support the claims of the publishers, or they do not provide anything new in comparison with older publications.

In order to resolve the aforementioned problem, text classification was used which is the process of categorizing text into organized groups. By using Natural Language Processing (NLP), text classifiers can automatically analyze text and then assign a set of pre-defined tags or categories based on its content. There is a two-step procedure followed, where in the first step features are extracted from the model and are transformed into a form that each model can understand, followed by the second step, where the aforementioned features are fed into the models in order to make predictions.

Argument mining, which is the main method of this project and is a research area within NLP, is the automatic extraction and identification of argumentative structures from natural language text with the aid of machine learning. The goal of argument mining is to teach machines about argumentative structures and automate the identification and classification of arguments within texts. This allows specific searching of arguments related to a certain topic.

Therefore, the project approach that we used was firstly to exploit the unstructured data that we were provided with, which consisted of biomedical abstracts in a broad number of topics. In particular, after the collection we proceeded with a deeper evaluation of those papers, where we identified and annotated the argumentized statements for each one of them, giving the appropriate category (claim or evidence). Consequently, a minority of papers were excluded from our dataset, as they did not provide enough evidence for the claims, and as a result would not provide any value to the analysis. Subsequently, we finalized the data that we used for our analysis.

After we had created a valid dataset that we could utilize, we used Python to conduct our analysis. Firstly, a baseline model was created in order to have a starting point and use this as a basis to evaluate and compare our models with. Secondly, Fasttext approach was used. Fasttext is a word embedding method, which instead of learning vectors for words directly, it represents each word as an n-gram of characters (artificial” with n=3, becomes “ar, art, rti, tif, ifi, fic, ici, ial, al”). Thirdly, a CNN and a Feed Forward Network were deployed and compared. CNN is a class of deep, feed-forward artificial neural networks and use a variation of multilayer perceptrons designed to require minimal preprocessing, while feedforward neural network is an artificial neural network wherein connections between the nodes do not form a cycle. Another model we ran was Specter model. With this we were able to assess the similarity of any paper with the others in the dataset. Finally, for the abstract clustering we used k-means algorithm, where we created embeddings and fed them to the model as a three-phase procedure (only the abstract, combination of abstract with 1 feature, combination of abstract and feature as well as argument embeddings).

# Data

In this section, we describe the steps that we took in order to collect data for our analysis including the sources from which our data came from, our data processing-cleansing methods as well as an overview of the characteristics of our dataset.

## Data Collection

As far as the collection of our data is concerned, around 115 documents were given across each team in order to try to generate more data for our analysis. The task of each team was to try to label all the abstracts using the labels background, aim/objective, method, result conclusion regarding the structure, and evidence, claim regarding the argument in each abstract (if there was any). At the final stage of data acquisition, a dataset combined with the documents of all the teams was generated and used in the contexts of this project together with some pre-existing data that were given by our professor.

## Explorative Analysis

In order to process our data, we had to load the datasets that were in json format and then combined the pre-existing data with our generated dataset and created 2 datasets with argument and structure labels. After conducting an explorative analysis on our dataset, we can see the frequency of our labels in the argument’s dataset and the structure dataset. More specifically, in the argument’s dataset, we observe that:

* 21471 sentences (around 69 percent of our dataset) had no label
* 6203 sentences (around 20 percent of our dataset) were labeled as “evidence”
* 3419 sentences (around 11 percent of our dataset) were labeled as “claim”.

Chart, pie chart

Description automatically generated

Figure 1 Pie Chart with the percentages of labels

On the other hand, the structure dataset had the following characteristics:

* 2705 sentences (around 26 percent of our dataset) were labeled as “RESULT”
* 2129 sentences (around 20 percent of our dataset) were labeled as “BACKGROUND”
* 1856 sentences (around 18 percent of our dataset) were labeled as “OBJECTIVE”
* 1602 sentences (around 15 percent of our dataset) were labeled as “METHOD”
* 1242 sentences (around 12 percent of our dataset) were labeled as “CONCLUSION”
* 1014 sentences (around 10 percent of our dataset) were labeled as “NEITHER”

Chart, pie chart

Description automatically generated

Figure 2 Pie chart with the percentages of structures

## Data preprocessing

We tokenized all the sentences into words and we removed stop words such as 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're". Then, we lemmatized our tokens which identify the part of speech for each word and its meaning in a sentence and convert the word, like ‘better’ becomes ‘good’ and ‘talking’ becomes ‘talk’ after using lemmatization. Afterwards, we applied a function that cleans our text from symbols, punctuation or shorthand of words like “we’ll” that is converted to ‘we will’ and converted them to lowercase. Below is a sentence before and after using the clean function:

**Before:**

'This result is achieved through novel cross-link agent made by boron- and fluorine-containing heterocycle that can react between themselves upon UV- and white-light exposure.'

**After:**

'This result is achieved through novel cross link agent made by boron and fluorine containing heterocycle that can react between themselves upon uv and white light exposure '

After cleaning our data, we split the dataset to 80% train and 20% test and used random state = 42 in order to split the dataset in the same way every time we run the code (reproducibility) and we also used stratification in order to have equal representation of all labels in both datasets.

|  |  |  |
| --- | --- | --- |
| **Labels** | **Train Dataset %** | **Test Dataset %** |
| No label | 68.97% | 68.98% |
| Evidence | 20.00% | 19.99% |
| Claim | 11.03% | 11.03% |

After splitting our dataset, we then proceeded to tokenization, where we created the tokenizer, fit it on our train dataset and converted our train and test datasets to sequences using our tokenizer. Then, we converted the sequential data of our Xs to one-hot vectors and we encoded our labels in order to have one hot encoding as well, with 3 columns where it had a ‘1’ if it had the corresponding label. Lastly, as we see in the table above, we had to deal with the imbalanced dataset, so we created the class weights that can be seen below in order to make it balanced and not have a biased model.

|  |  |
| --- | --- |
| Label | Weight |
| No label | 0.48330201 |
| Evidence | 1.66693674 |
| Claim | 3.02116985 |

# 4. Methodology

## Baseline

The first model that was created, as a heuristic, is a simple model that gets a list of words from a vocabulary derived from our train dataset with the top-10 words from sentences that were claims and the top-10 words from sentences that were evidence. If we find a word in a sentence that is present in one of our two lists, we categorize this sentence as claim or evidence respectively.

To start with the model, we first split our train dataset to 2 datasets, 1 for claim sentences and 1 for evidence, then we removed stopwords and we proceeded by removing words that had a length less than 4 in both datasets, because they would be something like stopwords and we didn’t want them to affect our final vocabulary. Then, we used a counter to count the frequency of each word in each dataset and get the top 15 words from each one. We also deleted the common words that were present in both datasets as it would create a confusion in the predictions.

In order to test the accuracy of the heuristic model we created, we created a dataframe with the sentences and labels of the test dataset and we added a new column with the predicted value, which had the value of 0 at start, and we changed to 1 if it contained a word from the top words list from evidence and 2 if it contained a word from the top words list from claims.

Below we can see the actual and predicted label frequency:

|  |  |
| --- | --- |
| **Actual** | |
| **Label** | **Frequency** |
| No label | 4.258 |
| Evidence | 1.234 |
| Claim | 681 |

|  |  |
| --- | --- |
| **Predicted** | |
| **Label** | **Frequency** |
| No label | 3.483 |
| Evidence | 1.780 |
| Claim | 910 |

We can see that the representation percentage of each label is almost the same but let’s see how well the actual predictions were.

After comparing the rows where the actual label was the same with the predicted one we got a success rate of 55% which is not actually that bad for a simple heuristic model. Keeping in mind that we have an imbalanced dataset we could have higher success rate as our model could be biased, but seeing that with just 10 words in our vocabulary for claims and another 10 for evidence we got more representation of those labels in the predictions shows that our model is actually doing good, considering how simple it is, and while adding more words in the lists it would actually make worse predictions.

## Feed Forward Model

The 2nd model we created is Feed Forward neural network, which consists of the input layer where we feed our training data, pass it through the hidden layers which consist of n neurons and the model can learn better as each layer gets as input the output of the previous layer. This way it learns deeper information about our data and the more layers or neurons we add the deeper it will learn. Lastly, the output of the last hidden layer is processed in order to get to the output layer, where our model make its predictions.

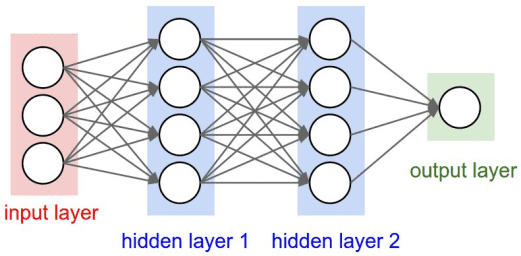


Figure 3 Feed Forward model layers

Since it is a simple Feed Forward network the only model hypermeters are batch size and number of epochs. We use a batch size of 32, which means that every time we feed our input layer with 32 sentences until all the sentences are used and we repeat this procedure for each epoch, and as number of epochs we use 20, which means that our model will run 20 times.

In the figure below we can see the architecture of our model, where we observe we have 512 neurons to feed our input train data and we use a batch normalization before the activation function to normalize the input to our activation function so that our data are centered in the linear section of the activation function. Then, our activation function is activated along with the dropout rate, in order to drop 50% of the neurons at random and avoid overfitting and we add one more hidden layer, with the same activation function and dropout rate as before, but in contrast we use batch normalization after the activation function this time. Lastly, we add the output layer.

Table

Description automatically generated

Figure 4 Total layers

For the compilation of our model, we use categorical\_crossentropy as a loss function, Stochastic gradient descent (SGD) as an optimizer and as evaluation metrics we use the Mean Absolute Error (MAE) and Accuracy metrics. We also use a callback of Early stopping which stops fitting our model if the validation loss has not decreased over 5 epochs. During the fitting of the model, we use 10% of our training data as validation to test the efficiency of our model.

## Convolutional Neural Network Model (CNN)

The 3rd model we created is Convolutional Neural Network Model, which consists of the input layer where we feed our training data, pass it through the hidden layers which consist of n neurons and the model can learn better as each layer gets as input the output of the previous layer. This way it learns deeper information about our data and the more layers or neurons we add the deeper it will learn. Lastly, the output of the last hidden layer is processed in order to get to the output layer, where our model make its predictions.

## FastText

FastText is an open-source, free, lightweight library that allows users to learn text representations and text classifiers. It works on standard, generic hardware. In order to train our model with fastText for the arguments and the structure dataset we had to preprocess the data in order to have the desired format (for example the label “evidence” had to be in the format “\_\_label\_\_evidence”). We also split the dataset into train and test (80-20) and then the train into train and validation (75-25). Before we train the classifier, we had to save the 3 dataframes into csv files which was necessary for the model. Regarding the argument’s dataset, we trained our model using the train dataset and we activated hyperparameter optimization by providing the validation dataset with the “autotune-validation” argument and then we evaluated our model with the test dataset. On the other hand, when we used autotune for the structure dataset, we noticed that the model was overfitting (having train accuracy 99 percent while on the test dataset really low) which occurs when a statistical model fits exactly against its training data but fails to generalize on new data. As a result, we experimented without autotuning with the various input parameters of the model, e.g., epochs, learning rate, wordNgrams. The model that we concluded had lr=0.1, epoch=7 and wordNgrams=1.

## Transformer

SPECTER is a model trained on scientific citations and can be used to estimate the similarity of two publications. We first create our transformer model 'allenai-specter' and then extract embeddings from this pretrained model for our abstracts. These embeddings align these abstracts in the space and this way we can find the most similar abstracts to the input abstracts with the corresponding similarity score.

## K-Means Model

For clustering a three-phase procedure was followed.

The first one is clustering of the abstracts using only the sentences from it. The initial part was to upload the two datasets (‘dataset\_aueb\_argument\_v3’,’ dataset’) and concatenate them (where we ended with 2686 abstracts before proceeding to the data processing steps. The first step was to explode the sentences, remove them from the initial listed form they were, and concatenate them again. In the next step we used TfidfVectorizer to transform the sentences into vectors of frequency for each word.

We parameterized TfidfVectorizer using min/max\_df in order to remove words that appeared too infrequently/frequently (remove the top 20% and the bottom 0.05). After fitting our data to the vectorizer, we also found the most frequent terms and distances between words. We ended up with a (2686,656) matrix (2686 abstracts and 656 common words were kept).

Finally, we created a k-means model and fit our data to it. It is a method of vector quantization that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster. As the distance parameter, Euclidean was used. As parameters for the model, we added the number of clusters, where we experimented with an abundance of numbers (4,5,6,7,8) and found that 6 produce the best results. The respected mean silhouette score of all clusters was also printed in the end, which was used as a metric to evaluate our results. Silhouette score determines how well each object lies within its cluster.

Subsequently, we created a dataframe called vocab\_frame where we have the total words out of all abstracts (776.482 words in total) in their initial form (without stemming) and the word with stemming, so we were able to spot the words in their initial form. Utilizing the above, we were able to find the top 3 words from each cluster in their initial form. In the end we named our clusters after these top 3 words, and created a plot with all the abstracts and colored them depending on the cluster they belong to.

The second phase, included the clustering of abstracts in combination with EU call as a feature. We uploaded the dataset for EU calls, and merged it with our dataset (for this case we used only “dataset\_aueb\_argument\_v3” dataset, since there wasn’t a column for EU call in the other one). After merging them, a similar procedure was followed with the exception that now we had to work with two features. In order to resolve that, we created a “TfidfVectorizer” and fitted the sentences column and then the “eu\_calls” column. After that, we ended up with two matrices, where we used “hstack” to concatenate them in one larger matrix (1015,2132). We also found the top terms for each feature (“sentences” and “eu\_calls”) and then added them together in a bigger vocabulary. After that, we did the same in order to find the words in their initial forms, by creating a “vocab\_frame” as previously mentioned. Finally, we ran a k-mean model again with 6 clusters and found a better mean silhouette value than before. After finding the top terms for each cluster we once again created a plot with the abstracts and colored them depending on the cluster they belong to.

The final phase, included the clustering of abstracts in combination with EU call and the argument embeddings. For the argument embeddings, we created a column where we have only the sentences which are either labeled as claim or evidence. Some abstracts were dropped since they did not have any sentences with claim/evidence labels. A similar procedure as stated before was followed for the other two columns. We then created three tfidf matrices, one for each feature and abstract. We concatenated all those matrices into a final larger one (783,2231). We found the top term for each feature and then added them all together in a bigger list. We created the total vocabulary, which had 518784 words with their initial form as well as their stemmed form. Finally, we ran the k-means, with 6 clusters once again (this came after experimentation), found the top words for each cluster and plotted all the abstracts with colors from the respective cluster they belong to.

# 5. Results

## Feed Forward Model

In the following plot, it can be observed that the model ran for 17 epochs before stopping, from the callback of early stopping, because validation loss stopped decreasing for 5 consecutive epochs. Both training and validation started approximately at 0.85, and finally declining at 0.7 and 0.22 for validation and training correspondingly.

Chart, line chart

Description automatically generated

Figure 5 Validation loss for Training/Validation (Feed Forward)

The following plot shows the MAE of both training and validation. They start roughly at 0.36 and end up at 0.11 and 0.18 respectively.

Chart, line chart

Description automatically generated

Figure 6 MAE for Training/Validation (Feed Forward)

In the following plot, we see that as the epochs pass, the accuracy of both training and validation gradually increases, peaking at 0.89 for the former and 0.75 for the latter.

Chart, line chart

Description automatically generated

Figure 7 Accuracy for Training/Validation (Feed Forward)

The following plot shows a confusion matrix. In the main diagonal, we can see the correct predicted values starting from top to bottom for each label. Regarding the “Evidence”, 353 observations were predicted as “No label”, 402 of the “No label” as “Evidence”, 233 of the “No label” were predicted as “Claim”.

Graphical user interface, chart

Description automatically generated

Figure 8 Confusion Matrix for predictions (Feed Forward)

Below we can see the predictions on the test data, where we have **77% accuracy**. Some useful equations in order to understand the following:

**Precision = true positives/ (true positives + false positives)**

**Recall = true positives/ (true positives + false negatives)**

**F1 score= 2\*(Precision\*Recall)/ (Precision + Recall)**

Table

Description automatically generated

Figure 9 Prediction Metrics (Feed Forward)

## CNN

The plot below shows the decrease of loss as the epochs pass. Validation ended on epoch 11 at 0.81 loss, while training ended on 0.3.

Chart, line chart

Description automatically generated

Figure 10 Validation loss for Training/Validation (CNN)

The following plot shows the accuracy of both the training and validation. Training peaked at 0.86 on epoch 11, while validation peaked at 0.72.

Chart, line chart

Description automatically generated

Figure 11 Accuracy for Training/Validation (CNN)

The following plot shows a confusion matrix as stated before. In the main diagonal are the correct predicted values starting from top to bottom. Regarding the “Evidence” label, 284 observations were predicted as “No label”, 511 of the “No label” as “Evidence”, 330 of the “No label” were predicted as “Claim”.

Chart, treemap chart

Description automatically generated

Figure 12 Confusion Matrix for predictions (CNN)

Below we can see the predictions on the test data, where we have **75% accuracy**.

Table, calendar

Description automatically generated

Figure 13 Prediction Metrics (CNN)

## FastText

The following plot shows a confusion matrix as stated before. In the main diagonal are the correct predicted values starting from top to bottom. Regarding the “Evidence” label, 585 observations where predicted as “Neither”, 252 of the “Neither” label as “Evidence”, while 86 of the “Neither” label were predicted as “Claim”. The best predicted label is “Neither” which makes sense since it was the most frequent label.

Chart

Description automatically generated

Figure 14 Confusion Matrix for predictions (fasttext - argument dataset)

The following plot shows a confusion matrix as stated before. In the main diagonal are the correct predicted values starting from top to bottom. The best predicted value is objective. Regarding the background label, 3 samples where predicted as conclusion while they were background labels, 13 samples where predicted as method while they were background labels, 24 samples were predicted as result while they are background labels etc.

Graphical user interface, application, Teams

Description automatically generated

Figure 15 Confusion Matrix for predictions (fasttext - structure dataset)

The final accuracy on the test data, regarding the **argument’s dataset is 77%,** while for the **structure dataset**, it is nearly **57.7%.**

## Transformer

In the following screenshots, we can see that after taking sample abstracts, we get the most simillar papers to it as well as their simillarity score.

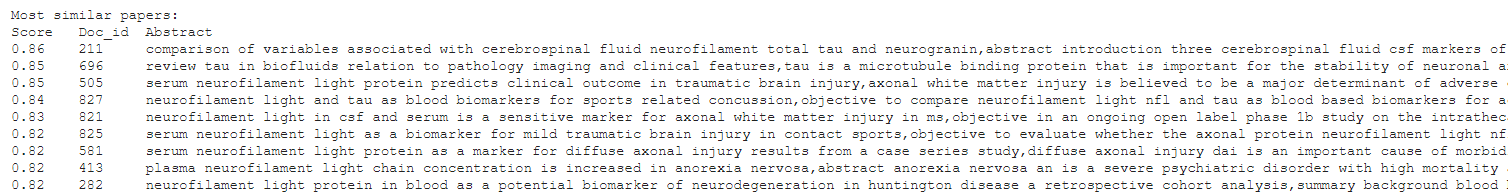


Figure 16 Similarity Score for the 2nd abstract

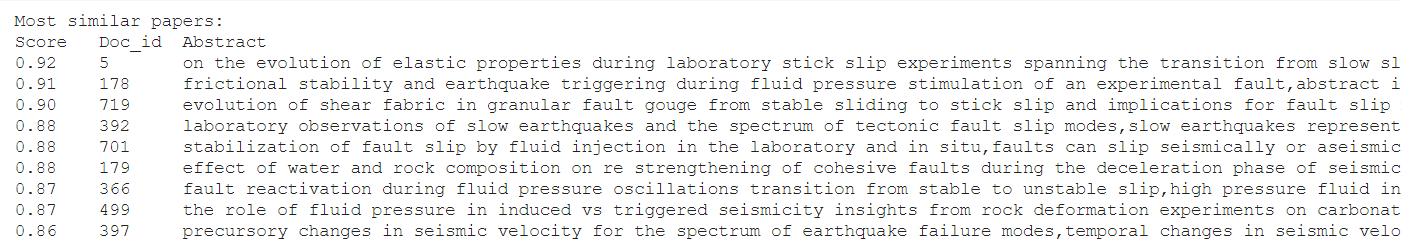
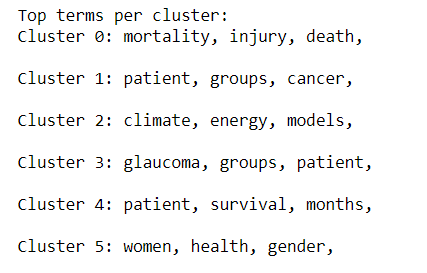


Figure 17 Similarity Score for the 3rd abstract

## K-Means Model

### Clustering only DE from the abstract

In the next caption, the top terms per cluster can be seen.



In the next plot, the cluster groups can be recognized by the different colors in the top right corner.

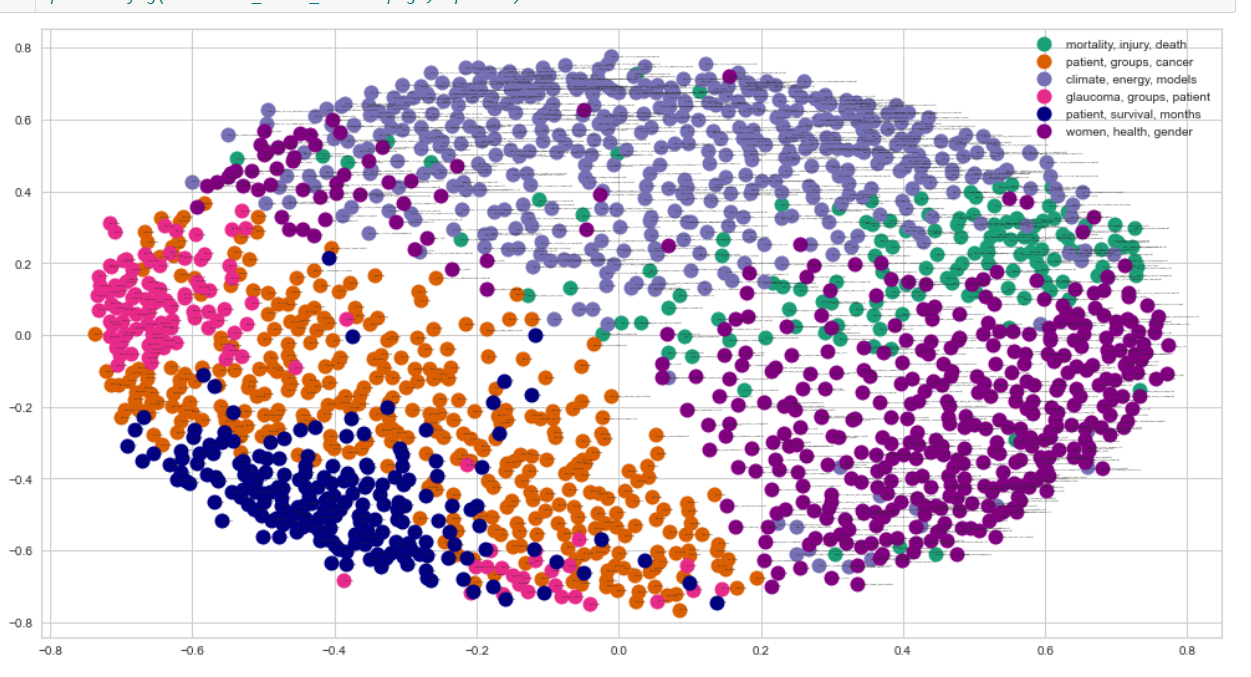
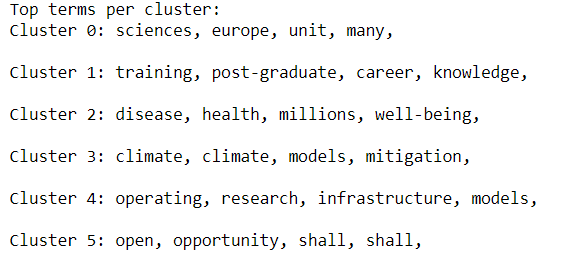


Figure 18 Clustering plot with 1 feature (Mean Silhouette Score: 0.037)

### Clustering with 2 features (sentences, EU Call)

In the next caption the top terms per cluster are shown, when using 2 features.



In the next plot, the cluster groups are shown more clearly than before.

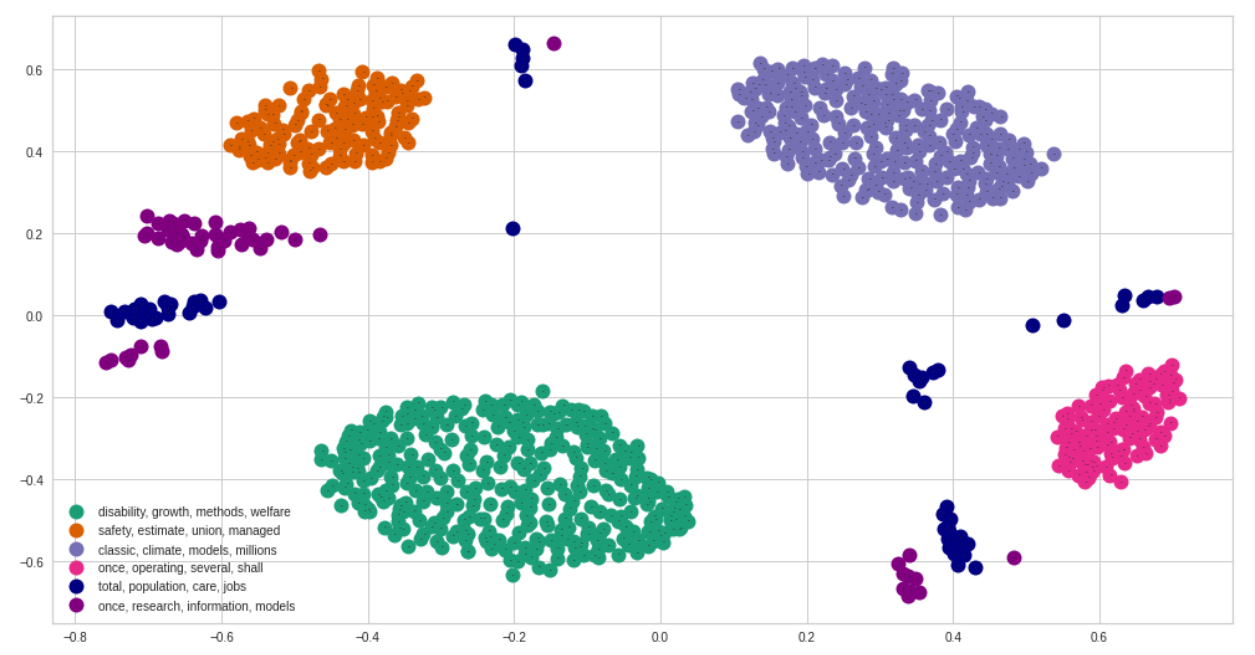
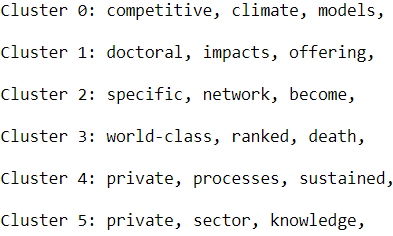


Figure 19 Clustering plot with 2 features (Mean Silhouette Score: 0.295)

### Clustering with 3 features (sentences, EU Call, argument embeddings)

The top terms per cluster are shown once again, but now when using 2 features and the arguments embeddings.



Finally, in the next plot the groups are shown but the metric (silhouette score) indicates that it is worse than the previous method.

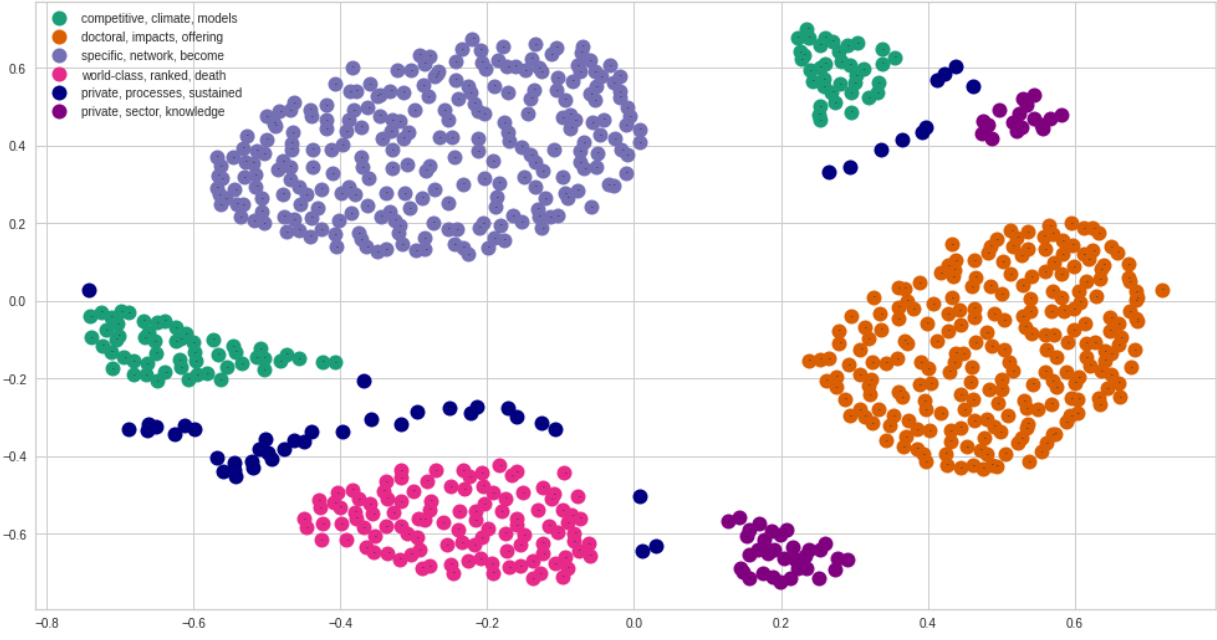


Figure 20 Clustering plot with 3 features (Mean Silhouette Score: 0.185)

### Conclusion

The best silhouette score is derived from clustering with 2 features (sentences and EU call) and the worst is clustering only with sentences.

# 6. Members/Roles

Our team comprised of 3 members: Athanasios-Marios Papadakis, George Pantelakis, Christos Porikis. All members took part in Annotation, data preprocessing, model construction and model evaluation as well as reporting.

Backgrounds:

**George** holds a bachelor’s degree in Computer Science from university of Piraeus.

**Thanos** graduated from Marketing & Communication at AUEB.

**Christos** is a graduate student of Economics from the University of Ioannina.

# 7. Time Plan

|  |  |
| --- | --- |
| **Date** | **Action** |
| 18 July - 31 July | Annotation of Abstracts |
| 1 Aug - 9 Aug | Data Engineering/ Base Line modeling |
| 10 Aug - 16 Aug | Initial Model Creation |
| 17 Aug – 18 Aug | Evaluation of model results |
| 19 Aug – 3 Sept | Re-evaluation of the results, parameter tuning and finalization of the models |
| 4 Sept – 5 Sept | Report Creation |

# 8. Bibliography

<https://en.wikipedia.org/wiki/Argument_mining>

<https://fasttext.cc/docs/en/supervised-tutorial.html>

[https://www.sbert.net/docs/pretrained\_models.html#scientific-publications](https://www.sbert.net/docs/pretrained_models.html%23scientific-publications)

<https://towardsdatascience.com/sarcasm-classification-using-fasttext-788ffbacb77b>

<https://towardsdatascience.com/clustering-documents-with-python-97314ad6a78d>

<http://brandonrose.org/clustering>

<https://arxiv.org/abs/1607.01759>

# 9. Other Sources

Haris Papageorgiou - Slides

Georgios Perakis – Lab sessions