UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

School of Design Engineering

Study on the contribution of Aerospace Engineering to achieve the United Nations Millennium Goals by Artificial Intelligence

Master's Thesis

Master's Degree in Aeronautical Engineering

AUTHOR: Sánchez Roncero, Alejandro

Tutor: Hoyas Calvo, Sergio

ACADEMIC YEAR: 2021/2022

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Study on the contribution of Aerospace Engineering to achieve the United Nations Millennium Goals by Artificial Intelligence

Author: Alejandro Sánchez Roncero

Supervisor: Sergio Hoyas

Final Master Thesis for the fulfilment of the Master's Degree in Aeronautical Engineering

Universitat Politècnica de València

September 15, 2022

Acknowledgements

I would like to thank Sergio Hoyas (Project supervisor) for giving me the opportunity to learn and develop my skills in such an interesting and emerging field as it is Artificial Intelligence. Having so many applications, I was able to apply it for Aeronautical Engineering. Also, I would like to thank to Jose Alberto Conejero and Ricardo Vinuesa for their help, insights and feedback during the development of the project. I will also thank Óscar Garibo for the preliminary version of the code. Finally, I will conclude by thanking them all for introducing me to the Sustainable Development Goals and being aware of all the work currently in progress by the United Nations to make of the world a better place to live in the future.

Abstract

Achieving the Millennium Goals of the United Nations (UN) is the primary goal of the 2030 Agenda. A critical step towards that objective is identifying if the scientific production is going in this way. Funders must do a manual recognition, impacting accuracy, scalability, and objectiveness. For this reason, we propose in this work an AI-based model for the automatic identification of the Sustainable Development Goals (SDGs) in scientific papers. This model is used to analyse more than eight thousand Aerospace-related scientific papers, comparing the impact that scientific papers with high and low citations have on the SDGs or whether this impact is positive or negative. The training database consists of manually extracted texts from the UN page. After pre-processing these texts, we train four different models: Non-Negative Matrix Factorization (NMF), Latent-Dirichlet Allocation (LDA), Top2Vec and BERTopic. The results obtained individually by each model are combined in a voting model. These models are validated with the database in Vinuesa et al. [15], obtaining a 97.2 % of accuracy with the training set and a 67.5 % with the validation set. The Aerospace dataset consists of ten thousand papers from 2017 to 2021 in Scopus, and it is shown that the major contribution to SDG7, SDG9 and SDG11. The databases used for training, validation, and analysis as well as the trained models are open source. The methodology, references, datasets, models and validation have already been published (Sánchez et al. [13]).

Resumen

Lograr los Objetivos del Milenio de las Naciones Unidas (UN) es el objetivo principal de la Agenda 2030. Un paso crítico hacia este objetivo es identificar si las publicaciones científicas se encuentran encaminadas. Para ello, los autores deben realizar un reconocimiento manual, lo que tiene impacto sobre la precisión, escalabilidad y objetividad de la evaluación. Por ello, proponemos en este trabajo un modelo basado en Inteligencia Artificial (AI) para la identificación automática de los Objetivos del Desarrollo Sostenible en publicaciones científicas. Este modelo se utiliza para analizar más de ocho mil publicaciones relacionadas con Ingeniería Aerospacial, comparando el impacto de aquellas publicaciones con un alto y bajo numero de citas sobre los objetivos o si este impacto es positivo o negativo. La base de datos de entrenamiento consiste en textos extraídos manualmente de la página de las Naciones Unidas. Tras el pre-procesamiento de los textos, entrenamos cuatro modelos diferentes: Non-Negative Matrix Factorization (NMF), Latent-Dirichlet Allocation (LDA), Top2Vec and BERTopic. Los resultados obtenidos individualmente por cada modelo se combinan en una votación. Estos son validados con la base de datos proporcionada en Vinuesa et al. [15], obteniendo un 97.2 % de precisión con los textos de entrenamiento y un 67.5 % con los de validación. La base de datos con los textos de ingeniería aerospacial cuenta con diez mil publicaciones desde 2017 hasta 2021 (Scopus) y se obtiene que estos contribuyen mayormente a los SDG7, SDG9 y SDG11. Las bases de datos utilizadas para entrenamiento, validación y análisis además de los modelos entrenados son open-source. La metodología, referencias, bases de datos, modelos y validación ya han sido publicados (Sánchez et al [13]).

Resum

Aconseguir els Objectius del Millenni de les Nacions Unides (UN) és l'objectiu principal de l'Agenda 2030. Un pas crític cap este objectiu és identificar si les publicacions científiques es troben encaminades. Per a això, els autors han de realitzar un reconeixement manual, la qual cosa té impacte sobre la precisió, escalabilidad i objectivitat de l'avaluació. Per això, proposem en este treball un model basat en Intelligència Artificial (AI) per a la identificació automàtica dels Objectius del Desenrotllament Sostenible en publicacions científiques. Este model s'utilitza per a analitzar més de huit mil publicacions relacionades amb Enginyeria Aerospacial, comparant l'impacte d'aquelles publicacions amb un alt i davall numere de cites sobre els objectius o si este impacte és positiu o negatiu. La base de dades d'entrenament consistix en textos extrets manualment de la pàgina de les Nacions Unides. Després del preprocessament dels textos, entrenem quatre models diferents: Non-Negative Matrix Factorization (NMF) , Latent-Dirichlet Allocation (LDA) , Top2Vec and BERTopic. Els resultats obtinguts individualment per cada model es combinen en una votació. Estos són validats amb la base de dades proporcionada en Vinuesa t'al. [16], obtenint un 97.2 % de precisió amb els textos d'entrenament i un 67.5 % amb els de validació. La base de dades amb els textos d'enginyeria aerospacial compta amb deu mil publicacions des de 2017 fins a 2021 (Scopues) i s'obté que estos contribuïxen majorment als SDG7, SDG9 i SDG11. Les bases de dades utilitzades per a entrenament, validació i anàlisi a més dels models entrenats són open-source. La metodologia, referències, bases de dades, models i validació ja han sigut publicats (Sánchez et al [13]).

Contents

Acronyms

AI Artificial Intelligence. 16, 18, 20

BERT Bidirectional Encoder Representations from Transformers. 42

BOW Bag of Words. 64

GHG Green House Gas. 15

GPU Graphics Processing Unit. 68

GSDR Global Sustainable Development Report. 22

HDBSCAN Hierarchical Density-Based Spatial Clustering of Applications with Noise. 37, 42

HLPF High Level Political Forum. 22

LDA Latent-Dirichlet Allocation. 2, 18, 19, 26, 32, 36

LSI Latent Semantic Index. 32

MDGs Millennium Development Goals. 13

ML Machine Learning. 23

NLP Natural Language Processing. 26, 64

NLTK Natural Language Toolkit. 27

NMF Non-Negative Matrix Factorization. 2, 18, 26, 36

pLSI probabilistic Latent Semantic Index. 32

S-BERT Sentence-Bert. 42

SDG Sustainable Development Goal. 9, 13, 14, 22, 24, 26

SDG1 No poverty. 14, 35, 56

SDG10 Reduced inequalities. 15, 47, 48

SDG11 Sustainable cities and communities. 15, 56, 61

SDG12 Responsible consumption and production. 15, 56

SDG13 Climate action. 15, 62

SDG14 Life below water. 15, 44

SDG15 Life on land. 15, 47, 48

SDG16 Peace, justice and strong institutions. 15

SDG17 Partnerships for the goals. 15

SDG2 Zero hunger. 14, 56

SDG3 Good health and well-being. 14, 31, 35, 44, 48, 61

SDG4 Quality education. 14, 35, 41, 47, 48

SDG5 Gender equality. 14, 55, 56

SDG6 Clean water and sanitation. 14, 31, 41, 56, 61

SDG7 Affordable and clean energy. 14, 31, 56, 61

SDG8 Decent work and economic growth. 14

SDG9 Industry, innovation and infrastructure. 14, 56

TF-IDF Term Frequency-Inverse Document Frequency. 26, 28

Top2Vec Topic to Vector. 36, 38

UMAP Uniform Manifold Approximation and Projection for Dimension Reduction. 38, 42

UN United Nations. 2, 13, 22, 24

List of Figures

List of Tables

Chapter 1

Introduction

In 2015 all the states members of the United Nations (UN) adopted the 2030 Agenda for Sustainable Development, which intends to promote peace and prosperity for people and the planet with a vision for the near future. In order to take that vision into a reality, the 2030 Agenda consists of 17 Sustainable Development Goals (Figure 1.1). They represent the actions that countries from all over the world (both developed and developing) should implement as global cooperation for the future of our planet. They are the result of decades of work by the UN countries together with the UN Department of Economic and Social Affairs[1]:

1. In June 1992 was adopted Agenda 21 for more than 178 countries. It defined a plan of action to build a global partnership for sustainable development to improve human lives and protect the environment.

2. In September 2000 at UN Headquarters in New York all the member states unanimously adopted the Millennium Declaration, which consisted of eight Millennium Development Goals (MDGs) focused on reducing extreme poverty by 2015.

3. At the World Summit on Sustainable Development (2002 in South Africa) it stated the Johannesburg Declaration on Sustainable Development and the Plant of Implementation, reaffirmed the global commitment to eradicate poverty.

4. In June 2012, Member states adopted the document "The Future We Want" which it was launched a process to develop a set of SDGs to build upon the MDGs and to establish the UN High-level Political Forum on Sustainable Development.

5. In 2013, the General Assembly set up a 30-member Open Working Group to develop a proposal on the SDGs.

6. In 2015 the negotiations of the 2030 Agenda for Sustainable Development finished, being lately adopted by the Member States in September.

7. Now, the annual High-level Political Forum on Sustainable Development serves as the central UN platform for the follow-up and review of the SDGs. The Division for Sustainable Development Goals (DSDG) in the United Nations Department of Economic and Social Affairs (UNDESA) provides substantive support and capacity-building for the SDGs and their related thematic issues, including water, energy, climate, oceans, urbanization, transport, science and technology, the Global Sustainable Development Report (GSDR), partnerships and Small Island Developing States. DSDG plays a key role in the evaluation of the UN system-wide implementation of the 2030 Agenda and advocacy and outreach activities relating to the SDGs. To make the 2030

[1] United Nations Web page

Agenda, a reality, broad ownership of the SDGs must translate into a strong commitment by all stakeholders to implement the global goals. DSDG aims to help facilitate this engagement.

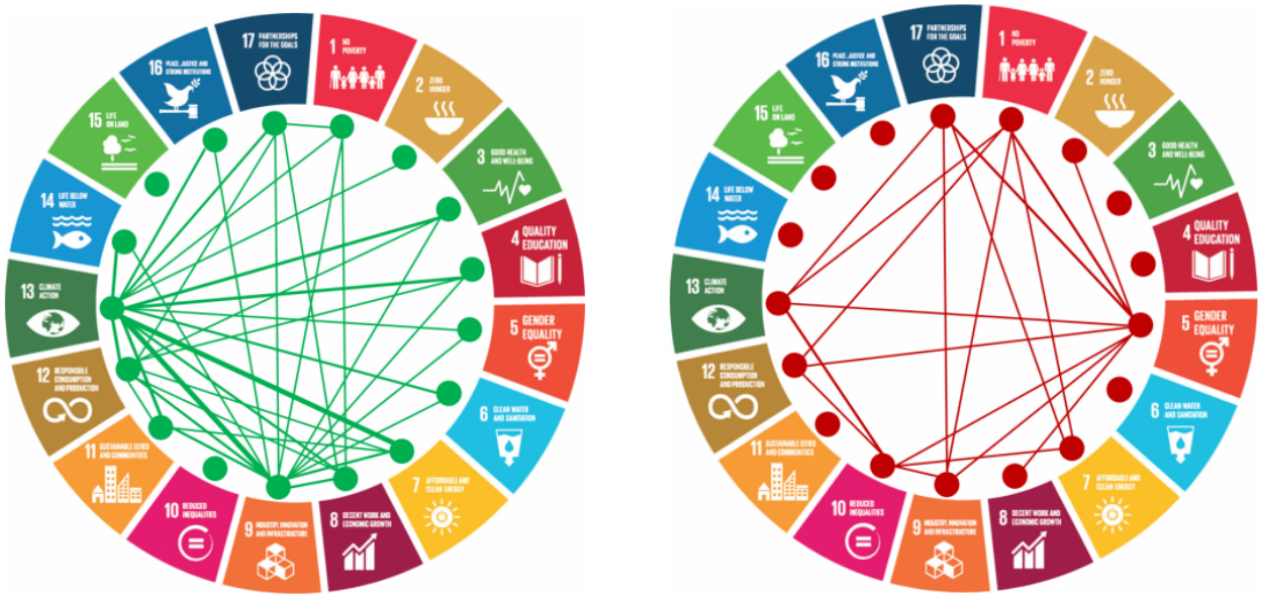
Figure 1.1: Poster of the Sustainable Development Goals

As depicted in Figure 1.1 the 17 SDGs are the following:

- No poverty (SDG1): end poverty in all its forms everywhere.
- Zero hunger (SDG2): end hunger, achieve food security and improved nutrition and promote sustainable agriculture.
- Good health and well-being (SDG3): ensure healthy lives and promote well-being for all at all ages.
- Quality education (SDG4): ensure inclusive and equitable quality education and promote lifelong learning opportunities for all.
- Gender equality (SDG5): achieve gender equality and empower all women and girls.
- Clean water and sanitation (SDG6): ensure availability and sustainable management of water and sanitation for all.
- Affordable and clean energy (SDG7): ensure access to affordable, reliable, sustainable and modern energy for all.
- Decent work and economic growth (SDG8): promote sustained, inclusive and sustainable economic growth, full and productive employment and decent work for all.
- Industry, innovation and infrastructure (SDG9): build resilient infrastructure, promote inclusive and sustainable industrialization and foster innovation.

- Reduced inequalities (SDG10): reduce inequality within and among countries.
- Sustainable cities and communities (SDG11): make cities and human settlements inclusive, safe, resilient and sustainable.
- Responsible consumption and production (SDG12): ensure sustainable consumption and production patterns.
- Climate action (SDG13): take urgent action to combat climate change and its impacts.
- Life below water (SDG14): conserve and sustainably use the oceans, seas and marine resources for sustainable development.
- Life on land (SDG15): protect, restore and promote sustainable use of terrestrial ecosystems, sustainably manage forests, combat desertification, and halt and reverse land degradation and halt biodiversity loss.
- Peace, justice and strong institutions (SDG16): promote peaceful and inclusive societies for sustainable development, provide access to justice for all and build effective, accountable and inclusive institutions at all levels.
- Partnerships for the goals (SDG17): strengthen the means of implementation and revitalize the Global Partnership for Sustainable Development.

The Sustainable Development Goals represent a complex holistic challenge due to their internal connection (i.e they are not independent). This is the key to understanding and unlocking their full potential, otherwise, the efforts put into the development of some of them could be made at the expense of others. In Kostetckaia et al. [7] it is studied the interlinkages as well as the trade-offs and synergies that arise from their internal relationships. The Figures 1.2a and 1.2b show the positive (synergies) and negative (trade-offs) interconnections among the SDGs respectively.



(a) Based on shares of positive correlations (b) Based on shares of negative correlations

Figure 1.2: Visualisation of SDG interlinkages between the goals

The synergies could be explained as follows: if more emphasis is put on clean and renewable energy then the Green House Gas (GHG) emissions will be reduced and the cities and communities will become more sustainable. In the same way, if the quality of the work is enhanced and the country is experiencing economic growth then there will be less poverty and the industry infrastructure will

also experiment benefits. On the contrary, they also exist trade-offs: if to increase the quality of the work and experience economic growth the country uses vast amounts of resources and energy, then there will be a trade-off between SDG7, SDG8 and SDG12. This is a vital point to understand since the actions to implement should consider both the positive and negative impacts. Relation to the current work also has a profound implication because all SDGs are interconnected. This means that not all models will work properly when having to identify the SDGs in a text since some classification algorithms assume internally to be associated with a unique topic.

In the past years, the interest and exploration in the field of Artificial Intelligence (AI) have increased notoriously. It allows humans to solve problems that could not be solved with previous tools. In the same manner, AI could be used to help the government and the world to achieve the SDGs as it is the primary goal of the 2030 Agenda. A critical step towards that objective is identifying if the scientific production is going in this way. When associating a specific work with the SDGs, funders must do a manual recognition, which impacts its accuracy, scalability and objectiveness. For this reason, it is proposed in this work an AI-based model for the automatic classification of scientific papers based on their impacts on the SDGs. The objectives of the project are described in Section 1.1, however, a brief picture of the project is that the model receives as input a specific text (usually abstracts or other types of short texts) and outputs the score of association that each text has with each of the SDGs. In this work, it will be analysed whether the Aeronautical work having been carried out in recent years may be in line and have a positive impact on achieving the 2030 Agenda or not. For that purpose, publications related to aeronautical engineering from 2017 to 2020 are downloaded from Scopus. This database together with those used for training the model and doing the validation is described in Chapter 2. The structure of the project as well the description of the models used to build it can be seen in Section 1.2 and Chapter 3 respectively.

1.1 Objectives

The main objectives of the current work are the following:

1. Implement a model able to identify the SDGs associated with a text. As mentioned before, there are 17 SDGs in total, however, this work will not consider SDG17. The reason is that it is a mixture of the other SDGs, so it will be left as future work.
2. Prepare and publish both training and validation datasets so that future work in this field is enhanced.
3. Validate the trained model with a previously expert-classified dataset.
4. Analyse the impact that the length of the texts has on the model accuracy.
5. Analyse the impact that the Aeronautical Engineering field has on achieving the SDGs based on publications made in recent years.
6. Compare the impact of publications with a low and high number of citations.

Under the main objectives and following the main idea of the work, they have also defined the following requirements:

- The model should be able to identify the SDGs correctly even if the text is small (i.e. less than 200 words). Usually, they will be used in abstracts or other kinds of short texts so the model should be prepared for that.

- The required computational power and energy used for training the model should be minimized. In line with the SDGs, training the model and using it should use as few resources as possible.
- The model should be able to run fast (i.e. less than 5 seconds per text) and with low computational resources. The same rationale of the previous requirement applies here.
- The model should be robust.
- The model should be scalable for future improvements. The model should be able to incorporate more models easily and without affecting its normal operation (i.e. that the model may be improved not only training better the already trained models).
- The model and datasets should be easy to use and open-source.

1.2 Project description

All the models, datasets and results are obtained using the programming language Python. The project is divided into the following directories:

- main directory: it contains the files for the configuration, data and tools that can be used by the other files (conf.py, data.py and tools.py respectively).
- codes directory: it contains a file per model and for the votation mechanism.
- datasets directory: it contains the dataset in both CSV and JSON format.
- test directory: it contains the files used to test each model and the votation mechanism separately. In this way, the models can be updated separately and tested without affecting the other models.
- analysis directory: it contains the files used to analyse the datasets, the results obtained from the models and the Aerospace dataset.

The code of the model can be seen in Appendix A. It is the same as the other models, so all of them share the same interface. In the case of the other files, the codes are self-explaining too. There are included the following examples:

- Main tools used for the project (A.1).
- File with the required methods for loading all the datasets (A.2).
- Models of the NMF, LDA, Top2Vec and BERTopic (A.3, A.4, A.5 and A.6 respectively).
- Test codes for the NMF model (A.7, being the same for the other models) and the votation mechanism (A.8).
- Code for the analysis of the aerospace dataset (A.9).

All the datasets are stored in JSON format, which allows for the storage of all the required files without any restriction on their length. The project is public to visit and download; GitHub project.

Chapter 2

Related work & Datasets

Nowadays, the Internet connects billions of people in the world and allows the exchange of information in an unprecedented way. For example, people can express freely their opinions about certain topics. In this way, a new problem arises: how could it be possible to understand what people are talking about and their feelings without having to do a manual reading of all those texts? Traditionally this problem could not be solved, since no technique allowed the computer to understand this internal and complex human logic. To tackle this problem, it arose the concept of "Topic modelling".

Topic modelling is an unsupervised learning technique that allows computer users to analyse, discover and gain insight into a specific set of texts. This concept is very powerful since it allows us to understand internal and hidden patterns in the data in an unsupervised manner (i.e. the user doesn't introduce any parameter). It extracts the main aspects of the texts and based on them they are categorized. In resume, the steps they followed are the following:

- Collecting the texts. The set of texts is called "corpus". In this work, how they were collected and some information about them is given in Section 2. The corpus is unstructured (i.e. the order of the texts is meaningless).
- Discovering the hidden semantic patterns in the texts, and the topics. Each topic is a collection of words. If the model is good enough, then the topics will be able to represent the global information of the corpus with much fewer resources.

In recent years, the amount of work and publications in Artificial Intelligence (AI) have seen an exponential increase; Figure 2.1 shows an increase from 10000 in 2000 to more than 120000 in 2019. Due to this popularity, each year new algorithms for topic modelling are discovered and published. Some of the traditional techniques largely used and accepted for topic modelling are Non-Negative Matrix Factorization (NMF) and Latent-Dirichlet Allocation (LDA) (probabilistic and non-probabilistic respectively). Some use scenarios of these models are spam detection and search query. These models rely entirely on the information that is passed to them, no previous training is required; they are described in Sections 3.1 and 3.2 respectively. On the contrary, there are other more complex models such as Top2Vec and BERTopic (Sections 3.3 and 3.4).

In Egger and You [3] it is conducted a comparison of the performance that NMF, LDA, Top2Vec and BERTopic have for topic modelling, analysing a big dataset of unstructured and short texts from Twitter posts. Due to their large differences, they compared NMF with LDA and Top2Vec with BERTopic separately. The NMF model only detects some of the topics and issues in the dataset, while LDA identifies almost all of them. However, LDA extracts more universal and irrelevant topics than NMF. In both cases, the topic extracted does not allow for an in-depth understanding of the

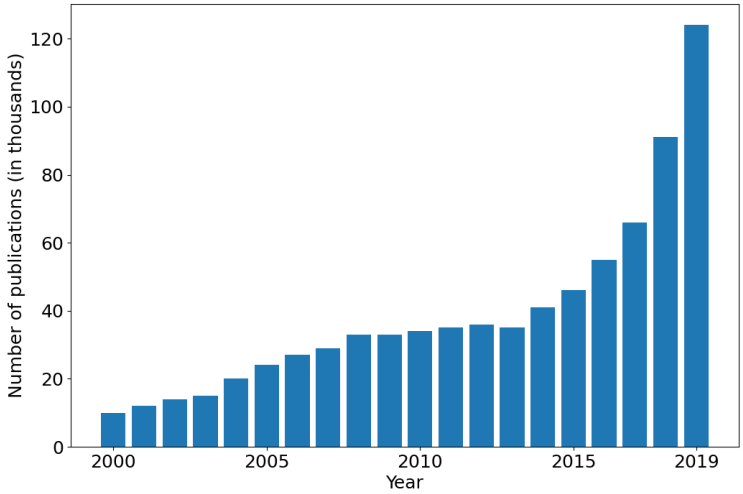


Figure 2.1: Number of peer-reviewed AI publications from 2000 to 2019 [16]

phenomenon. With Top2Vec and BERTopic they extract more accurate and meaningful topics, being more informative. Both models output logic and coherent topics, which are easy to understand and interpret. In conclusion, NMF and BERTopic are identified as better algorithms for topic modelling in short and very unstructured texts.

One of the objectives listed in Section 1.1 is the comparison of the results obtained when using full texts and only abstracts for topic modelling. In Nazemi et al. [12] they try to answer the following question: do abstracts of scientific publications provide a similar analysis capability compared to their corresponding full-texts from the Visual Analytics viewpoint? To answer it, the LSI and LDA methods are used as topic modelling and their results are internally compared. It is also compared whether lemmatizing [1] is positive or not. After analysing more than 2600 documents, they conclude that in general the main technological aspects of a paper are included in the abstracts, so they illustrate the main ambitions of the publication. Lemmatizing the vocabulary leads to a lower coherence value of the models. It was argued that in some topics the words were very similar when skipping lemmatization so it led to a higher coherence value. Statistically, there was no difference in the coherence of the results with and without lemmatizing when using only abstracts. The maximum obtained coherence was using full texts instead of abstracts.

In Marcelo La Fleur [9] they conduct a simplified version of this work, explaining a proof-of-concept machine learning model to measure how similar a given publication is to each of the 17 SDGs. They compute the SDGs' scores for some limited sections of DESA publications and provide some analytics. They use the Latent-Dirichlet Allocation (LDA) model as the main topic modelling technique, allowing them to analyse the texts at scale with objectivity and identify patterns across publications. Proceeding in the same way that they do, this data-driven based model can help decision-makers to identify how to maximize the impact of publications and how to improve the alignment between DESA's work and SDG implementation. The main advantage of using LDA is that it is assuming that documents are a combination of all the topics in the corpus, which makes sense since texts are rarely about a single subject. In this case, they use only 17 unique and balanced texts to train the LDA algorithm with 18 topics (the extra topic acts as a filter to remove the common words). To validate the model they follow two steps: first, each topic should be able to identify specifically one of the SDGs. Secondly, the words associated with each of the topics should be coherent to the SDG that they are linked. They analyse the association of the publications with each SDG and the

[1] Lemmatisation is the process in linguistics consisting of grouping together the inflected forms of words so they can be analysed as a single item (the words' lemma).

evolution that they experienced over the years.

In Hajikhani and Suominen [5] they compile the definitions of the SDGs to train a machine learning model to automate the detection of SDGs in texts such as patents. The main idea was to identify the extent to which SDGs were being addressed in patents as well as identify the SDGs' interrelations. Scientific publications from the last decade were downloaded. Then they were trained in several classification algorithms and their performance was evaluated. They compare the Naive Bayes Classifier, Linear Support Vector Machine, Logistic Regression, Word2Vec, Doc2Vec and multi-layer perceptron. For most of the SDGs, the models deliver an accuracy above 60 %, being the Word2vec model the one with the highest overall accuracy. Only 11 out of the 17 SDGs were correctly identified, having experienced issues with the other ones.

Finally, in Vinuesa et al. [15] they analyse the impact of the progress and evolution of Artificial Intelligence (AI) on achieving the Sustainable Development Goals. To date no study has been conducted on assessing the potential impacts of AI on the SDGs, however, it is expected to affect several areas such as productivity, equality or environmental outcomes (both in the short and long term). For this reason, it is discussed the implications of how AI can either enable or inhibit the delivery of the 17 goals and 169 targets, following a consensus-based expert elicitation process and based on previous studies that map SDGs interlinkages.

In Figure 2.2 it can be seen a summary of the positive and negative impacts of AI on the various SDGs. The study reveals that AI may act as an enabler on 134 targets (79 %) through technological improvement while 59 targets (35 % across all SDGs) may experience a negative impact from the development of AI. It should be remarked that in this study the SDGs are divided into three categories: Society, Economy and Environment.

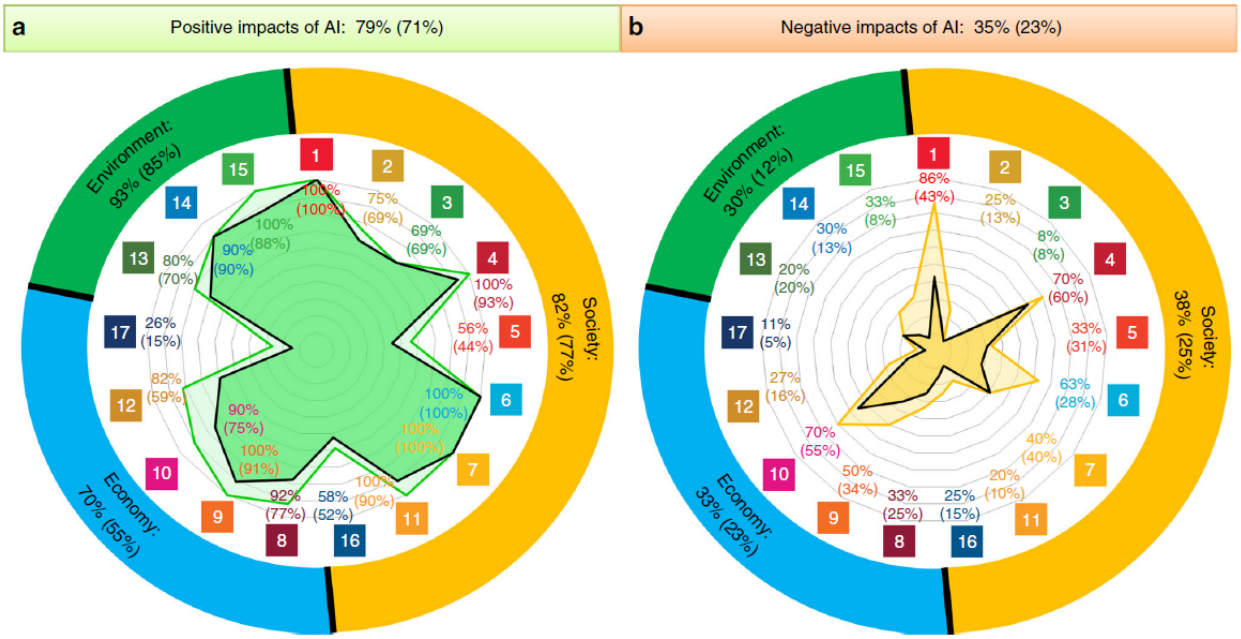


Figure 2.2: Summary of the positive and negative impact of AI on the various SDGs. Documented evidence of the potential of AI acting as (a) an enabler or (b) an inhibitor on each of the SDGs. The numbers inside the coloured squares represent each of the SDGs. The percentages on the top indicate the proportion of all targets potentially affected by AI and the ones in the inner circle of the figure correspond to proportions within each SDG. The results corresponding to the three main groups, namely Society, Economy, and Environment, are also shown in the outer circle of the figure. The results obtained when the type of evidence is taken into account are shown by the inner shaded area and the values in brackets [15].

Regarding the societal outcomes, 67 targets (82 %) could benefit from the development of AI. For example, it could support the provision of food, health, and energy or reduce carbon emissions by promoting a better and more smart circular economy. In this case, a much lower 25% could be impacted negatively, p.e., the progress of AI could require massive computational resources which could increase the energy requirements and carbon footprint. AI can be used as a catalyst to achieve the 2030 Agenda, however, it may also trigger inequalities among countries. Also, AI technology could lead to hate towards minorities, biased elections, social control or nationalism since it is mostly based on the needs and values of the nations in which it is being developed. Finally, AI is not evenly distributed; AI-enhanced agricultural equipment may only be accessible to big farms so it could lead to inequalities and gaps concerning small farms.

The progress of AI may also have a positive impact on the achievement of the SDGs in the Economy group, being classified 42 targets (70%) as having a positive impact. It may also have a negative impact (20 targets, 33%) since a large dependence on technological resources (required for a data-driven economy) could introduce great inequalities and gaps between regions. Moreover, the inequalities could increase even inside countries, since AI could be replacing old jobs with ones where technology would disproportionately reward the educated people.

Finally, it is analysed the environment group is composed of SDGs 13, 14 and 15. In this case, the impact of AI may be almost entirely positive, consisting of 25 positive impacts (93 %) and only 30% as negative. Figure 2.3 it is shown a detailed assessment of the impact of AI on this group. Some reasons for having such a positive impact are that AI will allow scientists to better understand climate change, it will support low-carbon and high-efficiency energy systems or it will be able to combat desertification and restore degraded land and soil.

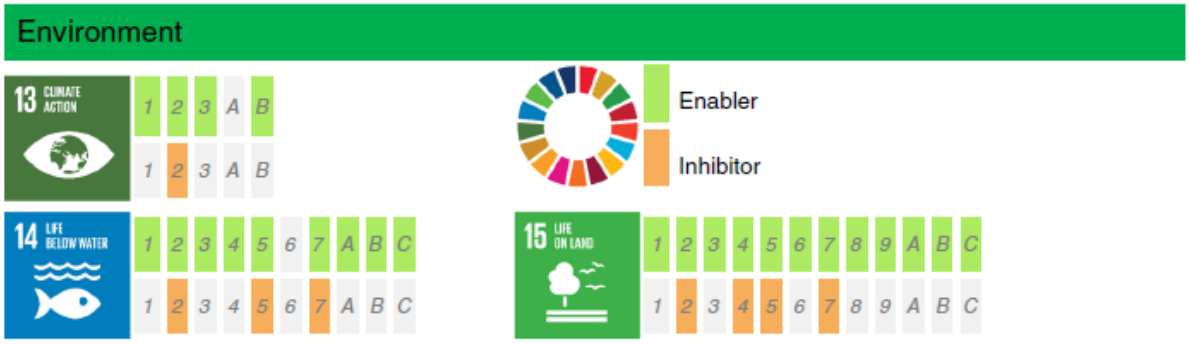


Figure 2.3: Detailed assessment of the impact of AI on the SDGs within the Environment group. Documented evidence of the positive or negative impact of AI on the achievement of each of the targets from SDGs 13, 14, and 15 [15].

It is also outlined the current research gaps on the role and development of AI. For example, it can be expected some bias in the AI research community and industry toward publishing positive results, or some aspects of AI may require long-term studies (not conducted yet). All the data supporting the findings of the study are available. Since this dataset has more than 100 related papers already labelled with SDGs via a manual elicitation-based consensus it will be used as a validation dataset of the trained models (Section 2).

Training dataset

The basic training texts are extracted from the United Nations web page. It is directly related to objectives 1 and 2 (Section 1.1) which briefly consist in implementing a model able to identify the SDGs in a text and prepare and publish a training dataset. All the texts are extracted manually from the following sections:

- SDGs description: each SDG is described in the 2030 Agenda and Key topics section. These texts are labelled according to their section. In general, they are associated with a unique SDG. They describe each SDG, as well as its targets and indicators.
- Implementation progress: every year, the UN Secretary-General presents an annual SDG Progress report, which is developed in cooperation with the UN System, and based on the global indicator framework and data produced by national statistical systems and information collected at the regional level. The reports range from 2016 to 2021. They also have clear distinct sections for each SDG.
- Global Sustainable Development Report (GSDR): it is a United Nations publication aiming to strengthen the science-policy interface at the High Level Political Forum (HLPF) on Sustainable Development, which replaced the Commission on Sustainable Development after Rio+20 as the main United Nations platform providing political leadership and guidance on sustainable development issues at the international level. It also gives a role in the follow-up and review of the new Agenda for each of the countries that adopted the 2030 Agenda. The GSDRs used were published in 2015, 2016 and 2019.
- Publications and Acceleration actions: these are projects published by the United Nations that are intimately related to the progress, development and implementation of the Sustainable Development Goals. They have proved to be very useful because they replace the general vocabulary provided by the others sections with one more specific and topic-related. Also, some of the projects are related to more than 1 SDG, so it makes the SDGs-interconnection task easier for the models.

Once all the information is downloaded, then it is split into homogeneous texts. As it is mentioned in Section 1.1, the texts to analyse will generally be short (i.e. abstracts) so the training texts should also be relatively small. Figure 2.4 shows the number of files classified by the range of words number that they contain. As it may be observed, the large majority of the texts fall in the category of 100 to 200 words. There are only a few texts with 300-400 words and less than 100 words.

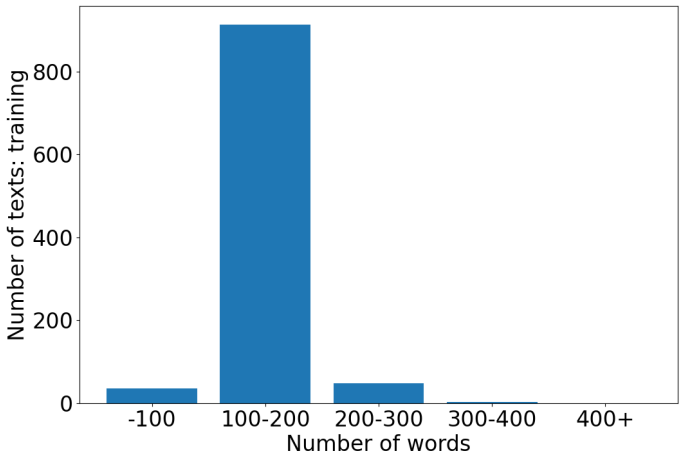
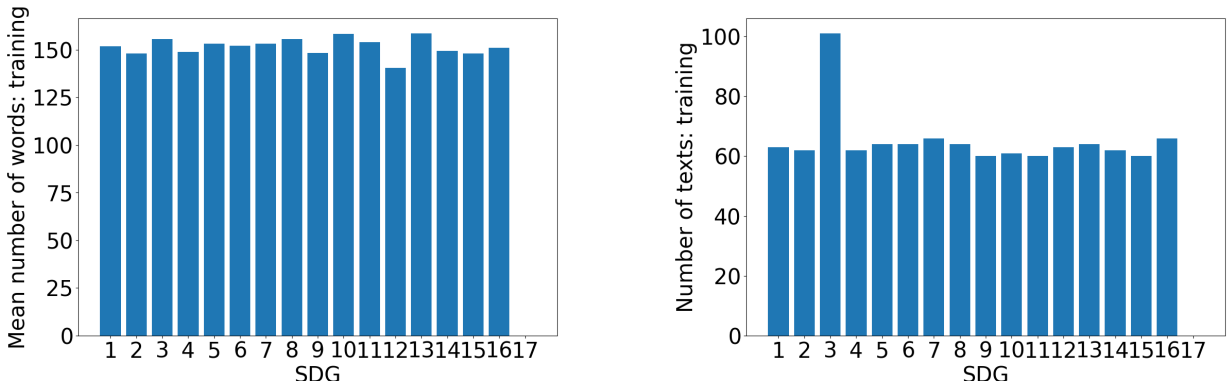


Figure 2.4: Number of training texts per number of words-range

The average number of words per SDG as well as the number of texts associated with each SDG are homogenized. In Figure 2.5a it can be seen that the mean number of words is roughly the same in all the SDGs (around 150 words). This is a very important restriction because some of the models will learn based only on the training texts (without prior information) so if some of the SDGs have more associated texts or the number of words is higher, then there will exist a bias towards them. Also, Figure 2.5b shows the number of texts associated with each SDG. In general, the number of training texts per SDG is 60, and all the SDGs have almost the same distribution. The only outlier is

SDG3. The reason is to show that some of the models (Top2Vec and BERTopic, which are explained in Section 3) behave better and the results they obtain are improved when more information is present, and do not necessarily follow a uniform distribution. In this case, some term-specific texts related to health care are introduced. These texts are extracted from MeDAL. It is an open-source medical dataset designed for Natural Language Understanding. In total, the training dataset contains 1000 individual texts.



(a) Mean number of words

(b) Number of texts

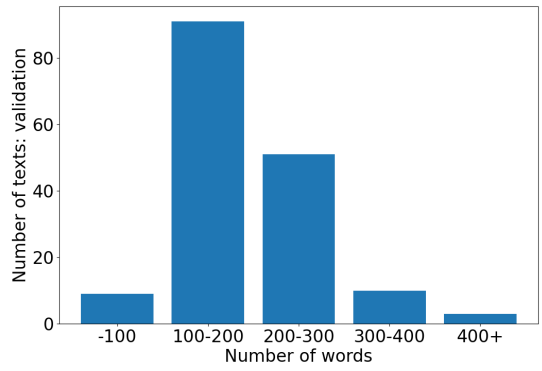
Figure 2.5: Information of the training files per SDG

All the texts are stored and uploaded together in the file "dataset.json". As an example, it is included here a training text labelled with the SDG1: " goal 1: end poverty in all its forms everywhere. more than 700 million people, or 10 % of the world population, still live in extreme poverty and are struggling to fulfil the most basic needs like health, education, and access to water and sanitation, to name a few. the majority of people live on less than $1. 90 a day live in sub-Saharan Africa. worldwide, the poverty rate in rural areas is 17. 2 per cent more than three times higher than in urban areas. having a job does not guarantee a decent living. 8 per cent of employed workers and their families worldwide lived in extreme poverty in 2018. poverty affects children disproportionately. one out of five children live in extreme poverty. ensuring social protection for all children and other vulnerable groups is critical to reducing poverty. ".

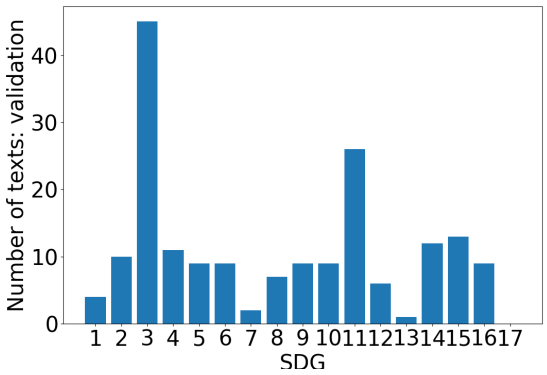
Validation dataset

In Machine Learning (ML) applications, the training and validation datasets should be differentiated (i.e. the texts used for training should never be used for any validation purposes). To validate the results, as mentioned previously it is used the dataset in Vinuesa et al. [15]. In this publication, they present and discuss the implications of how AI can either enable or inhibit the delivery of all 17 goals and 169 targets recognized in the 2030 Agenda for Sustainable Development. For that purpose, they use a consensus-based expert elicitation process to associate some publications with some targets of the SDGs. Based on their dataset, the abstracts and full papers are manually downloaded. The reason for the division is due to objective 4 presented in Section 1.1: it should be analysed the impact that the length of the texts has on the model accuracy. It is also related to objectives 2 and 3 (prepare and publish a validation and validate the model with an expect-classified dataset). In this case, it is compared the differences (if exist) between using only the abstracts and using more information from the other sections of the papers. Otherwise, more information should then be provided to evaluate the association of the texts with the SDGs. In total, the datasets consist of 164 abstracts and 186 full papers. Since the main objective is to analyse the abstracts, it is shown in Figures 2.6a, 2.6b the number of texts per words-range and the number of texts associated with each SDG respectively. It can be seen that the data is less uniform than in the case of the training dataset since the texts can not be modified. The greatest percentile of the texts has between 100 and 300 words, which is in line

with the texts used for training. The SDG with the highest frequency is SDG3, and there are other SDGs (p.e. SDG13) that are less representative.



(a) Number of texts per words range



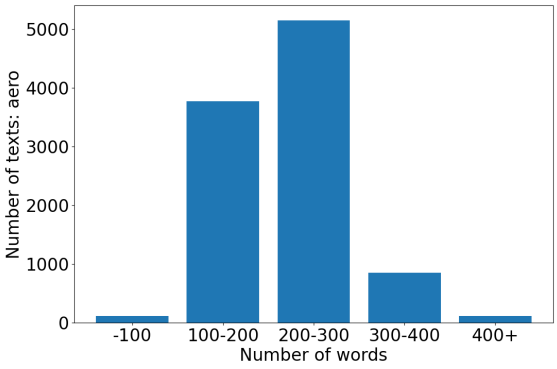
(b) Number of texts associated with each SDG

Figure 2.6: Information of the validation files

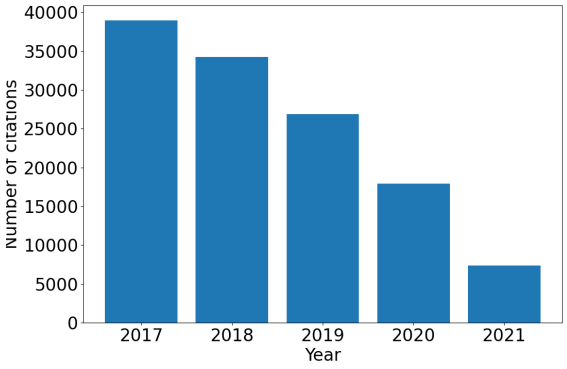
Analysing dataset

This section it is presented the dataset related to the final analysis of the current work. It consists of publications related to aerospace engineering since objective 5 (Section 1.1) consists in analysing the implication that this field is having about achieving the Sustainable Development Goals of the United Nations (i.e. if current work is aligned with the 2030 Agenda). It is also related to objective 6 (compare the impact of publications with a low and high number of citations). For this purpose they downloaded 2000 papers (per year) from 2017 to 2021 (both included) from Scopus[2]; in total, they analysed 10000 papers. They are ordered based on the number of citations that they have.

In Figure 2.7a it is is shown the number of papers per range of several words. Most of the texts have between 100 and 300 words, so again it aligns with the length of the texts used for training the models. Only a few of them have less than 100 words or more than 400. Figure 2.7b shows the number of citations per year (adding up the citations of all the papers in that year). As it is natural, the number of citations decreases with the year, since the papers are more recent. In 2017 the number of citations was almost 40 k, while in 2021 it is 7.5 k.



(a) Number of papers per number of words-range



(b) Total number of citations per year

Figure 2.7: Information of the aerospace files

[2]Scopues is the largest abstract and citation database of peer-reviewed literature, is widely used to create datasets for systematic reviews of research [5].

In total there are around 90 different countries represented in the dataset (although the number of papers and citations is not equally distributed): Singapore, United States, Netherlands, China, Belgium, United Kingdom, Iran, Germany, Romania, Australia, Italy, Canada, India, Poland, Ireland, Thailand, Switzerland, France, South Korea, Japan, Hungary, Norway, Hong Kong, Pakistan, Spain, Kazakhstan, Serbia, Lithuania, Brazil, Russian Federation, Sweden, Algeria, Austria, Argentina, Israel, Portugal, Turkey, Denmark, Macau, Indonesia, Croatia, Greece, Taiwan, Egypt, Saudi Arabia, New Zealand, Mexico, Finland, Malaysia, Estonia, Ethiopia, Czech Republic, PA, Viet Nam, Iraq, South Africa, Malta, Chile, Slovenia, Lebanon, Bulgaria, Morocco, Tunisia, Sri Lanka, Azerbaijan, Jordan, United Arab Emirates, Nepal, Luxembourg, Kuwait, Ukraine, Colombia, Uzbekistan, Bangladesh, Oman, Cyprus, Botswana, Georgia, Qatar, Bosnia and Herzegovina, Uruguay, Ghana, Ecuador, Inria, Nigeria, Libyan Arab Jamahiriya, Philippines, Latvia, Maroc, Venezuela.

Chapter 3

Models

This chapter provides the basic rationale and mathematics behind each Natural Language Processing model that is used for the Sustainable Development Goals identification. As mentioned in Section 1, four different models are trained: Non-Negative Matrix Factorization (NMF) (Section 3.1), Latent-Dirichlet Allocation (LDA) (Section 3.2), Top2Vec (Section 3.3) and BERTopic (Section 3.4). All of them are built on top of the same base python class, so they share the same inputs/outputs methods for the text analysis. Each section describes its respective properties and mathematics. However, the output of all of them is the same: the coherence or probability of association that a text has with each SDG. Due to their inherently different nature (it will be explained in Section 5), the information that each model extracts from a text is different, in other words, their functionalities are complementary. To take advantage of this fact, it is introduced a voting mechanism. It takes as inputs the scores of each model for each text. Collecting all the information, then it decides which of those identified SDGs have enough confidence to assume that the text relates to them.

3.1 Non-Negative Matrix Factorization

The Non-Negative Matrix Factorization model approximates a non-negative matrix by the product of two low-rank non-negative matrices; it is a linear-algebraic optimization algorithm [14] for dimension reduction and factor analysis method [8]. As it has been mentioned, the difference that this method has with other low-rank approximations of matrices is that they only have non-negative elements. This is important for the analysis of texts since this application will never have any negative value. This fact will lead to physically natural interpretations of the results. NMF is an unsupervised technique since there is no labelling of topics that the model will be trained on. The topics are then automatically discovered by the algorithm. It gives semantically meaningful results (information clustering) that are easily interpretable. For this reason, NMF has been widely used for the analysis of document data, specifically for topic modelling.

The steps that the model takes to perform the computations and which will be explained in the following sections are:

1. Pre-process the input text.
2. Transform the text into the Term Frequency-Inverse Document Frequency representation.
3. Ensemble the [document x words] matrix A.
4. Reduce the matrix dimensionality.

5. Identify the topics in the resulting matrices.
6. Ensemble the association matrix with the labelled SDGs.
7. Validate the model with the training and validation datasets.

Text Pre-Processing

The texts that are passed to the NMF model require some preprocessing. The model only reduces the dimensionality of the information that is passed, and it does not rely on any other source. For this reason, common words such as "when", "for" or "much" should be eliminated, since they are not meant for the study. The pre-processing stage comprehends the next steps:

1. Convert the text to lowercase. This applies to all the texts that are used.
2. Strip tags and punctuation characters (["!", "?", "."]).
3. Strip non-alphanumeric characters ([;]).
4. Strip multiple white spaces.
5. Remove words with a length lower than 2 characters.
6. Lemmatize the words. This is a common step in all the processes related to language analysis. Briefly, it consists of the grouping of different forms of the same word. For example, the words "leafs" and "leaves" would be converted into "leave". It avoids the presence of multiple forms of the same word in the topics since they do not give any extra information.
7. Remove stop words. This includes the dataset provided by gensim as well as a manually-made dataset (*stop_words.txt*, located in the *ref/* folder).

The functions used for preprocessing belong to the gensim library while the Lemmatizer class does to the Natural Language Toolkit one (NLTK). The output of these steps can be visualized in the following example:

- Input text: "goal 1: end poverty in all its forms everywhere. more than 700 million people, or 10% of the world population, still live in extreme poverty and are struggling to fulfil the most basic needs like health, education, and access to water and sanitation, to name a few. the majority of people live on less than $1. 90 a day live in sub-Saharan Africa. worldwide, the poverty rate in rural areas is 17. 2 per cent more than three times higher than in urban areas. having a job does not guarantee a decent living. 8 per cent of employed workers and their families worldwide lived in extreme poverty in 2018. poverty affects children disproportionately. one out of five children live in extreme poverty. ensuring social protection for all children and other vulnerable groups is critical to reducing poverty. "
- Output text: "goal end poverty world population live extreme poverty struggling to fulfil basic need like health education access water sanitation majority living le live sub-Saharan Africa worldwide poverty rate rural area time higher urban area having job doe guarantee decent living fact employed worker family worldwide lived extreme poverty affect child live extreme poverty ensuring social protection child vulnerable group critical reduce poverty"

As it may be deduced, the output is much clearer and has a higher focus on the topic that it is talking about, having deleted those words with no value.

Convert input text to TF-IDF representation

The Term Frequency-Inverse Document Frequency (TF-IDF) is a matrix that represents the importance that each word has to each document inside a corpus (collection of documents). This is one of the most popular schemes used for term-weighting purposes. Its value increases linearly with the frequency of the word and it is scaled with the total number of words, so it reduces the problems with those documents with a large number of words. Also, it assumes that words that appear only in a few documents will be more representative and have more information than more common. For that reason, it is scaled with the logarithmic fraction between the total number of documents and the document frequency of each term (i.e. the number of documents in which the term appears). In this way, if the word appears in all documents, the term will have a tf-IDF value of 0 (p.e. this is useful for stop-words filtering). The corresponding equation is shown in Equation 3.1.

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \cdot \left(\log \left(\frac{N}{df(t)} \right) + 1 \right), \tag{3.1}$$

where tf-IDF(t,d) is the term frequency-inverse document frequency of a term t, $f_{t,d}$ is the frequency of a term t in document d, $\sum_{t' \in d} f_{t',d}$ is the total number of terms inside a document d, N is the total number of documents, and df(t) is the document frequency of the term t.

It is used the TfidfTransformer from scikit-learn. First, the *CountVectorizer* convert the corpus of documents into a matrix of term counts (using a sparse matrix representation). The output is then passed to the tfidf Transformer that converts the count matrix into its tf-idf representation. The configuration that is used is the following:

- min_df : 2. This means that the terms must appear at least in 2 documents to be considered. The list with those words that do not satisfy this condition can be found in the path *out/NMF/stopwords.csv*.
- ngram_range : (1,3). This means that they will be used bigrams and trigrams in the search process. Bigrams and trigrams are expressions consisting of two and three words respectively that are repeated across documents (p.e. “climate change” or “gender equality” could be considered as bigrams).

The rest of the parameters are left with the default values.

Dimensionality reduction and topics extraction

As it was mentioned previously, the NMF process consists in finding two non-negative matrices (W, H) whose product approximates the initial non-negative matrix (TF-IDF). In Figure 3.1 it may be seen the graphical representation of the model. A brief insight inside each matrix is the following:

- Matrix A(TF-IDF), size: [terms x documents]. Represents the contribution of each term to each document.
- Matrix W, size: [terms x topics].
- Matrix H, size: [topics x documents].

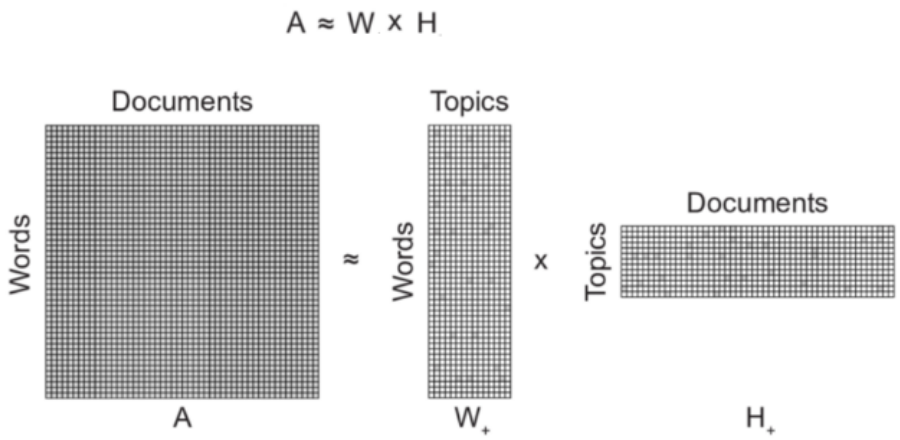


Figure 3.1: Graphical representation of an NMF model.

The model internally tries to minimize the objective function provided in Equation 3.2. The other terms are not so shown in the equation because they all are set to 0, so they do not have any effect on the results.

$$L(W, H) = 0.5 \cdot \|A - WH\|^2_{loss} \tag{3.2}$$

As it may be de deduced, despite being an unsupervised model (the topics are extracted automatically), the number of the topics that shall be used must be introduced by the user. In this case, it is set to 20 topics. The reason is that the model should be able to identify the 16 SDGs[1] and as described in Section 2, they have introduced some specific-information texts related to the SDG3, so it is assumable to expect that it should appear in two topics. Nevertheless, this process has been iterated from 15 to 21 topics, and then it is concluded that 20 was the best option based on the quality of the topics extracted. The number of iterations is set to 2000. With these parameters, the Frobenius norm of the matrix difference between the training data and the reconstructed data from the fitted model is only 29.6, being 10305 the total sum of the matrix (low reconstruction error).

SDGs-Association Matrix

Once the model has been trained, and the topics extracted, then it is required to associate each of those topics with at least 1 SDG. This process could be done manually, but it would be very demanding for the user, and it would be a difficult task for those topics that are associated with more than 1 SDG since the user would not know what percentage assign to each of them beforehand. For this reason, the SDGs association is done automatically. The steps that automatically take are the following:

1. Each text has been labelled during the generation of the training dataset with at least 1 SDG.
2. A SDG-Association matrix is initialized. It has a size of (number_topics x number_SDGs) so in this case, it would be (20 x 17). Each of the rows indicates the degree of association (in the range [0, 1]) that the topic has with each SDG.
3. Once the model is trained, then each of those training texts is input into the model. The output is the score of association that the text has with each of the topics (i.e. a vector between -1 and 1 with size [1 x number_topics]).

[1]The 17th SDG is currently not considered since it is a mix of the other SDGs.

4. A new vector is created per text, with 0's in general and with 1's in those positions corresponding to the SDG that the text is associated with (p.e. if the text is associated with SDG3, then the vector would be: [0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0]).

5. For each topic, it is added to the corresponding row in the matrix the result of the multiplication of the topic_score with the vector of 1's and 0's. Once all the texts have been processed, all the rows are normalized if any of the values are greater than 0.1. Otherwise, an exception is raised since that topic is not relevant to the study.

The final matrix of the NMF model is shown in Table 3.1. It can be seen that some topics are associated with a unique SDG such as topics 1 and 2 with 9 and 6 respectively. The last row is the sum of the total representation that each SDG has in the matrix. This value is important since a low one would indicate that the SDG is not represented in the matrix, so it is expected that the model would work poorly to identify that SDG. This is the main reason why they selected 20 topics because in this case, the lowest value is 0.81 for the SDG1. If a lower number was selected, this value was lower than 0.31, so it could not be accepted. Finally, it must be remarked that due to the greater number of texts associated with SDG3, its representation in the matrix is also higher. In this case, it does not mean that there will be a bias towards that SDG, but that more than 1 topic is associated with that SDG.

Topic	SDG																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	0	0	0	0	0	0	0	0	0.51	0	0.49	0	0	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0.16	0	0	0.84	0	0	0	0	0	0	0	0	0	0	0
4	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
8	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
9	0.81	0	0	0	0	0	0	0	0	0.19	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
11	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
14	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
15	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
18	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
19	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Total	0.81	1	2.16	1	1	1.84	1	1	1.51	1.19	1.49	2	1	1	1	1	0

Table 3.1: Association matrix of the NMF model. Each row indicates the score of association ([0, 1]) that the corresponding topic has with each SDG.

Table 3.2 shows the top 25 words of topics 1, 2 and 4. At the top, it can be seen the SDGs with which they are associated, being the SDG7, 6 and 3 respectively. Each word has associated a score, which indicates the importance of that word for each topic, the higher the score the more important that the word will be. As it can be seen, the auto-generated topics with the NMF model are direct to understand and guess what the topic is related to the topic1 related to the SDG7 has the words energy, renewable, electricity, energy efficiency, fuel, emission, energy consumption which are related

to the Affordable and clean energy. The same applies to topic2 with the Clean water and sanitation (SDG6) and topic4 with Good health and well-being (SDG3). Also, it must be remarked how the use of bigrams and trigrams greatly improves the results, and they have proved to be very useful. Some of them are: "energy efficiency", "clean energy", "water scarcity", or "mortality rate".

Word	Topic1 - 1.00*SDG7	Topic2 - 1.00*SDG6	Topic 4 - 1.00*SDG3
0	1.416:energy	1.374:water	0.305:africa
1	0.298:renewable	0.264:water resource	0.300:saharan
2	0.275:electricity	0.227:management	0.300:sub saharan
3	0.257:renewable energy	0.210:resource	0.299:sub
4	0.254:energy efficiency	0.174:water resource management	0.296:sub saharan africa
5	0.237:efficiency	0.167:resource management	0.296:saharan africa
6	0.214:fuel	0.163:scarcity	0.276:hiv
7	0.198:technology	0.149:supply	0.235:rate
8	0.193:clean	0.147:water scarcity	0.210:adolescent
9	0.161:emission	0.139:water related	0.200:asia
10	0.159:access	0.131:water quality	0.196:child
11	0.143:carbon	0.123:sector	0.189:death
12	0.114:global	0.121:transboundary	0.155:incidence
13	0.112:energy consumption	0.121:water supply	0.149:new
14	0.109:cooking	0.120:sanitation	0.146:infection
15	0.108:fossil	0.118:water efficiency	0.139:globally
16	0.107:power	0.114:integrated water	0.137:tuberculosis
17	0.106:sector	0.114:integrated water resource	0.136:death live
18	0.101:clean energy	0.109:efficiency	0.133:mortality
19	0.101:energy energy	0.105:water sector	0.131:southern asia
20	0.101:energy sector	0.104:fresh water	0.129:southern
21	0.098:access clean	0.102:country	0.125:region
22	0.097:fossil fuel	0.102:fresh	0.116:mortality rate
23	0.096:global energy	0.099:water policy	0.114:live
24	0.094:source	0.097:water supply sanitation	0.112:highest
25	0.094:fuel technology	0.097:supply sanitation	0.111:uninfected

Table 3.2: Top 25 words per topics 1, 2 and 4 of the NMF model.

3.2 Latent-Dirichlet Allocation

In Section 3.1 it was introduced the reduction features of the TF-IDF method, which can perform a basic identification of those sets of words that are discriminative for documents. However, it provides only a small reduction in the size of the matrices. To address this issue, other methods appeared as the Latent Semantic Index (LSI). It is a statistical method that used a singular value decomposition o the original matrix to identify a linear subspace in the TF-IDF matrix, trying to capture most of the variance in the collection. One step forward was the introduction of the probabilistic Latent Semantic Index (pLSI) method. It uses a probabilistic method instead of Singular Value Decomposition. The main idea is that it tries to find a probabilistic model with hidden variables (i.e. latent variables) which can generate the data in the original document-term matrix.

To tackle the problems of both LSI and pLSI, it was introduced the Latent-Dirichlet Allocation (LDA) model. It is a generative probabilistic model for collections of discrete data [2]. It is a three-level hierarchical Bayesian model, where each item of a collection (i.e. a text) is modelled as a finite mixture over an underlying set of topic probabilities; each topic is then modelled as an infinite mixture over an underlying set of topic probabilities. In terms of text modelling, these topic probabilities provide an explicit representation of a text. However, LDA does not only model text corpora and can apply to other collections of data, such as data from collaborative filtering, content-based image retrieval and bioinformatics.

LDA is a mixture model that captures the exchangeability of both words and documents. The assumption of exchangeability for words in a document means that the order of words in a document is not important, and likewise for the ordering of documents in a corpus. This does not imply that they are fully independent, but they are conditionally independent based on an underlying probability distribution [2]. This assumption has been also mentioned in the NMF model (Section 3.1). The principal reason why it is used is that it leads with high computational efficiency, and is fast to run.

Based on the idea that each text can be represented as a random mixture over latent topics (i.e. hidden topics that can not be modified externally) and each topic follows a distribution over words, then to generate a text:

1. Randomly chooses a distribution over topics (θ, following a Dir(α))
2. For each word in a document:
   - Randomly chooses a topic from the distribution over topics (z, following a Multinomial(θ)).
   - Randomly chooses a word from the corresponding topic (w, given z and following a Multinomial(β)).

The dimensionality k of the Dirichlet distribution is assumed know and fixed. A k-dimensional Dirichlet random variable θ can take values in the (k $-$ 1) simplex (a k-vector θ lies in the (k $-$ 1) simplex if $\theta_i >= 0, \sum_{i=1}^{k} \theta_i = 1$. Given the parameters α and β, the marginal distribution of a document is:

$$p(d|\alpha, \beta) = \int p(\theta|\alpha) \left(\prod_{n=1}^{N} \sum_{z_n} p(z_n|\theta) p(d_n|z_n, \beta) \right) d\theta, \tag{3.3}$$

where $p(\theta|\alpha)$ is the probability of θ following a Dirichlet distribution over α, $p(z_n|\theta)$ is the probability of a topic following a multinomial distribution over θ and $p(d_n|z_n, \beta)$ is the probability of a document given a multinomial distribution over β and being conditioned by the topic z_n.

Figure 3.2 shows the representation of the LDA model as a probabilistic graphical model. As it was mentioned previously, there exist three hierarchical levels:

1. The corpus level, with the parameters α and β and are sampled only once when generating the corpus. These are the parameters that the model should learn.
2. The document level, with the parameter θ is sampled once per document.
3. The word level, with the variables z_n and w_n sampled once per document and word.

The great advantage that LDA has over other models is that each document can be associated with multiple topics since the topic node is sampled repeatedly within each document.

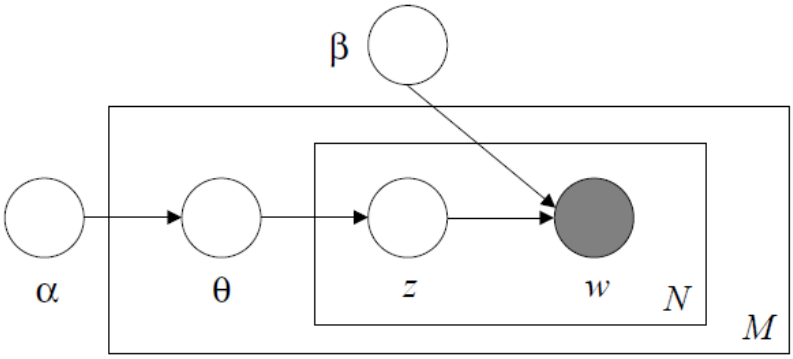


Figure 3.2: Graphical model representation of LDA [2].The outer plate represents documents, while the inner plate represents the repeated choice of topics and words within a document.

Optimization

In this case, several parameters are optimized to get the model to work properly:

- Bigrams and trigrams are allowed (two and three consecutive words that together have a special meaning. To consider the bigrams and trigrams in the vocabulary, they should appear at least 5 times in the whole corpus.
- Only those words that appear at least two times in the corpus are allowed. Also, if the frequency of the word appearing in the corpus is greater than 70 % then they are deleted (a word with a high frequency does not give information to the model).
- All the texts extracted from the dataset (training, validation and analysis) are lemmatized.

Once it is defined the base configuration of the input to the model, is followed by an optimization procedure to adjust the remaining parameters of the model:

- Number of topics: this is the same parameter as in the NMF model (Section 3.1). Since there are 16 SDGs into consideration, this parameter is varied from 16 to 21.
- Passes: number of times that the optimization procedures pass over the whole corpus. Since the corpus is small, this parameter is fixed to 1000.

- Iterations: maximum number of iterations that the model can take to obtain the topic distribution in the corpus. This is varied from 10 to 400.
- Only positive: a boolean flag that if set to true, then only those scores of topics whose values are greater than 0 are considered for the SDGs mapping. It is set to true.
- Random state: Seed used for reproducibility. It is set to 1.

The other parameters of the model are left to the default values. The best model resulting from the optimization is selected based on the best association matrix (i.e. the best representation for all the SDGs), the quality of the topics and the log-perplexity of the model. The perplexity is a intrinsic evaluation metric widely used for model evaluation [6]. It tries to capture how surprised a model is with new data that it has not seen before. Generally, it is calculated based on the held-out data (i.e. the part of the dataset that is not used for training). Since in this case the complete dataset is used for training, then it is used some texts from the analysis dataset to evaluate the performance of the different configurations. The lower the perplexity, the better the model is since it can generalize better (it is less surprising when new words are being used). After the optimization, the final configuration is: 17 topics, 1000 passes, 10 iterations and only positive True. To choose the optimal configuration, it is mainly taken into account the representativity that the model has with each SDG (i.e. all of them should be present), the coherence of the topics and the log perplexity. In this case with 17 topics, it is obtained the lower perplexity (-11.5).

SDGs-Association Matrix

The process of how the association matrix is obtained was described in Section 3.1; even if the models are internally different both of them share the same interface. The Association matrix of this model can be seen in Table 3.3. The lower contribution is for SDG2 (0.68), however, it is still acceptable. In the other SDGs, it is very proximate to 1, being the case of SDG8 (1.48) the SDG most representative.

Top.	SDG															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.77	0.00	0.23	0.00	0.00	0.00
2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.18	0.62	0.20	0.00	0.00	0.00	0.00	0.00	0.00
3	0.00	0.19	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.63	0.00	0.00	0.00	0.19
4	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
6	0.00	0.50	0.00	0.00	0.00	0.00	0.50	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
7	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
8	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
9	0.51	0.00	0.00	0.00	0.00	0.00	0.00	0.30	0.00	0.20	0.00	0.00	0.00	0.00	0.00	0.00
10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.34	0.66	0.00	0.00	0.00	0.00
11	0.39	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.61	0.00	0.00	0.00	0.00	0.00	0.00
12	0.00	0.00	0.00	0.00	0.00	0.00	0.44	0.00	0.22	0.00	0.00	0.00	0.34	0.00	0.00	0.00
13	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
14	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
15	0.25	0.00	0.00	0.75	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
16	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00
17	0.00	0.00	0.00	0.00	0.00	0.00	0.27	0.00	0.00	0.00	0.00	0.00	0.73	0.00	0.00	0.00
Tot.	1.15	0.68	1.00	0.75	1.00	1.00	1.22	1.48	0.84	1.00	1.11	1.29	1.30	1.00	1.00	1.19

Table 3.3: Association matrix of the LDA model. Each row indicates the SDGs to which the corresponding topic is related.

Table 3.4 shows the top 25 words of topics 7, 8, 13 and 15. At the top, it can be seen the SDGs with which they are associated. Each word has associated a score, which indicates the importance of

that word for each topic, the higher the score the more important that the word will be. It can be noticed that from the words of each topic it could be possible to guess the SDGs with which that topic is associated, in other words, the generated topics are coherent and easy to understand. For example, topic 8 contains the words: health, death, disease, mortality, child, communicable, maternal... so it would be assumable to think that the topic is related to Good health and well-being (SDG3). Topic 15 contains words such as education, school, primary, and learning but also others such as extreme or poverty, so it is also rational to think that it is associated with No poverty (SDG1) and Quality education (SDG4).

Topic 7	**Topic 8**	**Topic 13**	**Topic 15**
1.00*SDG6	**1.00*SDG3**	**1.00*SDG16**	**0.75*SDG4,0.25*SDG1**
0.087:water	0.031:health	0.036:country	0.041:education
0.027:sanitation	0.026:death	0.022:right	0.029:child
0.016:country	0.024:disease	0.015:institution	0.025:school
0.015:access	0.014:rate	0.014:human	0.020:primary
0.013:drinking	0.013:mortality	0.013:violence	0.014:poverty
0.012:management	0.012:child	0.013:child	0.013:country
0.012:service	0.010:care	0.013:victim	0.010:learning
0.009:drinking_water	0.009:global	0.012:law	0.009:africa
0.009:water_sanitation	0.008:globally	0.012:human_right	0.009:saharan
0.008:resource	0.008:africa	0.011:access	0.008:sub
0.008:asia	0.008:wa	0.010:justice	0.008:sub_saharan
0.008:basic	0.008:country	0.008:national	0.008:secondary
0.007:lack	0.008:sub	0.007:sexual	0.008:teacher
0.006:facility	0.008:live	0.007:region	0.007:quality
0.006:safely	0.007:saharan	0.007:trafficking	0.007:rate
0.006:population	0.007:hiv	0.006:available	0.007:world
0.006:sector	0.007:sub_saharan	0.006:level	0.007:student
0.006:supply	0.007:worldwide	0.006:access_justice	0.007:access
0.006:level	0.007:maternal	0.006:africa	0.007:level
0.005:hygiene	0.007:risk	0.006:sub	0.006:skill
0.005:safe	0.007:communicable	0.006:proportion	0.006:extreme
0.004:transboundary	0.006:progress	0.006:conflict	0.006:training
0.004:global	0.006:estimated	0.006:public	0.006:population
0.004:health	0.006:new	0.006:population	0.005:primary_secondary

Table 3.4: Top 25 words associated with topics 7, 8, 13 and 15 of the LDA model. The score before the word indicated the weight that word has on the corresponding topic.

3.3 Top2Vec

The Non-Negative Matrix Factorization (NMF) and Latent-Dirichlet Allocation (LDA) models described in the Sections 3.1 and 3.2 have some major limitations:

- The user has to introduce the number of the internal topic to consider since they discretize the continuous topic space into those topics. This is a major limitation since the user does not know beforehand which could be an appropriate number of topics (especially for very large or unfamiliar datasets), requiring for example some kind of measure to understand how well a model fits the data.
- Both methods rely on the bag-of-words assumption, so intrinsically they are losing the information about the semantics and order of the words. To overcome these limitations it is introduced the Topic to Vector (Top2Vec) method.
- If stop-words are not filtered out from the texts or the words are not lemmatized, then some very frequent words such as "the" or "and" will have the highest probabilities and the topics extracted will lack meaning and coherence. This has the extra difficulty that sometimes the collection of stop-words is not well known since apart from the global words are also problem-specific words to remove and in most cases, it is done via manual recognition.
- LDA is a probabilistic generative model, which means that it fits the latent variables to minimize the reconstruction error of the matrix (i.e. recreate the original document-word distributions). The major limitation of this approach is that it can not differentiate informative from uninformative words

Topic to Vector (Top2Vec) is a method based on the distributed representation of both words and documents, being able to capture the semantics of the words (and then, the documents) [1]. Some of the great advantages of this model are that it does not require stop-word lists, stemming or lemmatization. This is a great advantage because words such as "big" and "large" would be recognized as different since they do not share the same stem, but they are semantically recognized as similar with this model. Also, it automatically finds the number of topics. The resulting topic vectors are jointly embedded with the document and word vectors with the distance between them representing semantic similarity. The topics are assumed to be continuous as a certain topic may be represented by many combinations of weighted words. Each document is associated with a unique topic, being a continuous combination of the topics discovered by the model. In this way, the topic of the document is described as a set of weighted words and they should be able to represent a high-level summary of its contents.

Distributed representation of words, topics and documents

A distributed representation implies that the learned concept (p.e. a word) is represented by many neurons inside the network. The task is then to adjust the weights of the internal neurons, meaning that they may participate in the representation of multiple concepts at the same time. The great advantage of this distributed learning is that it automatically leads to the generalization of the learned concepts, avoiding the problem known as "overfitting". The vector representation of words also guarantees the "distributional hypothesis", which means that semantically similar words will be used in similar contexts.

The first model that put this idea into practice was word2vec [11]. It is a neural network able to capture both semantic and syntactic word relationships. It learns by trying to predict the words

that should be adjacent to a specific word given its context (i.e. a sliding window over the whole document). Then it satisfies the distributional semantics hypothesis since it learns word vectors from those words used in similar contexts. It has been a very useful model, although current tendencies try not to rely on neural networks (the cost of training and using them is higher).

Extending the idea and work developed in word2vec, it was introduced doc2vec [10]. This model also learns the internal vectors of paragraphs. Now it not only used the context window of words to predict a specific word but also the paragraph vector, acting them as a memory of the topic of the document. In this way, it can learn vector representation of whole documents.

The semantic space is a mathematical space representation in which distance is directly related to semantic similarity (i.e. two words that are close in the semantic space will be semantically similar). Doc2vec can learn both vector representations of words and documents. In this way, those words whose vectors are very close to the document vector will be more representative of the document's topic. The combination of both word and document embedding is known as "semantic embedding".

The assumption used in top2vec is the following: the semantic embedding space is a continuous representation of both words and documents. Then each point in the semantic space is a different topic that is best represented by those words which are closer to it. In the same way, those documents that are close to each other near the point will have a similar topic, they are semantically similar. Based on this, top2vec looks for those areas in the semantic space that are denser than others, and from those dense areas the topic is extracted (those will be the prominent topics). The topics vector is extracted from the centroid of each area; its interpretation is an average document representative of all the documents around that point. Also, another great improvement introduced with top2vec is that the user can decide to hierarchical group those topics that are very close in the semantic space, which is usually known as "topic reduction" (since the number of topics discovered is reduced).

Description of the model

In the previous section it has been described the Semantic Space as a mathematical space with the following special properties:

- The distance between words and document vectors represented the semantic association that they have.
- Documents that are semantically similar will be placed close together.
- Those words that surround a document should be able to describe the document well.
- To learn jointly embedded word and document vectors is used word2vec.

An example of semantic space can be visualized in Figure 3.3. The description of how the model internally learns the corresponding matrices is out of the scope of the present work (it can be found in [1]).

As it has been mentioned before, in the jointly embedded word and document space both words and documents are represented as positions. Those documents that are semantically similar are placed close to each other. In this way, those areas of the space with a high density of documents are indicative of an underlying common topic that is shared among those documents. The topic point is then calculated as the centroid of each of those areas, and the words closer to those centroids will be the most representative. In order to find those areas, it is used the Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN).

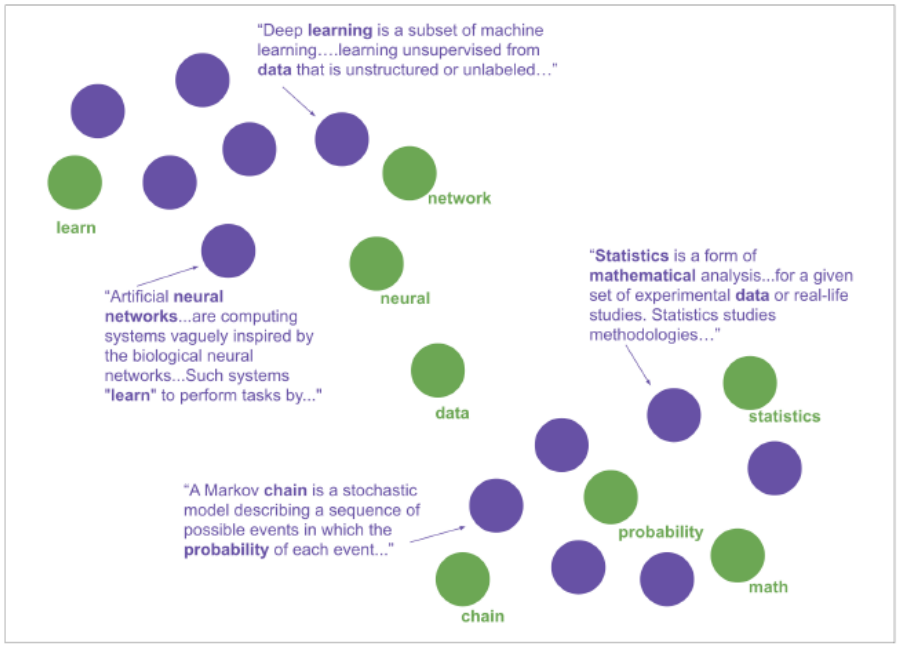


Figure 3.3: An example of a semantic space. The purple points are documents and the green points are words. Words are closest to documents they best represent and similar documents are close together [1]

Before finding the dense area of documents, it is required to perform a dimension reduction on the vector space. This is due to the problem called as "curse of dimensionality", where the document vector embeddings are very space when the dimension of the vector is high enough, which makes it difficult to find the areas of a high density of documents. The dimensionality reduction is done with the Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP) algorithm.

Once the clusters with high density have been discovered in the semantic space, then it is required to calculate the topic vectors. Although there exist several algorithms for this purpose, in this case, the topic vector is calculated as the centroid of those clusters. It then is calculated as the arithmetic mean of all the document vectors associated with that cluster. Figure 3.4 is shown the topic vector as the centroid of the dense area of documents identified by HDBSCAN.

Once the topics are found, then it is required to find the words associated with each topic (statistically the words that are more representative of that topic). The distance of each word to the topic indicates how similar is that word to the topic. Then, the words that are closer to the centroid of the cluster (topic vector) can be used to summarize the contents of that topic and the documents close to it. Those words that are common to all the documents will be equally distant from all the topics, and thus will not be considered representative words of any of the clusters. This implies that stop-word removal is not required when using Top2Vec.

Optimization

The optimization of Top2Vec is more complex than the optimizations followed in Sections 3.1 and 3.2 [2]. The studied parameters are the following:

- min_count: minimum number of times that a word must appear in the corpus to be considered. This is set to 2.

[2]Documentation about Top2Vec can be found in Top2Vec.

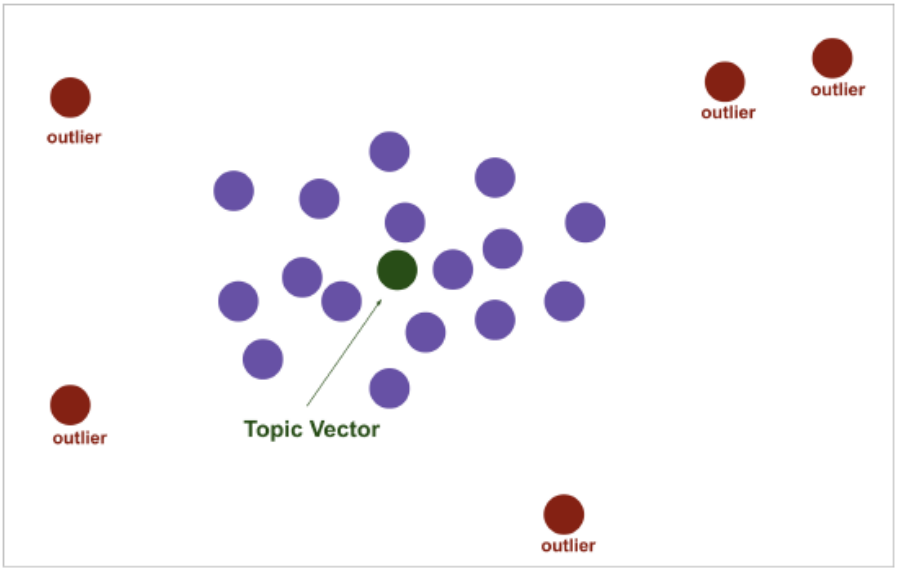


Figure 3.4: The topic vector is the centroid of the dense area of documents identified by HDBSCAN, which are the purple points. The outliers identified by HDBSCAN are not used to calculate the centroid [1]

- ngram_vocab: similar to using bigrams and trigrams. Based on the results obtained previously, it is set to True.
- embedding_model: the model used to generate the document and word embeddings. They used the doc2vec embedder (it trains a model from scratch working well with large datasets), the universal-sentence-encoder (faster model to train than the others since it has already been pre-trained, and it's suggested for small datasets) and the all-MiniLM-L6-v2 (it uses the SBERT pre-trained sentence transformer.).
- speed: it determines the speed of learning when the doc2vec is used as the embedding model. It can be fast-learn, learn and deep-learn. All of them are tested.
- split_documents: if set to True then the documents are split before being used. This is set to False.
- lemmatized: although theoretically it is not required to lemmatize and remove stop-words from the texts (As explained in Section 3.3), both raw and lemmatized versions of the corpus are tested.
- only_positive: if set to true, then only those topics whose coherence is greater than 0 are used for the SDGs mapping. This is set to true.

It should be remarked that even with the same configuration, the trained model may vary from one training to the next one. For that reason, each configuration is run several times and then the obtained results are averaged. The obtained results can be seen in Table 3.5. Based on them, the final chosen configuration is the first one due to the following reasons:

- Using configuration 2 the obtained scores are worse than with the first one, it takes more time to run and one of the SDGs only has a 0.11 score of representation in the Association Matrix, which is too low to be considered valid.
- Using configuration 3 the number of topics obtained is reduced (only 14) so the matrix association is mixed. Furthermore, when one topic is detected to be related to the text automatically it is induced one tendency to other SDGs that may not be related to the contexts of that text.

- Using the universalsentenceencoder the number of topics is ok, but the minimum representation of the SDGs is 0.5, which is much worse than configuration 1 (in which the min one is 1).
- Using the first configuration with the dataset not being lemmatized it takes more time to train and the obtained results are worse, so it is also discarded.

embedding_model	**speed**	**parsed**	**nTopics**	**Sc. Ind**	**Sc. Gr**	**Min. Rep**	**Max. Rep**
doc2vec	learn	True	17	57	50	1	1.44
doc2vec	deep-learn	True	15	26.22	25	0.11	1
all-MiniLM-L6-v2	learn	True	14	33	30	0.4	1.7
universal-sentence-encoder	learn	True	16	37	32.15	0.5	1.4
doc2vec	learn	False	15	41.46	38	0.41	1.37

Table 3.5: Configuration and results of the Top2Vec model's optimization. Sc.Ind and Gr are the scores obtained for identifying correctly the individual and group SDGs respectively in the validation dataset. Min.Rep and Max.Rep are the minimum and maximum representation of the SDGs in the Association Matrix.

To summarize, the final chosen configuration is: doc2vec (embedding model), learn (speed) and lemmatized.

SDGs-Association Matrix

The process of how the association matrix is obtained was described in Section 3.1; even if the models are internally different both of them share the same interface. The Association matrix of this model can be seen in Table 3.6. The are 17 topics in total. As can be seen, the minimum representation of the SDGs is 1.00. The topic1 is the only topic that mixes SDGs, the others are associated with a unique SDG. This is very positive at the time to analyse and identify the SDGs in the texts. The max value is 1.45, so it is still close to 1.

Top.	SDG																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	0.00	0.00	0.00	0.00	0.00	0.00	0.45	0.00	0.00	0.00	0.28	0.27	0.00	0.00	0.00	0.00	0.00
2	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
4	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
6	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00
9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
10	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
11	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
12	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
13	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
14	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
15	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00
16	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
17	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Tot.	1.00	1.00	1.00	1.00	1.00	1.00	1.45	1.00	1.00	1.00	1.28	1.27	1.00	1.00	1.00	1.00	0.00

Table 3.6: Association matrix of the Top2Vec model. Each row indicates the SDGs to which the corresponding topic is related.

Table 3.7 shows the top 25 words of the SDG14, SDG4, SDG16 and SDG6. At the top, can be seen the SDGs with which they are associated. Each word has associated a score, which indicates the

importance of that word for each topic, the higher the score the more important that the word will be. It can be noticed that from the words of each topic it could be possible to guess the SDGs with which that topic is associated, in other words, the generated topics are coherent and easy to understand. For example, the topic associated with the Quality education (SDG4) contains the words: secondary, proficiency, reading, school, learning, enrolment, scholarship or teacher. Also, the topic associated with the Clean water and sanitation (SDG6) contains words such as defecation, sanitation, drinking, wastewater, toilet or handwashing.

1.00*SDG14	**1.00*SDG4**	**1.00*SDG16**	**1.00*SDG6**
0.865:overfishing	0.903:secondary	0.844:punishment	0.873:safely
0.855:acidification	0.888:proficiency	0.821:rule	0.858:defecation
0.837:destructive	0.879:reading	0.817:aggression	0.849:sanitation
0.835:coastal	0.850:school	0.815:underreporting	0.832:drinking
0.833:marine	0.848:learning	0.813:violation	0.797:soap
0.830:fish	0.837:completion	0.807:torture	0.787:wastewater
0.825:ocean	0.834:mathematics	0.807:discipline	0.786:toilet
0.821:anthropogenic	0.826:enrolment	0.790:childrens	0.784:practise
0.807:mile	0.803:developmentally	0.783:disturbing	0.753:latrine
0.806:acidity	0.802:education	0.777:psychological	0.752:hygiene
0.791:precious	0.793:numeracy	0.776:unsettling	0.750:lacked
0.787:oxygen	0.788:literacy	0.775:detention	0.737:water
0.785:nautical	0.786:scholarship	0.774:freedom	0.713:operational
0.779:conserve	0.779:write	0.774:journalist	0.711:discharged
0.774:biologically	0.778:finished	0.774:intercourse	0.711:lake
0.770:biodiverse	0.776:primary	0.773:upholding	0.709:river
0.767:eutrophication	0.768:acquire	0.766:physically	0.684:washing
0.763:portion	0.767:lifelong	0.766:violence	0.673:menstruation
0.762:jurisdiction	0.766:read	0.763:homicide	0.669:handwashing
0.755:yield	0.750:teacher	0.759:armed	0.667:treated
0.754:fishing	0.747:ofschool	0.758:unionist	0.663:unsafe
0.752:shortest	0.742:refocused	0.755:justice	0.661:responded
0.739:impairs	0.742:ready	0.754:untold	0.652:withdrawn
0.734:unreported	0.739:teaching	0.749:previous	0.651:lacking
0.733:ultimately	0.739:pedagogical	0.747:victim	0.645:scarcity
0.733:fishery	0.739:skill	0.747:feel	0.645:hand

Table 3.7: Top 25 words associated with SDG14, SDG4, SDG16 and SDG6 of the Top2Vec model. The score before the word indicated the weight that word has on the corresponding topic.

3.4 BERTopic

BERTopic is a topic modelling technique very similar to Top2Vec (Section 3.3) since both are clustering-based techniques and unsupervised [4]. BERTopic extracts coherent topic representation via the implementation of a class-based variation of TF-IDF. The steps it follows are[3]:

1. Generating the document embeddings with a pre-trained transformer-based language model, Bidirectional Encoder Representations from Transformers (BERT). The embedded words which are semantically similar will be placed close to each other in semantic space. In this way, document-level information is extracted from the corpora. This embedding step is performed using the Sentence-Bert (S-BERT) framework. It has the advantage that the transformer has already been pre-trained on a large corpus, so it will work better when the dataset is small than the other models.

2. The document embeddings are dimensionally reduced. This is because as data increases in dimensionality, the distance to the closest point tends to approach the distance to the farthest point. As a result, in high dimensional space, the concept of spatial locality becomes ill-defined and distance measures differ little [4]. It uses Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP).

3. Following a density-based method clusters is created. This technique assumes that words closer to the cluster's centroid are most representative of that cluster. However, in practice, a cluster will not always lie within a sphere around a cluster centroid which might conduce to the extraction of misleading topics. It used Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN).

4. Topics vectors are extracted from the cluster. To overcome the limitation of the centroid-based perspective, it is used a class-based version of TF-IDF. This has the advantage that the clustering technique is separated from the topics generation, allowing to have more flexibility. It is explained in the following section.

Topic representation and Dynamic Topic Modelling

The TF-IDF was explained Section 3.1. It measures how important a term is for a document (i.e. how much information it provides). This technique is generalized by treating all documents in a cluster as a single document. Once they are concatenated, the TF-IDF is used by translating the documents to clusters using Equation 3.4.

$$W_{t,c} = tf_{t,c} \cdot \log(1 + \frac{A}{tf_t}), \tag{3.4}$$

$W_{t,c}$ represents the weight of the term in the cluster, $tf_{t,c}$ is the frequency of the term in the cluster, A is the average number of words per cluster and tf_t is the frequency of the term in all the clusters. In this way, if the term is very common then the logarithmic will tend to 0 and the term will receive no importance during the topic extraction.

BERTopic also differs from the others algorithm in the temporal nature of the generated topics. Traditionally the topics generated are static, however, BERTopic uses dynamic modelling. It then assumes that the temporal nature of topics should not influence the creation of global topics. To achieve it, first, it is generated a global representation of topics before developing a local representation.

[3]The steps are have been summarized since they are detailed in Section 3.3

In practice, firstly all the topics are generated following the procedures explained before. Then, local copies of the topics are created by multiplying the term frequency of documents at that timestep with the pre-calculated global values.

Optimization

The optimization of BERTopic is more complex than the optimizations followed in Sections 3.1 and 3.2 [4]. The studied parameters are the following:

- n_gram_range: the range of the n-grams that should be considered. Based on the results obtained with the other models, it is set to (1,3) (i.e. bigrams and trigrams are allowed to be discovered).
- embedding_model: model to generate the word and document vectors. Both "all-MiniLM-L6-v2" and "all-mpnet-base-v" are tested.
- top_n_words: the number of words per topic to extract. The recommended is 10, so it is studied from 5 to 15.
- min_topic_size: minimum size of the topic. It is studied from 5 to 15.
- nr_topics: the number of topics to which the model will reduce (based on a hierarchical method) the number of topics. It is set to None to avoid modifying the initial topics.
- seed_topic_list: a list of seed words per topic to converge around. It is studied with and without this list. It is generated a list with 16 topics (1 per SDG), the respective words can be seen in Table 3.8.
- lemmatized: whether to use lemmatized and filtered texts or in raw form. Both forms are studied.
- calculate_probabilities: whether to calculate the probabilities of all topics per document instead of the probability of the assigned topic per document. It is set to True since the documents may be a mix of several topics.

It should be remarked that even with the same configuration, the trained model may vary from one training to the next one. For that reason, each configuration is run several times and then the obtained results are averaged. The obtained results can be seen in Table 3.9. Based on them, the final chosen configuration is: all-MiniLM-L6-v2 (embedding model), 10 top words, 10 topic sizes, no seed list, and texts lemmatized. The decision is mainly taken based on the number of topics (they should be around 16, so if they are less than 15 or greater than 19 they are discarded), the minimum and maximum representation values of the SDGs should be 0.5 and 1.4 respectively to avoid having a large variance.

SDGs-Association Matrix

The process of how the association matrix is obtained was described in Section 3.1; even if the models are internally different both of them share the same interface. The Association matrix of this model can be seen in Table 3.10. The are 17 topics in total. As can be seen, the minimum representation of the SDGs is 1.00, so all of them are correctly represented when identifying texts. The maximum value, in this case, is 1.45, so it is still close to 1. Also, almost all the topics are uniquely associated with 1 SDG.

[4] Documentation about BERTopic can be found in BERTopic.

SDG	Words
1	poverty, social, disaster, poor, vulnerable
2	food, hunger, nutrition, food insecurity
3	health, disease, mortality, death
4	education, school, teacher, learn
5	gender, gender equality, sexual
6	water, clean water, sanitation, drinking
7	energy, renewable, electricity
8	decent work, economic growth, employment, productivity
9	industry, innovation, infrastructure, manufacturing
10	inequality, reduced inequality, developed country
11	city, community, sustainable, urban, public transport
12	consumption, production, material, footprint
13	climate, adaptation, global warming
14	marine, ocean, fish, marine ecosystem
15	forest, biodiversity, land
16	peace, justice, institution, human right

Table 3.8: List of words per SDG used as a seed list for the topics generation in the BERTopic model.

ngram	embed	topW	minS	list	lem	perc_test	perc_train	nT	min	max
(1,3)	miniLM	5	10	False	True	54.88 - 50.61	91.10 - 87.60	16	0.887	1.26
(1,3)	miniLM	10	10	False	True	43.90 - 40.85	53.10 - 50.80	8	0.37	0.94
(1,3)	miniLM	15	10	False	True	57.32 - 51.83	94.20 - 91.20	14	0.7	0.94
(1,3)	miniLM	10	5	False	True	51.83 - 47.56	83.90 - 81.20	16	0.4	1.2
(1,3)	miniLM	10	10	False	True	56.10 - 51.83	90.80 - 87.20	16	0.96	1.26
(1,3)	miniLM	10	15	False	True	48.17 - 43.90	54.70 - 52.40	11	0.28	0.98
(1,3)	miniLM	10	10	0	True	56.10 - 51.22	91.70 - 88.40	17	1	1.3
(1,3)	miniLM	10	10	0.5	True	54.27 - 49.39	87.50 - 84.50	13	0.5	1.3
(1,3)	miniLM	10	10	1	True	48.17 - 43.90	53.40 - 51.50	11	0.54	1.2
(1,3)	miniLM	10	10	False	False	54.88 - 50.61	92.90 - 90.00	19	0.9	1.25
(1,3)	mpnet	10	10	False	True	55.49 - 50.61	89.90 - 86.90	13	0.38	1.05

Table 3.9: Configuration and results of the BERTopic model's optimization. embed: embedding_model, miniLM: all-MiniLM-L6-v2, mpnet: all-mpnet-base-v2, topW: top_n_words, minS: min_topic_size list: seed_topic_list, nT: n_topics.

Table 3.11 shows the top 10 words of SDG3, SDG7, SDG16 and SDG14. At the top, it can be seen the SDGs with which they are associated. Each word has associated a score, which indicates the importance of that word for each topic, the higher the score the more important that the word will be. It can be noticed that from the words of each topic it could be possible to guess the SDGs with which that topic is associated, in other words, the generated topics are coherent and easy to understand. For example, the topic associated with the Good health and well-being (SDG3) contains the words: health, disease, mortality, care or HIV. Also, the topic associated with the Life below water (SDG14) contains words such as ocean, marine or fishing.

Topic	SDG																
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
1	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
4	0.83	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.17	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00
8	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
9	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00
11	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
12	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
13	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
14	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
15	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
16	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
17	0.42	0.30	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.27	0.00	0.00	0.00	0.00
Total	1.25	1.30	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.17	1.00	1.00	1.27	1.00	1.00	1.00	0.00

Table 3.10: Association matrix of the BERTopic model. Each row indicates the SDGs to which the corresponding topic is related.

Topic0: SDG3	**Topic1: SDG7**	**Topic2: SDG16**	**Topic5: SDG14**
0.037:health	0.073:energy	0.023:institution	0.049:ocean
0.033:death	0.023:electricity	0.023:country	0.047:marine
0.031:disease	0.020:renewable	0.022:right	0.023:fishery
0.023:mortality	0.017:fuel	0.019:victim	0.019:coastal
0.017:rate	0.017:renewable energy	0.018:violence	0.018:sustainable
0.015:care	0.016:efficiency	0.017:justice	0.016:fishing
0.014:child	0.016:technology	0.016:child	0.016:ecosystem
0.013:maternal	0.015:energy efficiency	0.014:human right	0.014:stock
0.013:communicable	0.015:access	0.014:human	0.013:resource
0.013:hiv	0.014:carbon	0.014:law	0.013:fish

Table 3.11: Top 10 words associated with SDG3, SDG7, SDG16 and SDG14 of the BERTopic model. The score before the words indicates the weight that they have on the corresponding topic.

Chapter 4

Validation

This chapter presents the validation of all the models described in Chapter 3, using the training dataset (for initial estimation) and then the validation dataset (for evaluating the model performance).

4.1 NMF

To validate the trained model the training dataset is the first step. Since the model has been trained with those texts, it should be able to identify the labelled SDGs present in them with high accuracy. To associate a text with some SDGs, the model takes the following steps:

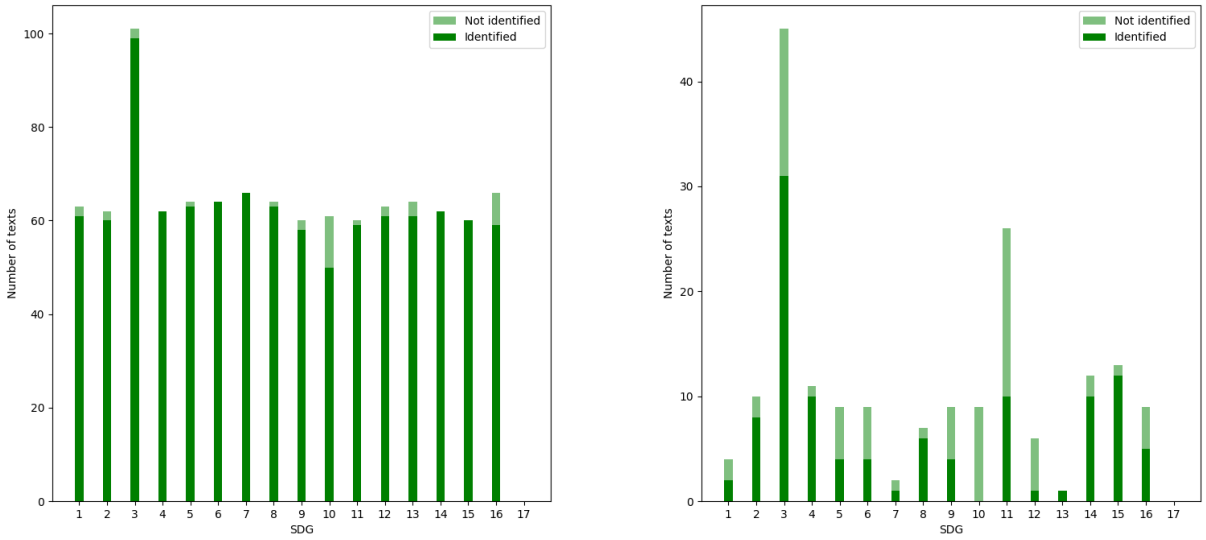
1. The topics related to that text are queried. Each topic has a score.
2. It is 0-initialized a vector with size (1 x 17). For each topic, add to the vector the multiplication of the topic_score with the associated topic_row in the association matrix (Section 3.1).
3. Those SDGs with a score lower than 0.05 are filtered out.
4. The obtained vector is scaled by a certain amount, which depends on the other models as well. As an example, in this case, it is set to 4.0.
5. If the obtained score for an SDG is equal to or greater than 0.1, then the SDG is assumed to have been identified. The threshold value is dependent on the model, and the main purpose is to differentiate those situations in which the model obtains noise or false identifiers. It is set manually based on results analysis. Finally, it must also be remarked that no normalization takes place. The reason for that is that some texts may not be associated with any SDG, so the model should be able to identify those situations too. If a normalization stage takes place, then the noise values could be greatly incremented, resulting in erroneous output.

Using the training dataset, it is obtained a 96.6 % of accuracy for the SDGs set and 97.8 % for individual SDGs. The former indicates how well are identified all the SDGs that a text is associated with while the latter indicates how well SDGs are identified. For example, if a text is related to SDGs [1, 4, 6], and the model outputs the SDGs [1, 4] then the individual score would be incremented by 2, while the global would not, since the 3 present SDGs were not identified by the model.

The result obtained is positive, since it indicates that the model can identify correctly all the associated-training texts, however, it is not sufficient because they are part of the training dataset,

so the model should be validated with a previously-unseen dataset. For that purpose it is used the Dataset from Nature (Section 2). It has 153 texts in total. In this case, it is obtained a 58.54 % in global and a 65.24 % in individual.

In Figure 4.1 is compared the identified SDGs by the NMF model with the labelled ones. It can be seen that the model performs extremely well on the training dataset, which is a necessary but not sufficient condition. It is to be remarked since they used more than 1000 documents for training. In contrast, when using the model with the validation dataset the number of SDGs not identified increases. The SDGs in which it best performs are Quality education (SDG4) and Life on land (SDG15). Others never get identified, as is the case of the Reduced inequalities (SDG10). Note however that these results are conservative since the threshold has been set to guarantee that the SDGs identified by the model are related to the contents of the text.



(a) Training dataset

(b) Validation dataset

Figure 4.1: NMF model validation comparing the identified SDGs with the previously labelled ones.

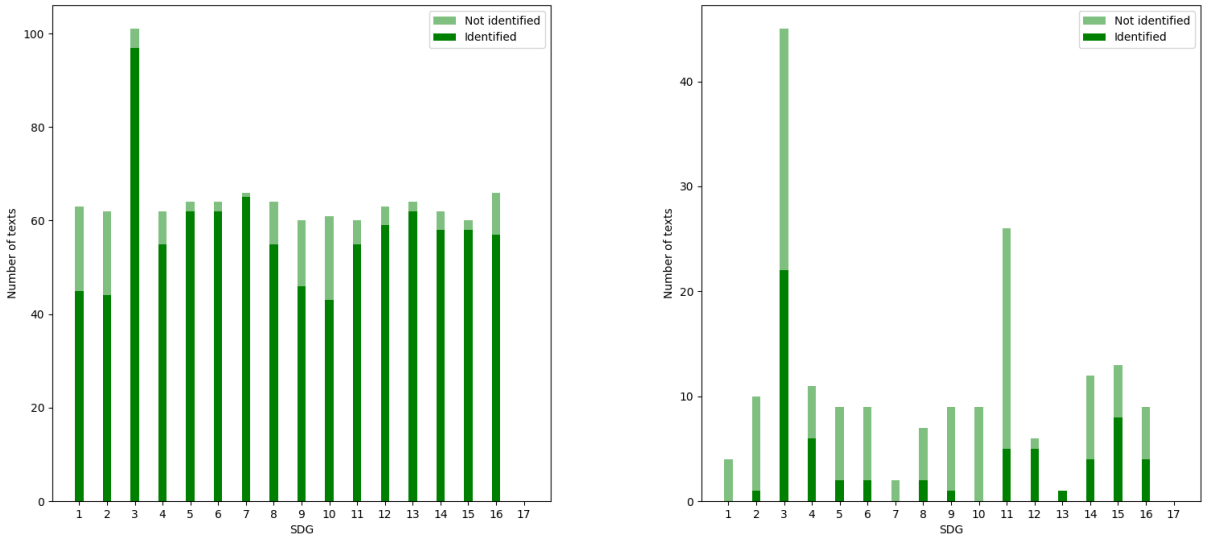
4.2 LDA

The procedure followed to validate the LDA model is the same as the one described with the NMF model (Section 4.1). Firstly it is used the training dataset; the model was trained on this dataset, so it should be able to identify the SDGs perfectly. The model outputs the probability or score of text to be related to the SDGs. In this case, it is set as a threshold value of 0.2, so if the score is greater or equal to 0.2 that SDG will be assumed to be identified. This is a very conservative value. The expand factor, in this case, is 1.3.

Using the training dataset, it is obtained a 90 % of accuracy for the SDGs set and 92 % for individual SDGs. The former indicates how well are identified all the SDGs that a text is associated with, while the latter indicates how well SDGs are identified. The result obtained is positive, since it indicates that the model can identify correctly all the associated-training texts. The procedure is repeated with the validation dataset. In this case, it is obtained a 36 % in global and a 40 % in individual.

In Figure 4.2 it is compared the identified SDGs by the LDA model with the labelled ones. It

can be seen that the model performs extremely well on the training dataset, which is a necessary but not sufficient condition. In contrast, when using the model with the validation dataset the number of SDGs not identified increases. The SDGs in which it best performs are Good health and well-being (SDG3), Quality education (SDG4) and Life on land (SDG15). Others never get identified, as is the case of the Reduced inequalities (SDG10). Note however that these results are conservative since the threshold has been set to guarantee that the SDGs identified by the model are related to the contents of the text.



(a) Training dataset

(b) Validation dataset

Figure 4.2: LDA model validation comparing the identified SDGs with the previously labelled ones.

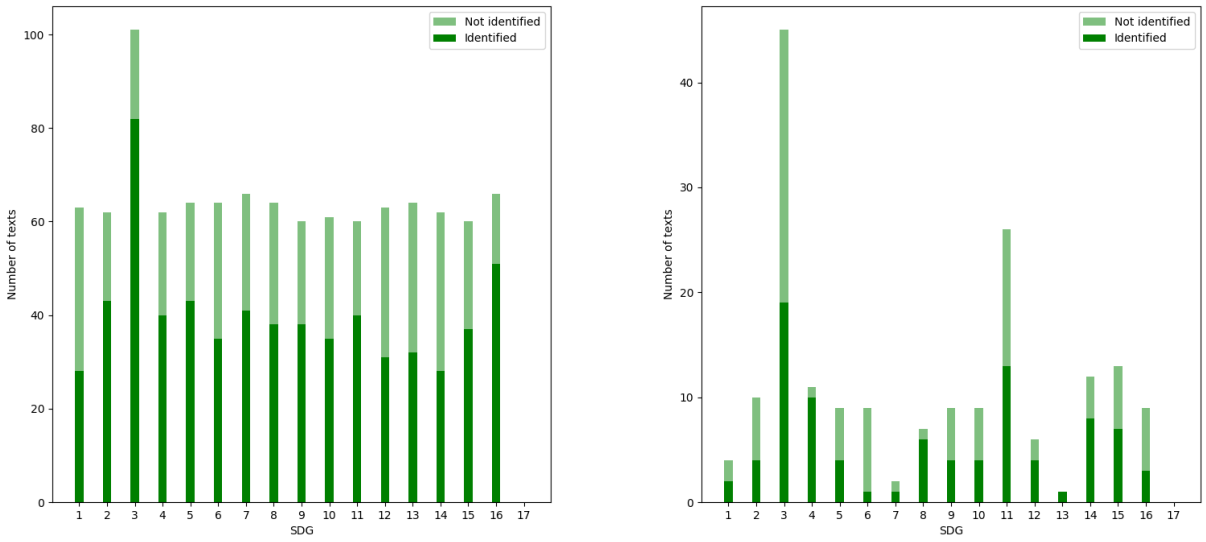
4.3 Top2Vec

The procedure followed to validate the Top2Vec model is the same as the one described with the NMF model (Section 4.1). Firstly it is used the training dataset; the model was trained on this dataset, so it should be able to identify the SDGs perfectly. The model outputs the probability or score of text to be related to the SDGs. In this case, it is set 0.2 as a threshold, so if the score is greater or equal to 0.2 that SDG will be assumed to be identified. This is a very conservative value. The expand factor, in this case, is 1.0.

Using the training dataset, it is obtained a 64 % of accuracy for the SDGs set and 64 % for individual SDGs. The former indicates how well are identified all the SDGs that a text is associated with, while the latter indicates how well SDGs are identified. In this case, the obtained results are not as good as those obtained with the NMF and LDA models. This is because those models learn only from the training texts, without using previous information (they are not pre-trained). In contrast, the Top2Vec uses pre-trained transformers so it not only focuses on the information provided. Repeating the process with the validation dataset, it is obtained a 48 % in global and a 54 % in individual. As it can be seen, even having a very conservative threshold to consider an SDG as identified the obtained results are better than with the other models.

In Figure 4.2 it is compared the identified SDGs by the Top2Vec model with the labelled ones. It can be seen how even if there is 36 % of the SDGs not being identified, the model can correctly identify all the types of SDGs. When using the model with the validation dataset the same results are

obtained. There are approximately 45 % of the texts whose SDGs are not being identified, however, there is at least one identified for all the kinds of SDGs.



(a) Training dataset

(b) Validation dataset

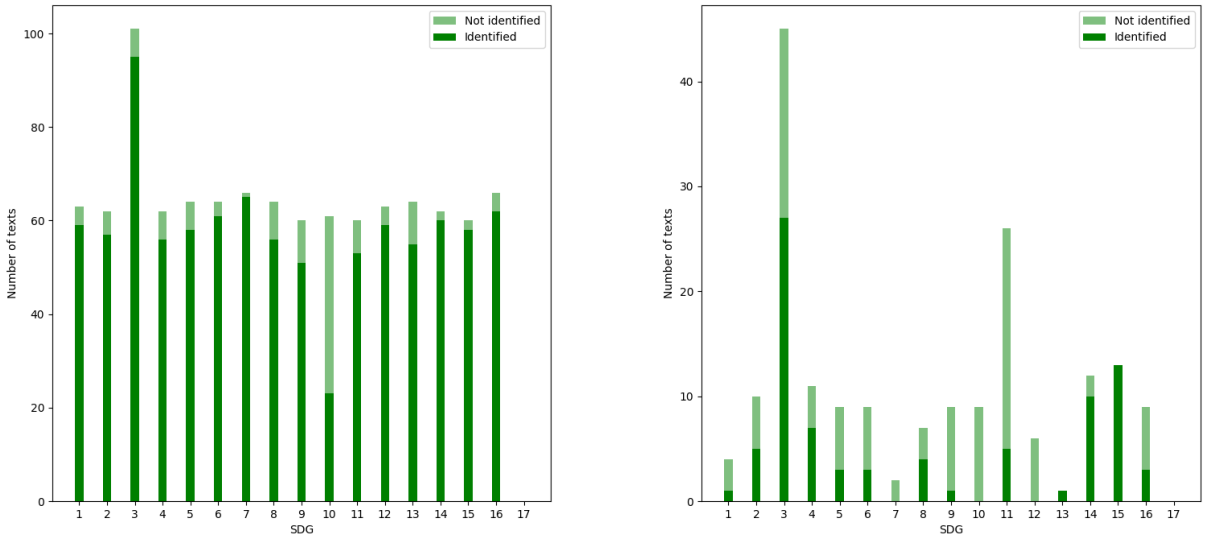
Figure 4.3: Top2Vec model validation comparing the identified SDGs with the previously labelled ones.

4.4 BERTopic

The procedure followed to validate the BERTopic model is the same as the one described with the NMF model (Section 4.1). Firstly it is used the training dataset; the model was trained on this dataset, so it should be able to identify the SDGs perfectly. The model outputs the probability or score of text to be related to the SDGs. In this case, it set 0.2 as a threshold value, so if the score is greater or equal to 0.2 that SDG will be assumed to be identified. This is a very conservative value. The expand factor, in this case, is 1.0.

Using the training dataset, it is obtained a 92.6 % of accuracy for the SDGs set and 89 % for individual SDGs. The former indicates how well are identified all the SDGs that a text is associated with, while the latter indicates how well SDGs are identified. Furthermore, the model can correctly identify the SDGs in the training texts since there are more than 1000 texts. Repeating the process with the validation dataset, it is obtained a 51 % in global and a 54.2 % in individual. As it can be seen, even having a very conservative threshold to consider an SDG as identified the obtained results are better than with the other models. Also, the SDGs in the identified texts are unequivocally identified, that is, the model does not confuse SDGs or identify SDGs erroneously.

In Figure 4.4 it is compared the identified SDGs by the BERTopic model with the labelled ones. In the case of the training files (Figure 4.4a) almost all the SDGs are very well-identified, being the DSG the only exception (less than 50 %). In the case of the validation files (Figure 4.4) some SDGs were not identified (e.g SDG7 or SDG10) but on the contrary, SDG15 was completely identified.



(a) Training dataset

(b) Validation dataset

Figure 4.4: BERTopic model validation comparing the identified SDGs with the previously labelled ones.

Chapter 5

Voting mechanism

In Sections 3.1, 3.2, 3.3 and 3.4 it has been described the NMF, LDA, Top2Vec and BERTopic models respectively. These models have different mathematical backgrounds (i.e. hypothesis and theory) so in practice, the results obtained from all of them should complement each other, they are not redundant. To see if this holds, it is shown in Figure 5.3 the results obtained with the validation procedure for all the models. From the figure, it is depicted that some SDGs that were not identified by some models were correctly identified by others. For example in the SDG8 the NMF and Top2Vec models perform relatively well, however, the LDA could only identify a few of them. Also, in the SDG15 the BERTopic model obtains a 100 % score, while the LDA and Top2Vec only get around a 50 %. It is also to be remarked that SDG10 was not identified by any of the models, so there should be more work on it. In summary, it is reasonable to think that if the models are complementing each other, then it could be designed a voting mechanism that makes use of all the information provided by each model, being more precise and robust at the same time.

In Figure 5.2 it can be seen a scheme explaining the workflow of the voting mechanism, having the following steps:

- First a text to analyse is extracted from the corpus.
- The model is passed to each model separately, and they output the score or probability that the text has with each SDG.
- The score obtained from each model is multiplied by a parameter α.
- All the scores are added together, obtaining a final vector of 1x17.
- If an SDG score is equal to or greater than 0.12 and at least the score of two models (after being scaled) is equal to or greater than 0.1 then the SDG is considered as identified.

From the voting mechanism, it should be remarked that the obtained scores are not normalized in any step, they are always in absolute terms. The reason is to contemplate those cases in which the text may not be associated with any SDG. In this way, the scaling parameters were introduced, since the output of each model was in a different order of magnitude. Although not included in the scheme, the scores are saturated to 0.0 and 0.5 (min and max limits respectively) before being added together. The scaling parameters were adjusted as $1/\beta$ where β was the maximum score obtained during the validation of the model with the training database. Since the models were being tested with the training texts, they should contain the maximum value that the model can output

Once the voting mechanism is finished, then it is also validated with both the training and validation datasets. In the case of the training texts (Figure 5.3a), it obtained a 97.2 % of accuracy, so the

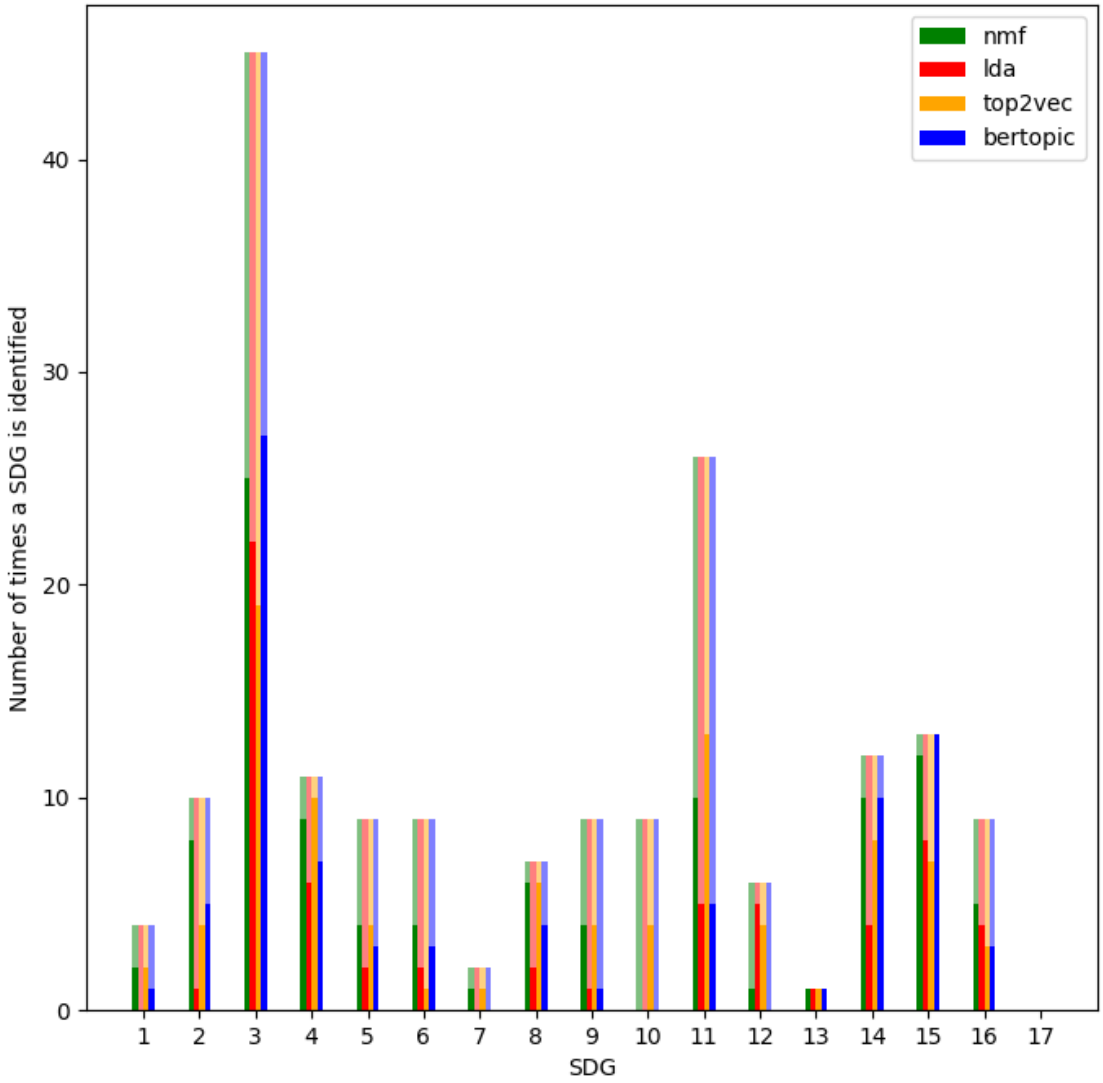


Figure 5.1: Validation of the NMF, LDA, BERTopic and Top2Vec models with the previously labelled dataset (only abstracts). Dark: correctly identified, Light: incorrectly identified.

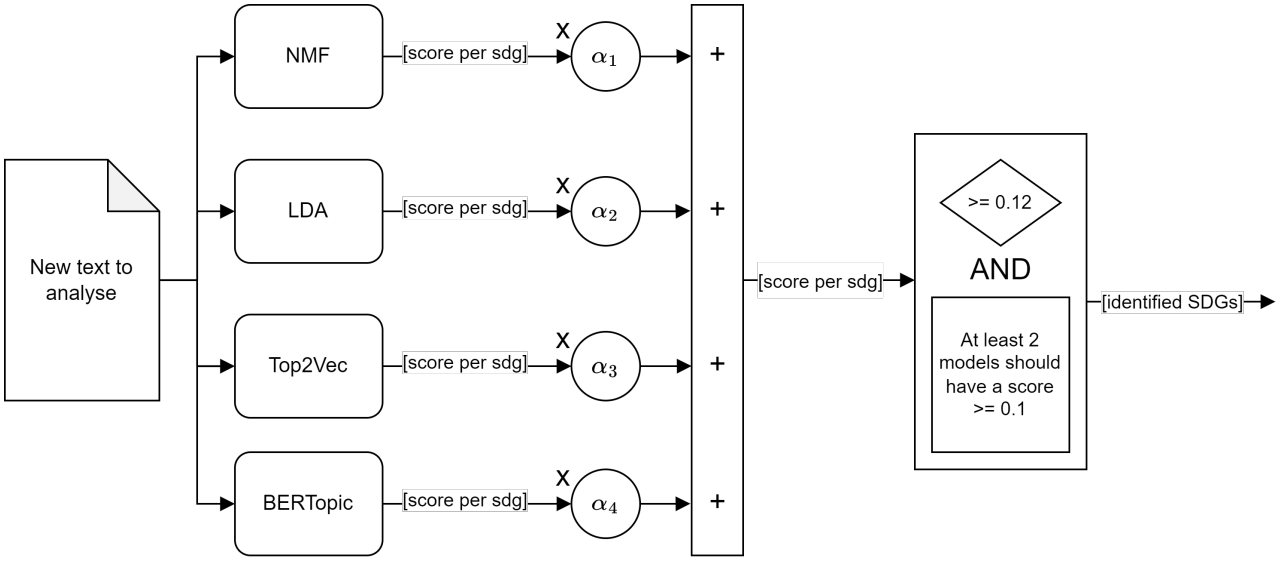
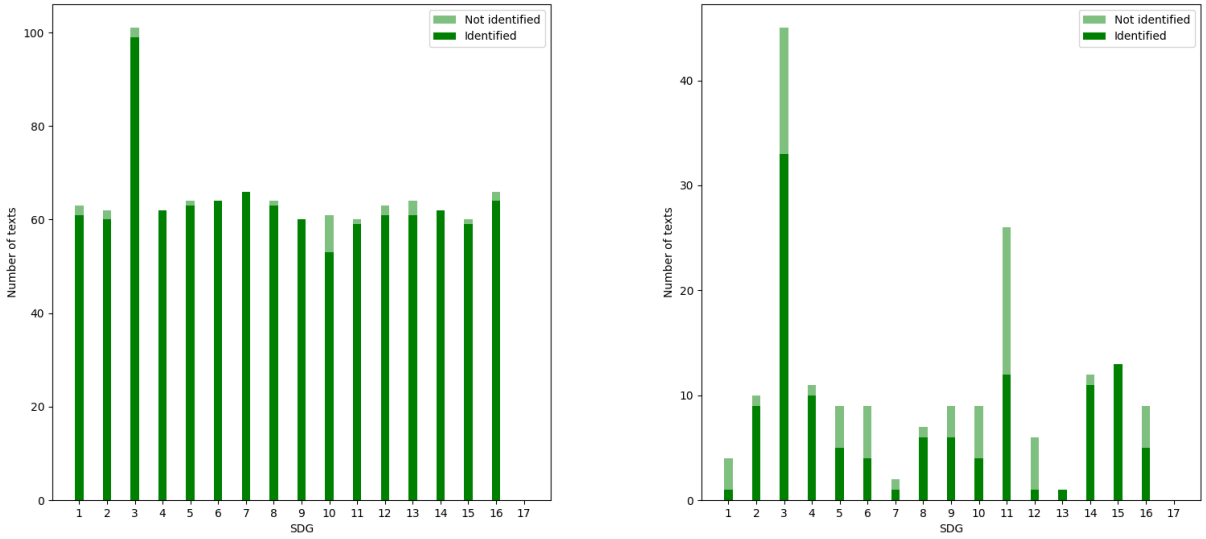


Figure 5.2: Scheme explaining the different stages of the voting mechanism.

results are very promising. There are some SDGs with a 100 % such as the SDG4 or SDG6, however, where the model does not perform well on the SDG10. With the validation texts (Figure 5.3b) it is obtained a 67.5 % of accuracy. It can be seen how the result obtained here has improved from the results obtained with each model separately. Also, the robustness of the model has increased, so it identifies only a few SDGs (instead of being noisy) and those identified are related to the contents of the passed texts. Some SDGs are perfectly identified as SDG15 or SDG3. The results obtained with all the texts are annexed to this work, however, in the following paragraphs, they are shown some of the validation texts with their respective labelled and identified SDGs.



(a) Training dataset

(b) Validation dataset

Figure 5.3: Validation of the voting mechanism with both the training and validation datasets.

Text 1: " this paper examines the use of remote sensing satellite data to predict food shortages among different categories of households in famine-prone areas. normalized difference vegetation index (ndvi) and rainfall estimate data, which can be derived from multi spectral satellite radiometer images, have long been used to predict crop yields and hence famine. this gives an overall prediction of food insecurity in an area, though in a heterogeneous population it does not directly predict which sectors of society or households are most at risk. in this work we use the information on 3094 households across uganda collected between 2004 2005. we describe a method for clustering households in such a way that the cluster decision boundaries are both relevant for improved specificity famine prediction and are easily communicated. we then give classification results for predicting food security status at a household level given different combinations of satellite data, demographic data, and household category indices found by our clustering method. the food security classification performance of this model demonstrates the potential of this approach for making predictions of famine for specific areas and demographic groups.". Labelled SDGs: [2]. Identified SDGs: [0.39:2, 0.13:15].

Text 2: " hypertensive disorders are the leading cause of deaths during pregnancy. risk pregnancy accompaniment is essential to reduce these complications. decision support systems (dss) are important tools for patients accompaniment. these systems provide relevant information to health experts about clinical condition of the patient anywhere and anytime. in this paper, a model that uses the naïve bayesian classifier is introduced and its performance is evaluated in comparison with the data mining (dm) classifier named j48 decision tree. this study includes the modeling, performance evaluation, and comparison between models that could be used to assess pregnancy complications. evaluation analysis of the results is performed through the use of confusion matrix indicators. the founded results show that j48 decision tree classifier performs better for almost all the used indicators,

confirming its promising accuracy for identifying hypertensive disorders on pregnancy. ". Labelled SDGs: [3]. Identified SDGs: [0.42:3, 0.15:13].

Text 3: "multiple studies have been conducted on project listen, an intelligent tutoring system (its) used to analyze educational learning through case analysis of students interactions with its. studies have defined the phenomenon by exploring what happens when/if questions and analyzing these in the context of the specified phenomenon occurrence. while its often focus on student decisions regarding when and how to use the systems resources, we suggest further analysis and monitoring are needed to get the best results from these systems. in this study, we argue that boys interact differently with its than girls. this finding is evident in results from both the bayesian knowledge tracing and learning curve analysis models.". Labelled SDGs: [4]. Identified SDGs: [0.43:4].

Text 4: " produced by the artificial intelligence for ecosystem services (aries) platform (with ess supply defined as carbon storage and flood regulation, and demand specified as recreation and water use). these are then used for (iii) a joint spatial prioritisation of biodiversity and ess employing marxan with zones, laying out the spatial representation of multiple management zones. given the transboundary setting of the danube river basin, we also run comparative analyses including the country level purchasing power parity (ppp)adjusted gross domestic product (gdp) and each countrys percent cover of the total basin area as potential cost factors, illustrating a scheme for balancing the share of establishing specific zones among countries. we demonstrate how emphasizing various biodiversity or ess targets in an ebm model coupling framework can be used to cost effectively test various spatially explicit management options across a multi national case study. we further discuss possible limitations, future developments, and requirements for effectively managing a balance between biodiversity and ess supply and demand in freshwater ecosystems.". Labelled SDGs: [6]. Identified SDGs: [0.32:6, 0.22:15, 0.18:14].

Text 5: "this paper presents the results of attempt to perform modeling of so 2 concentration in urban area in vicinity of copper smelter in bor (serbia), using anfis methodological approach. the aim of obtained model was to develop a prediction tool that will be used to calculate potential so 2 concentration, above prescribed limitation, based on input parameters. as predictors, both technogenic and meteorological input parameters were considered. accordingly, the dependence of so 2 concentration was modeled as the function of wind speed, wind direction, air temperature, humidity and amount sulfur emitted from the pyrometallurgical process of sulfidic copper concentration treatment.". Labelled SDGs: [11]. Identified SDGs: [0.34:11, 0.23:7, 0.21:13].

Text 6: "in line with a long research tradition focused on the use of information and communication technology for development ict4d we explore the role of artificial intelligence ai4d we start with a rather technical review of four of the characteristic traits of deep learning technologies which leads to natural metaphors for international development based on the empirical evidence of 24 case studies we derive four characteristics of the use of ai4d that align with the four technological traits in isolation each one of them presents a plethora of opportunities to contribute to international development especially to the attainment of the sustainable development goals sdgs however in combination they create a clear tension between a looming threat of a hegemonic intelligence indoctrination pushed by global economies of scale and the potential promise to not only honor but to celebrate local diversity with the help of flexible ai designs we conclude that the latter cannot be achieved without an active public policy dialogue on the international level and a determined effort on the national levels especially in developing countries the study provides terminology and concepts to identify and frame the arising discussions". Labelled SDGs: [5]. Identified SDGs: [0.30:9, 0.23:4, 0.13:14, 0.13:12].

As it can be seen from the examples (the score before the topic indicates its importance, going on the scale from 0 to 0.5), in the first texts the labelled SDG was correctly identified (in general, with the highest score) which means that the model is working properly, but also, that it is identifying other SDGs in the texts with which they were not labelled (secondary topics). For example, text 4

it is talking mainly about an efficient method to handle the waste and use of water, however, it also includes some work related to biodiversity so SDGs 14 and 15 are also identified. In the case of text 6, it was labelled with the Gender equality (SDG5), however, it can be seen how this topic is not explicitly included in the text. This is also the case in other texts from the validation database; for this reason, it was assumed as completely valid with a global accuracy of 67.52 %, since there is not enough information in the texts for the model to work properly.

Some of the texts are not correctly identified because they lack explicit words that refer to the SDGs with which they were labelled. For this reason, it is checked whether including more information is beneficial or not for SDGs identification. Apart from the Abstract, it is passed the keywords, introduction, body and conclusion of the papers to the model. These parts are joined together. The results are shown in Figure 5.4, noticing that they are not better than those obtained using only the abstracts. The reason is that some SDGs that were not previously identified now are, but it also occurs vice-versa. Since the models were trained on short texts, it is also checked whether segmentizing the input texts to the model and averaging the scores obtained by the individual segments improve the results or not; the obtained results are almost the same, so there is no advantage in doing that.

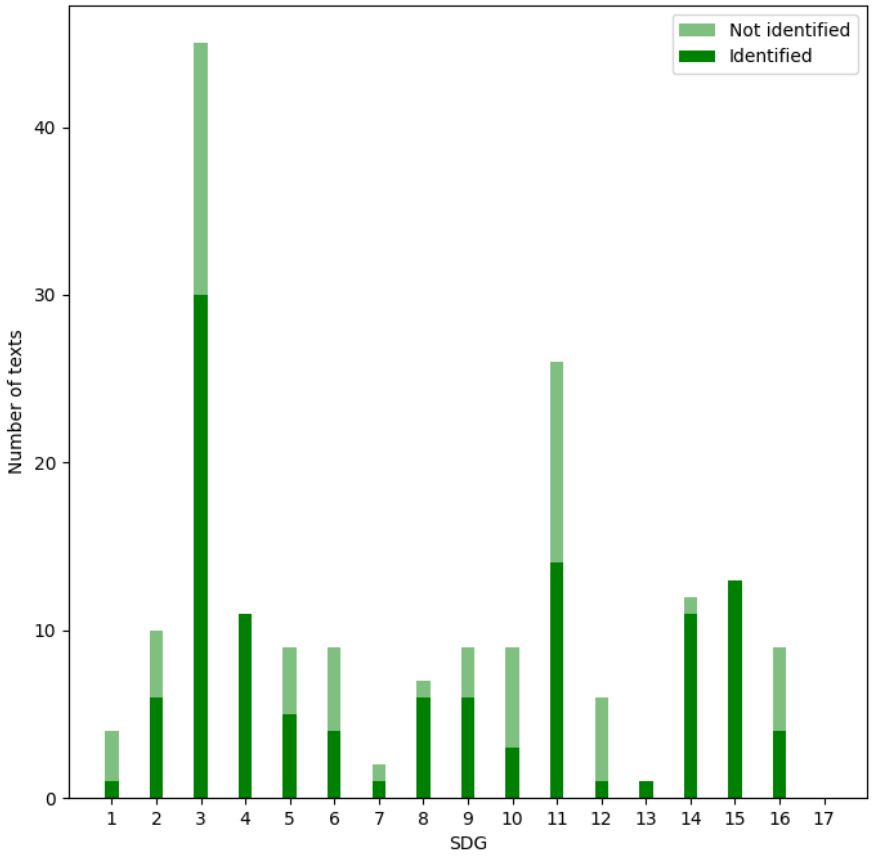


Figure 5.4: Validation of the voting mechanism with the validation dataset in long format.

Finally, it is to be remarked that the models, validation and votation mechanism have already been published in Sánchez et al. [13].

Chapter 6

Results & Discussion

Chapter 2 explained the different datasets that are used in the project. One for training, the other for validation, and the last one for analysis. The last dataset contains abstracts of scientific publications related to aerospace engineering, including extra information such as the title of the work, keywords or year and country of publication. In this chapter, it is analysed this dataset (as part of the work objectives, Section 1.1) obtaining results such as the total weight of each SDG in the corpus or the evolution of the contribution to the SDGs over the years.

Figure 6.1 shows the total weight of each SDG, which is obtained by adding up the scores of all the analysed texts in the corpus. Representing the total weight of each SDG is a more accurate measure than just counting the times that an SDG is present, since then the score is not being considered. It can be noticed how the Affordable and clean energy (SDG7) is the most prominent one, which makes sense since a lot of the texts in the corpus talk about fuels, energy, combustion and renewables. Then they come the Industry, innovation and infrastructure (SDG9), Sustainable cities and communities (SDG11) and Responsible consumption and production (SDG12); it is also reasonable to link the aerospace texts with these SDGs since generally it is being talked about innovation, the infrastructure, how cities could be improved or about ways of using fewer resources and optimizing the waste of materials. Other SDGs are less represented such as the No poverty (SDG1), Zero hunger (SDG2) or Clean water and sanitation (SDG6). The least represented is the Gender equality (SDG5), which again makes sense since generally all the abstracts are technical and they only focus on the description of the study, without talking about these other topics. In the following paragraphs, it is discussed the association of some texts of the dataset with each SDG based on the individual scores of the models.

Text 1: “excess mortality (mort) in china due to exposure to ambient fine particulate matter with aerodynamic diameter 2.5 m (pm2.5) was determined using an ensemble prediction of annual average pm2.5 in 2013 by the community multiscale air quality (cmaq) model with four emission inventories and observation data fusing. estimated mort values due to adult ischemic heart disease, cerebrovascular disease, chronic obstructive pulmonary disease, and lung cancer are 0.30, 0.73, 0.14, and 0.13 million in 2013, respectively, leading to a total mort of 1.3 million. source-oriented cmaq modelling determined that industrial and residential sources were the two leading sources of mort, contributing to 0.40 (30.5%) and 0.28 (21.7%) million deaths, respectively. additionally, secondary ammonium ion from agriculture, secondary organic aerosol, and aerosols from power generation were responsible for 0.16, 0.14, and 0.13 million deaths, respectively. a 30% mort reduction in china requires an average of 50% reduction of pm2.5 throughout the country and a reduction by 62%, 50%, and 38% for the beijing tianjin hebei, jiangsu zhejiang shanghai, and pearl river delta regions, respectively. reducing pm2.5 to the caaqs grade ii standard of 35 g m 3 would only lead to a small reduction in mortality, and a more stringent standard of lt;15 g m 3 would be needed for more remarkable reduction

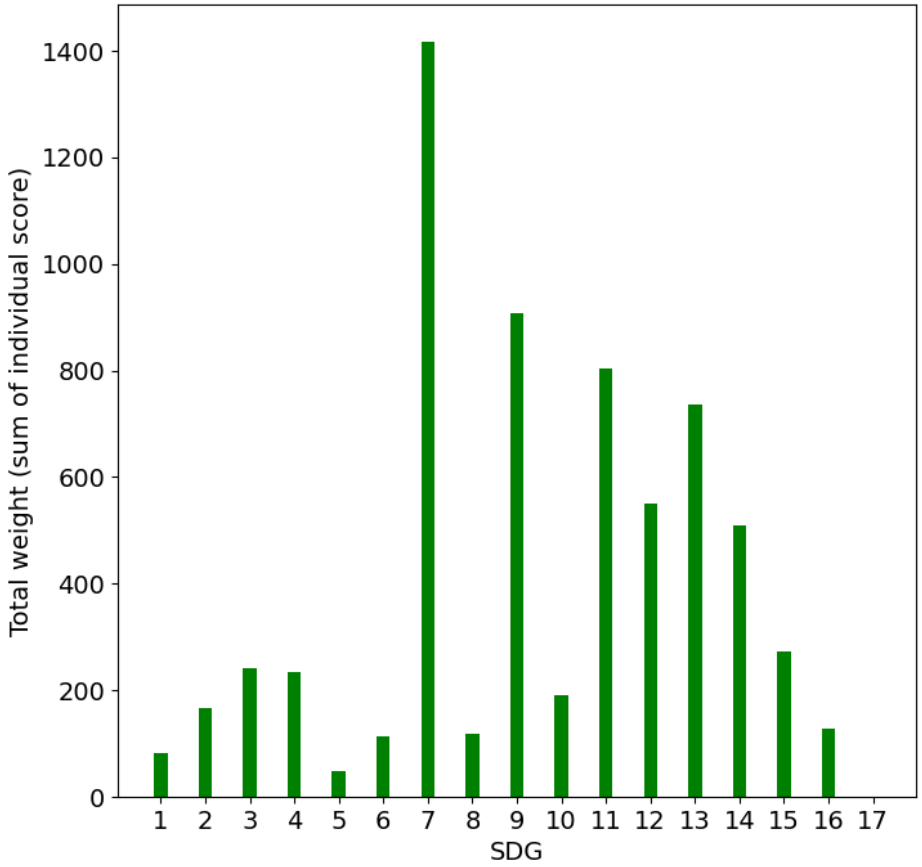


Figure 6.1: Total weight per SDG of the complete Aerospace dataset. The scores obtained individually (i.e. per text) are added together.

of mort.".

This abstract talks about the relation of the mortality rate with the exposure of the population to fine particulate matter. To analyse the results obtained with each model, first, it is calculated an approximation of the top words that each model would mainly associate with each text. They are calculated based on the score that each word has inside the topic (determined during the training) and the score that each text has with the topic. Once all the words are collected, then they are sorted to obtain those words with a higher representation. For this reason, it should be remarked that it is an approximation since it is not considered the original words inside each text that triggered the associations. In Table 6.1 they are shown the top 15 words that each model associates with the text. The NMF, LDA and Top2Vec share the same first two words, talking about health, disease or dying, what is directly related to SDG3 and the contents of the text (mortality, heart disease, lung cancer...). The NMF also identifies some words related to education, school or primary. They are mainly related to SDG4, and one main reason is that the text contains the word "secondary" repeated times. Since the NMF model can not identify polysemy, it assumes that secondary may be part of secondary education, which does not hold in this case. The LDA contains other words such as urban, country, population or city, which makes sense since the text contains some of these words or words with related meanings. In the case of Top2Vec, it only identifies words related to SDG3. BERTopic is not able to capture correctly the topic of the text, since it does not include any words related to health. These results can also be seen in a different representation in Table 6.2. As could be expected from the most important words, the NMF, LDA and Top2Vec identify the SDG3 as the

primary topic, including also the SDG4 (related to education). For this reason, the main SDG in the votation mechanism is SDG3. As secondary topics, they are obtained SDG4 (discussed previously) and SDG11. LDA and Top2Vec include some words related to SDG11 such as city or population, but the main reason behind the confusion is that BERTopic is not working properly, since it identifies SDG11 as the main topic. Finally, the identified SDGs with the votation mechanism are: [0.36:3, 0.20:4, 0.16:11, 0.12:12, 0.12:9].

NMF	LDA	Top2Vec	BERTopic
0.0517:health	0.0075:death	0.1976:dying	0.0076:urban
0.0454:disease	0.0072:disease	0.1896:communicable	0.0064:city
0.0370:death	0.0070:health	0.1875:malaria	0.0036:transport
0.0290:mortality	0.0068:climate	0.1864:mortality	0.0035:disaster
0.0219:communicable	0.0067:water	0.1827:disease	0.0034:housing
0.0211:maternal	0.0055:change	0.1776:diabetes	0.0031:population
0.0203:waste	0.0053:urban	0.1774:infectious	0.0030:slum
0.0198:communicable disease	0.0044:country	0.1768:era	0.0028:climate
0.0178:education	0.0043:child	0.1764:skilled	0.0026:public
0.0177:care	0.0042:climate_change	0.1762:tuberculosis	0.0022:urban population
0.0175:climate	0.0041:rate	0.1756:prevented	0.0021:climate change
0.0160:school	0.0041:city	0.1750:hepatitis	0.0020:change
0.0160:cancer	0.0039:mortality	0.1748:death	0.0020:air
0.0132:primary	0.0038:population	0.1740:infected	0.0019:policy
0.0130:mental	0.0036:country	0.1740:vaccination	0.0018:energy

Table 6.1: Approximation of the top 15 words obtained by each model about the text #1.

SDG	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
NMF	0.00	0.00	0.50	0.15	0.00	0.00	0.06	0.00	0.11	0.00	0.00	0.13	0.12	0.00	0.00	0.00	0.00
LDA	0.00	0.23	0.50	0.25	0.00	0.10	0.07	0.09	0.25	0.00	0.13	0.00	0.08	0.00	0.00	0.06	0.00
Top2Vec	0.00	0.00	0.44	0.41	0.08	0.23	0.18	0.11	0.13	0.00	0.17	0.37	0.10	0.07	0.10	0.00	0.00
BERTopic	0.09	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.33	0.00	0.17	0.00	0.00	0.00	0.00
Mean	0.02	0.06	0.36	0.20	0.02	0.08	0.08	0.05	0.12	0.00	0.16	0.12	0.12	0.02	0.03	0.01	0.00

Table 6.2: Scores associated with each SDG obtained by the NMF, LDA, Top2Vec and BERTopic model in the text #1 .

Text 2: “thanks to the continuous improvement of calculation resources, computational fluid dynamics (cfd) is expected to provide in the next few years a cost-effective and accurate tool to improve the understanding of the unsteady aerodynamics of darrieus wind turbines. this rotor type is increasingly welcome by the wind energy community, especially in case of small size applications and/or non conventional installation sites. in the present study, unique tow tank experimental data on the performance curve and the near wake structure of a darrieus rotor were used as a benchmark to validate the effectiveness of different cfd approaches. in particular, a dedicated analysis is provided to assess the suitability, the effectiveness and the future prospects of simplified two dimensional (2d) simulations. the correct definition of the computational domain, the selection of the turbulence models and the correction of simulated data for the parasitic torque components are discussed in this study. results clearly show that,(only) if properly set, two dimensional cfd simulations are able to provide with a reasonable computational cost an accurate estimation of the turbine performance and also quite reliably describe the attended flow field around the rotor and its wake”.

This abstract is talking about how the improvements and evolution of the computation resources will allow in the future to obtain better and more accurate results when simulating the flow in darrieus wind turbines. First, it is shown in Table 6.3 the first 15 words that each model has identified with the text. The first word present in NMF, LDA and BERTopic is “energy”, since the text is mentioning several times the same word, and it also includes others such as wind, which is associated with renewables too. In the case of the NMF, there are also other interesting

words such as innovation, development, renewable energy and energy efficiency, which in this case are also totally related to the text and SDG7. The LDA identifies some words that are not related to the text, such as woman, child or food. In the case of BERTopic, almost all the words are related to SDG7, including also some of them related to SDG12 such as material, waste or consumption. The global scores can be seen in Table 6.4, which reflects the same ideas that were previously mentioned. It can be seen how the four models identify as the primary topic the SDG7 (being the main topic of the texts), but also include other topics such as the SDG11 (whose relationship with the text is arguable) or the SDG12. In this last case, it can make sense, since the text is arguing about the use of simplified 2D, which uses fewer computation resources and then, there exists a more responsible consumption. In this case, the models are misled to identify the SDG14, since the texts include several times the word fluids, wake or fluid dynamics. The models should be improved in this scenario. Finally, the identified SDGs with the votation mechanism are: [0.31:7, 0.19:14, 0.17:12].

NMF	LDA	Top2Vec	BERTopic
0.0616:energy	0.0080:energy	0.2391:paper	0.0139:energy
0.0325:policy	0.0062:country	0.2360:approach	0.0043:electricity
0.0184:chapter	0.0062:marine	0.2358:discus	0.0037:renewable
0.0165:paper	0.0059:ocean	0.2309:evaluation	0.0033:fuel
0.0158:waste	0.0059:country	0.2258:analysis	0.0032:renewable energy
0.0156:government	0.0044:woman	0.2256:policy	0.0030:efficiency
0.0154:oecd	0.0041:sustainable	0.2233:explores	0.0030:technology
0.0145:innovation	0.0038:including	0.2232:examines	0.0029:energy efficiency
0.0134:development	0.0037:global	0.2214:chapter	0.0028:access
0.0131:digital	0.0031:child	0.2210:strategic	0.0027:carbon
0.0130:renewable	0.0031:development	0.2185:effectiveness	0.0014:climate
0.0120:electricity	0.0031:manufacturing	0.2168:highlight	0.0014:disaster
0.0112:renewable energy	0.0029:sustainable	0.2167:summarises	0.0014:material
0.0110:energy efficiency	0.0028:food	0.2161:organisation	0.0014:waste
0.0107:support	0.0027:developing	0.2122:attractiveness	0.0011:consumption

Table 6.3: Approximation of the top 15 words obtained by each model about the text #2.

SDG	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
NMF	0.00	0.00	0.00	0.00	0.00	0.06	0.25	0.00	0.08	0.00	0.08	0.11	0.00	0.07	0.00	0.00	0.00
LDA	0.00	0.17	0.05	0.25	0.00	0.00	0.16	0.09	0.19	0.17	0.00	0.10	0.08	0.32	0.11	0.14	0.00
Top2Vec	0.00	0.00	0.00	0.22	0.00	0.00	0.50	0.00	0.00	0.00	0.45	0.33	0.50	0.38	0.00	0.00	0.00
BERTopic	0.00	0.00	0.00	0.00	0.00	0.00	0.34	0.00	0.00	0.00	0.00	0.13	0.00	0.00	0.00	0.00	0.00
Mean	0.00	0.04	0.01	0.12	0.00	0.02	0.31	0.02	0.07	0.04	0.13	0.17	0.15	0.19	0.03	0.04	0.00

Table 6.4: Scores associated with each SDG obtained by the NMF, LDA, Top2Vec and BERTopic model in the text #2.

Text 3: “scramjet is found to be the efficient method for the space shuttle. in this paper, numerical simulation is performed to investigate the fundamental flow physics of the interaction between an array of fuel jets and multi air jets in a supersonic transverse flow. hydrogen as a fuel is released with a global equivalence ratio of 0.5 in presence of micro air jets on a flat plate into a mach 4 crossflow. the fuel and air are injected through streamwise aligned flush circular portholes. the hydrogen is injected through 4 holes with 7dj space when the air is injected in the interval of the hydrogen jets. the numerical simulation is performed by using the reynolds averaged navier stokes equations with menters shear stress transport (sst) turbulence model. both the number of air jets and jet to freestream total pressure ratio are varied in a parametric study. the interaction of the fuel and air jet in the supersonic flow present extremely complex feature of fuel and air jet. the results present various flow features depending upon the number and mass flow rate of micro air jets. these flow features were found to have significant effects on the penetration of hydrogen jets. a variation of the number of air jets, along with the jet to freestream total pressure ratio, induced a variety of flow structure in the downstream of the fuel jets. ”.

This text is talking about the simulation of scramjet and the associated turbulence with computational fluid dynamics. The global scores can be seen in Table 6.6. In the table 6.5 they are shown the main words, being in this case mostly related to energy and electricity. This makes sense since the text contains words such as hydrogen, air, or fuel. Other included words are climate, material and waste, which should be related to "efficient". In Table 6.6 they are shown the scores of each model. The main identified SDG is the 7 since the text is mainly talking about energy and jet fuel. SDG11 is also identified, which does not correlate with the contents of the text, but it does SDG9 since it talks about innovation and analysis of the flow structure. The identified SDGs with the votation mechanism: [0.37:7, 0.23:11, 0.21:9, 0.19:12].

NMF	LDA	Top2Vec	BERTopic
0.0571:water	0.0151:water	0.1535:slum	0.0051:energy
0.0273:energy	0.0114:energy	0.1490:urban	0.0016:electricity
0.0242:urban	0.0059:sanitation	0.1437:urbanization	0.0014:renewable
0.0199:policy	0.0058:country	0.1411:breathing	0.0014:climate
0.0195:city	0.0040:country	0.1356:city	0.0013:disaster
0.0113:chapter	0.0035:access	0.1343:settlement	0.0013:material
0.0110:water resource	0.0033:renewable	0.1318:unplanned	0.0013:waste
0.0101:paper	0.0032:consumption	0.1312:sprawl	0.0012:fuel
0.0098:slum	0.0030:drinking	0.1282:space	0.0012:renewable energy
0.0095:government	0.0029:climate	0.1259:ingenuity	0.0011:efficiency
0.0094:oecd	0.0029:marine	0.1256:distance	0.0011:technology
0.0094:management	0.0028:electricity	0.1243:particulate	0.0011:energy efficiency
0.0088:transport	0.0028:technology	0.1242:convenient	0.0010:access
0.0088:innovation	0.0028:fuel	0.1239:incubator	0.0010:carbon
0.0088:health	0.0028:ocean	0.1229:monthly	0.0010:consumption

Table 6.5: Approximation of the top 15 words obtained by each model about the text #3.

SDG	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
NMF	0.00	0.00	0.13	0.00	0.00	0.00	0.17	0.00	0.07	0.06	0.31	0.08	0.00	0.00	0.00	0.00	0.00
LDA	0.00	0.06	0.21	0.16	0.06	0.00	0.37	0.00	0.50	0.00	0.13	0.08	0.10	0.00	0.00	0.15	0.00
Top2Vec	0.00	0.00	0.11	0.00	0.00	0.06	0.50	0.00	0.28	0.00	0.50	0.50	0.30	0.15	0.06	0.00	0.00
BERTopic	0.00	0.00	0.00	0.00	0.00	0.00	0.45	0.00	0.00	0.00	0.00	0.12	0.00	0.00	0.00	0.00	0.00
Mean	0.00	0.04	0.01	0.12	0.00	0.02	0.31	0.02	0.07	0.04	0.13	0.17	0.15	0.19	0.03	0.04	0.00

Table 6.6: Scores associated with each SDG obtained by the NMF, LDA, Top2Vec and BERTopic model in the text #3.

Text 4: "recharge assessment is of critical importance for groundwater resources evaluation in arid/semiarid areas, as these have typically limited surface water resources. there are several models for water balance evaluation. one of them is wetspass, which can simulate spatially distributed recharge, surface runoff, and evapotranspiration for seasonally averaged conditions. this paper presents a modified methodology and model, wetspass m, in which the seasonal resolution is downscaled to a monthly scale. a generalized runoff coefficient was introduced, enabling runoff estimation for different land use classes. wetspass m has been calibrated and validated with observed streamflow records from black volta. base flow from simulated recharge was compared with base flow derived via a digital filter applied to the observed streamflow and has shown to be in agreement. previous studies have concluded that for this basin, small changes in rainfall could cause a large change in surface runoff, and here a similar behavior is observed for recharge rates. an advantage of the new model is that it is applicable to medium and large sized catchments. it is useful as an assessment tool for evaluating the response of hydrological processes to the changes in associated hydrological variables. since monthly data for streamflow and climatic variables are widely available, this new model has the potential to be used in regions where data availability at high temporal resolution is an issue. the spatial temporal

characteristics of the model allow distributed quantification of water balance components by taking advantage of remote sensing data.". Identified SDGs: [0.34:6, 0.19:14, 0.13:13].

Text 5: "as a fast and efficient short distance transportation means, the subway line has been built and expanded in an increasing number of cities. the pressure in the tunnel fluctuates significantly while metro trains pass. this kind of pressure may damage the equipment and workers in the tunnel. considering that, the metro train does not have airtightness, and that pressure can spread inside the vehicle, passengers in the vehicle would be directly affected by the alternating aerodynamic pressure, which causes discomfort to passengers. this phenomenon is exacerbated at high speeds. therefore, it is important to estimate the aerodynamic alternating pressure generated by the metro train in the tunnel before construction. in this study, the aerodynamic performance of a metro train running between two adjacent platforms in a tunnel was simulated by using fluent. in this work, the effects of acceleration and speed of the metro train, and of platform spacing, on the alternating pressure on the train and in the tunnel are studied. in the analysis of the impact of train acceleration, the pressure change inside a passenger train in a 1 s timespan was used to evaluate the comfort of passengers. maximum and average p (pressure changes in amplitude) shows an exponential relationship with a (acceleration), vc (constant speed) and lplatform (platform spacing), especially the p measured on tunnel surface. the fluctuation of the train surface pressure is more intense than that of the tunnel. the pmin (minimum pressure) on the train surface and in the tunnel is not affected by the acceleration of the train, but it is mainly related to the highest train speed in the tunnel. when the platform spacing is higher than 1500 m, pmax, pmin, and p in the tunnel and on the train surface showed little change. these findings contribute not only to the design of the metro train and tunnel system, but also to the guidance of the metro train operation.". Identified SDGs: [0.35:11, 0.14:14, 0.13:15].

Text 6: "spacecraft and satellite are susceptible to aerothermoelastic flutter instability, which may jeopardize the mission of the spacecraft and satellite. this kind of instability may result from the coupling of the thermal radiation from the sun and the elastic deformations of aeronautical components. as a first endeavor, the aerothermoelastic flutter and buckling instabilities of functionally graded carbon nanotube reinforced composite (fg cntrc) cylindrical shell under simultaneous actions of aerodynamic loading and elevated temperature conditions are investigated. the formulations are derived according to the first order shear deformation theory, donnell shell theory in conjunction with von karman geometrical nonlinearity. thermomechanical properties are assumed to be temperature dependent and modified rule of mixture is used to determine the equivalent material properties of the fg cntrc cylindrical shell. the quasi steady krumhaars modified piston theory by taking into account the effect of panel curvature, is used to determine the aerodynamic pressure. the nonlinear dynamic equations are discretized in the circumferential and longitudinal directions using the trigonometric expression and the harmonic differential quadrature method, respectively. effects of various influential factors, including cnt volume fraction and distribution, boundary conditions, geometrical parameters, thermal environments, freestream static pressure and mach number on the aerothermoelastic instabilities of the fg cntrc cylindrical shell are studied in details. it is found that temperature rise has a significant effect on the aerothermoelastic flutter characteristics of the fg cntrc cylindrical shell. it is revealed that cylindrical shells with intermediate cnt volume fraction have intermediate critical dynamic pressure, while do not have, necessarily, intermediate critical buckling temperature. it is concluded that the critical circumferential mode number (mcr) corresponding to the minimum critical dynamic pressure, depends not only on the radius to thickness ratio but also on the distribution of the cnts.". Identified SDGs: [0.29:13, 0.15:14, 0.14:11].

From the texts, it can be said that:

- Some SDGs are very well-identified and they are related to the texts. That is the case of the Good health and well-being (SDG3) with the Text 1, Affordable and clean energy (SDG7) with the Text 3, Clean water and sanitation (SDG6) with the Text 4 or the Sustainable cities and communities (SDG11) with the Text 5.

- Generally, if the text is related to one SDG then the corresponding score will also be high. The scale of the scores is from 0 to 0.5, so a score very close to 0.5 means that the text is very related to that SDG. Then they also noticed some SDGs to which the texts are related, and they also have some presence in the text but they are usually less related (i.e. less accurate).
- Some texts that are associated with some SDGs based on the text information, such as Text 6 with the Climate action (SDG13). The text is talking about aerothermoelastic instabilities of spacecraft, so it mentions words such as carbon, thermal, temperature, and environment which makes the model wrongly assume that the text is related to SDG13. To solve this issue, the model should be trained on more specific and context-related texts, instead of being trained only on generic and SDGs-related texts.

Figure 6.2 shows the evolution of the total weight of SDG7, SDG9, SDG11 and SDG13 from 2017 to 2021. They are only shown these SDGs because they are the most representative (Figure 6.1). The differences between years are slight, and it does not show any tendency. For example, from 2017 to 2019 the contribution to the SDG11 increases, but then it decreases again until 2021. In the case of SDG9, it is the same. It is interesting to see the same order of magnitude (contribution) among years since each year has 2000 different texts.

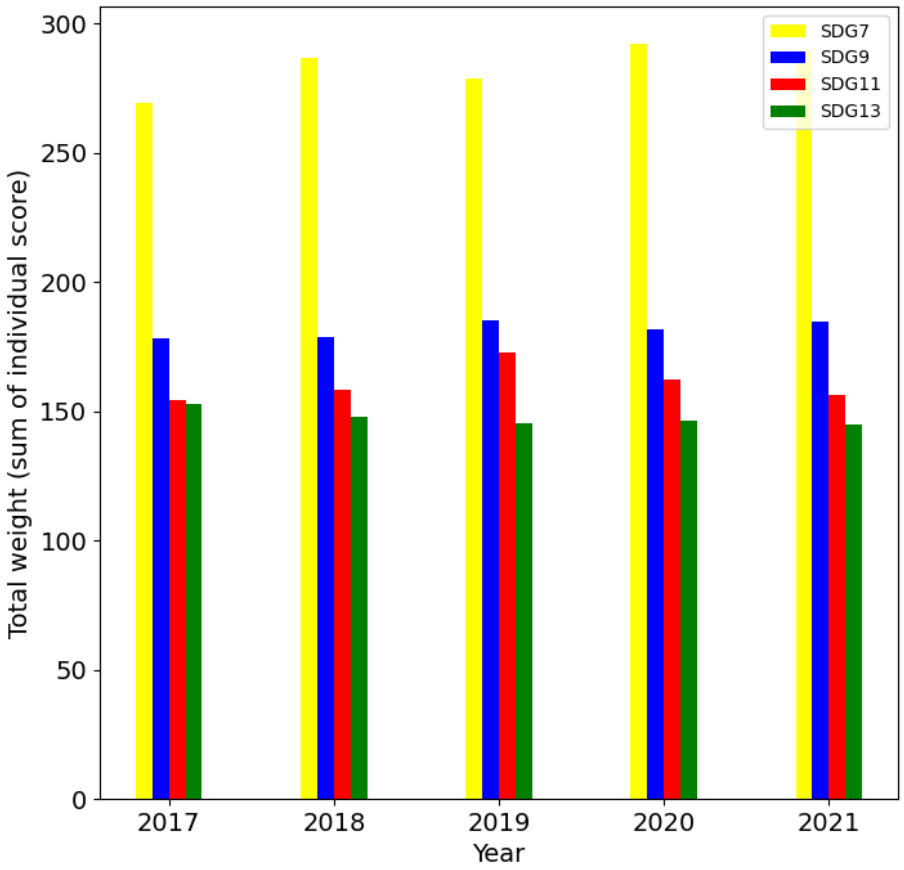


Figure 6.2: Evolution of the total weight of SDG7, SDG9, SDG11 and SDG13 from 2017 to 2021.

Figure 6.3 shows the comparison of the contribution to each SDG by those papers with citations lower and higher than the median. In this case, the median is 8. It can be seen that both parts contribute approximately the same order of magnitude to the SDGs. Still, there exist some differences

for example those papers with lower citations contribute more to the SDG9 and SDG10, however, those with higher have a significantly higher contribution to the SDG7 and SDG3.

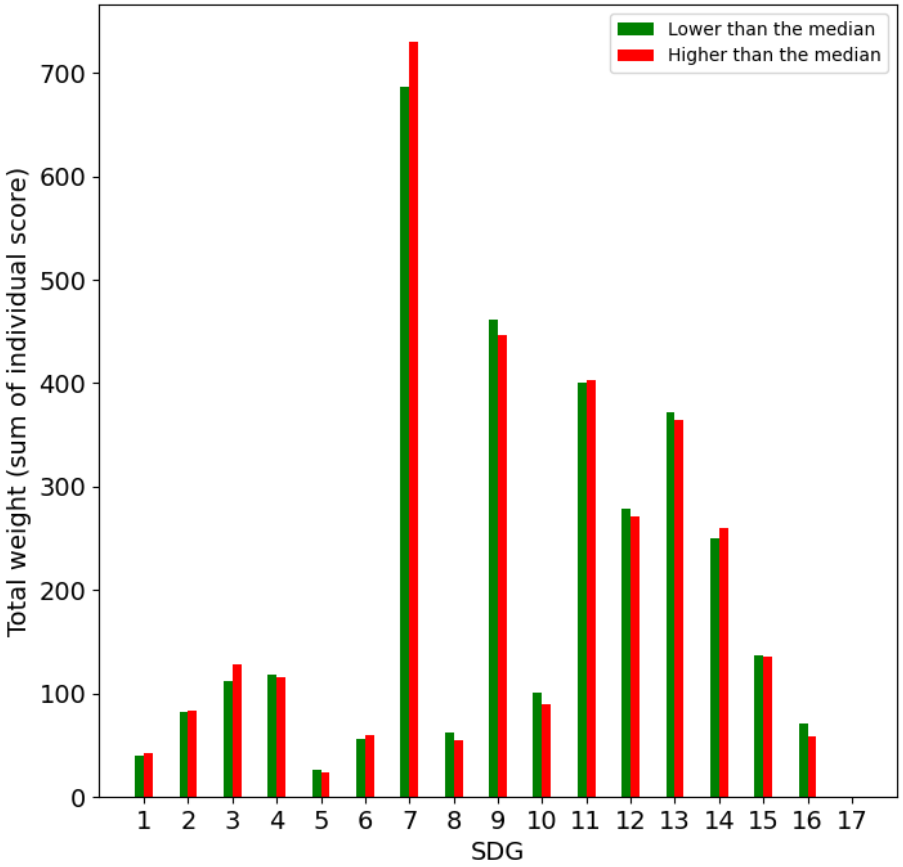


Figure 6.3: Comparison of the contribution to each SDG by those papers with citations lower and higher than the median, being the median of 8.

Chapter 7

Conclusions & Future Work

In this work, it has been presented a complete AI-based model for the identification of those SDGs related to a given text. Firstly, the datasets were introduced. It has been shown that they have a different mathematical nature, so each one of them has indeed different strengths. For example, the NMF model learns more incoherent topics than LDA, however, it identifies clearly the SDGs associated with one text without introducing noise to the other SDGs. On the other side, LDA is able to create more descriptive topics (i.e. topics easier to understand by humans), but LDA works worse than NMF when the texts of the corpus are small, as is the case here.

Both NMF and LDA models rely on the Bag of Words (BOW) assumption, so they do not use semantic information from the texts (the order is meaningless). On the contrary, Top2Vec and BERTopic are Transformers-based models, so they transform the words and documents into a vector before clustering them. For that reason, they perform better, since the texts used in the dataset are small (contain little information) and they are able to understand more complex things such as polysemia. However, they are not perfect either. The greatest weakness of both Top2Vec and BERTopic is that they assume that each document is mainly related to a unique topic (during topic clusterization), so it does not reflect the case since most of the texts are actually related to more than one SDG (i.e. more than one topic).

The NMF and LDA models have been traditionally used for years since they were published more than twenty years ago. They have shown very good results, being able to correctly identify the texts in the training dataset, and partly the texts in the validation dataset. They perform well when the texts are actually related to the SDGs and specific words are explicitly mentioned in them since they only used the information from the training dataset (i.e. they are not pre-trained, models). This makes the task more difficult since a lot of the texts in the aerospace datasets include words such as carbon, fluids or energy that might mislead these models to identify those texts with SDGs such as SDG13, SDG6 or SDG7. Nevertheless, they are simple models that can be trained and used very fast, so they should always be considered as an option for gaining insight into the problem and the datasets.

The Top2Vec and BERTopic are more advanced models, and they perform better than NMF and LDA. As explained before, they also have some weaknesses, and they are also more difficult (and expensive) to understand the theory, optimize the parameters and train. The great advantage of these models is that they are internally split into several stages, which allows them to keep up to date with new releases and methods since they can be updated independently from each other. This is a great asset since each year more and more models of Natural Language Processing (NLP) are published. After training them, it was depicted a voting mechanism that was able to combine the information from all of the individual models. It has proved to be extremely useful, potentiating both the accuracy and robustness of the results. Also, it compared the use of full texts against using only abstracts, but

there were no improvements.

Finally, was analysed the Aerospace dataset. They obtained several insights such as that the papers mostly contributed to the SDGs SDG7, SDG9, SDG11 and SDG13 or that the papers with lower citations contributes in the same order of magnitude to the SDGs as the papers with higher citations (with some exceptions, such as the SDG3 or SDG7). Generally, the model was able to identify correctly the main topic of the texts (i.e. the topic with the highest score) but failed to identify the secondary ones in most of the cases. For this reason, the following tentative lines of future work are proposed:

- Improve the model accuracy and robustness by including a fifth model in the voting process.
- Improve the model accuracy by providing technical texts related to aerospace engineering.
- Increase the length of the Aerospace texts by including other sections such as keywords.
- Improve the training database by providing texts related to SDGs but from different technical backgrounds.
- Increase the confidence levels used to identify the SDGs once the model gets more robust.

Chapter 8

Specifications

In this chapter, they have defined the specifications or conditions that should be guaranteed in order to proceed with the current work. Firstly, they have specified the conditions related to the health care of the worker. They are mostly related to the ambient and office conditions that allow the worker to work without assuming any risk. Then, they have defined the requirements of both software and hardware that will allow the worker to complete the work.

8.1 Office specifications

According to Law 31/1995, the office space should comply with some specifications in order to guarantee the prevention of occupational hazards. Concretely, in the *Real Decreto 488/1997, 14 april* they specified the minimum safety conditions when the worker is exposed to monitors. Moreover, in the *Real Decreto 486/1997, 14 april* they specified the minimum safety conditions and health in the workspaces.

Equipment

The general rule is that the usage of electronic equipment should never imply a risk to the health of the worker. The different components of the equipment and their respective specifications are as follows:

- Screen: the letters on the screen should be correctly defined and legible. The image should be stable and eye-comforting. Based on the worker's needs, it should be possible to adjust the screen height and orientation.
- Keyboard: it should be independent of the screen in order to allow the worker to adopt a comfortable configuration and avoid overloading the joints or other parts of the body. The keyboard should have legible characters on it.
- Office desk: it should be large enough to allow having the screen, keyboard and other required documents on it. Also, the worker should be able to lay the arms and hands on the desk when using the keyboard.
- Office chair: it should be stable and with adjustable height and reclining

Environment

The work environment is crucial for the correct development of the tasks, as well as to guarantee the health of the worker and prevent occupational hazards. The environment is subdivided in:

- Workspace: the workspace should be large enough and be conditioned such that it allows for movements and posture adaptation.
- Illumination: both general illumination and office lamps should give enough levels of illumination to avoid the visual fatigue of the worker. The disposition of the lights should avoid glaring.
- Glare: in case the sources of natural or artificial light generate glares on the worker screen, they should be mitigated with an external device.
- Noise: the noise in the work environment should not perturb the attention of the worker and the conversations. The equipment present in the environment should be designed to take into consideration this aspect.
- Heat: the equipment in the environment should not generate enough heat to perturb the workers in it.
- Emissions: the radiation emitted by the equipment in the environment (out of the visible spectrum) should be below the allowed limits.
- Humidity: it should be kept at acceptable levels.

Computer and worker interrelation

Before acquiring the required programmes and equipment, they should be considered the following factors:

- The programme should be designed according to the assigned task.
- The programme should be able to adapt to the user's knowledge.
- The programme should give insights about its development
- The programme should give the information according to the user's speed and format.

Constructive conditions

The workspace should comply with some requirements related to its time of construction or reform:

- Design and constructive characteristics: they should be safe in case of a slip, drop, shock or impact against an object and collapse or detachment of objects against the workers.
- Emergency situations control: it should have emergency exits correctly indicated in case of emergency. It should also have all the required systems to guarantee the prevention, alert and extinction in the event of a fire.
- Access: access to the space should always be available and in good condition, apart from having walking lines, work spots and toilets adapted for disabled people.

Arrangement, cleaning and maintenance

The entrances, exits and walking lanes should be designed to allow the correct evacuation of the building. In this way, they should be free from obstacles and have the correct indications. The infrastructure that will be used during working hours should be cleaned periodically and the maintenance tasks should be guaranteed when required (also, these tasks should not imply any risk). In the case of using a ventilation system, it should be kept in good condition and should have an alert system that indicates a fault condition in case of being a risk to the health of the workers.

Cleaning services and resting areas

The workspaces should provide the worker with the following services:

- Drinking water. The workspaces will provide the worker with enough drinking water and it should be totally accessible. All possible causes of water contamination should be avoided.
- Showers, changing rooms and toilets. Showers and changing rooms are only strictly required in those places where the worker needs to change clothes or do physical exercise.
- Resting areas. Based on the number of workers or type of work, it should be made available to the workers in some resting areas. These spaces should have enough capacity for all the workers and include the necessary amenities.

8.2 Software & Hardware requirements

This section contains all the information associated with the hardware and software required to perform the training, validation and analysis parts of the project. From a hardware point of view, it is not required a high computational performance computer. If possible, it could be used a Graphics Processing Unit (GPU) to accelerate the training phase but it is not mandatory. One of the objectives of the project (Section 1.1) was to develop models that could be trained and used fast, with low computational resources. From the software perspective, it is only used Visual Studio Code (Free software) to develop the algorithms, train the models and perform the validation and analysis. GitHub is used as source control GitHub Desktop.

Hardware conditions

For hardware it has only been used a laptop (Vivobook 15 X513) with the following specifications:

- Processor: 11th Gen Intel(R) Core(TM) i7-1165G7. Quadcore 2.80 GHz and turbo up to 4.7 GHz.
- Graphics: Intel Iris X Graphics
- RAM: 16 GB DDR4.
- Storage: 512 GB M.2 NVMe PCIe 3.0 SSD.

Software conditions

As mentioned before, it has only been used Visual Studio Code for all the phases of the project: parsing and analysis of texts, algorithm development, models training, models validation and analysis and plotting of the results. It is open-source. GitHub Desktop has been used as source control (open-source).

Chapter 9

Budget

This chapter describes the estimated budget associated with the work, had it been done in a private company. For that purpose the work is divided into different stages, counting the number of hours dedicated to each one, as well as the human and computational resources.

9.1 Phases of the project

The current project can be divided into the following phases:

- Phase 1: preparation of the training and validation dataset. All the texts are downloaded from the UN page and they are parsed accordingly. The texts are checked manually and they are uniform.
- Phase 2: bibliographic revision of the current state of the art and related works. Decide which NLP models would fit most of the application, download them and perform the first test to guarantee that everything goes perfect.
- Phase 3: implementation of the models. They are trained and optimized based on their private configuration parameters.
- Phase 4: validation of the models. Analysis of results.
- Phase 5: preparation and analysis of the aerospace dataset. Results are extracted and conclusions are drawn.
- Phase 6: writing of the thesis document and preparation of the presentation.

9.2 Dedicated resources

The resources that have been used during the project can be separated into three groups: human resources, equipment and software. Table 9.1 shows the associated used resources for each group.

Dedicated resources		
Human	Equipment	Software
x1 engineer	Laptop	Visual Studio Code
x1 doctorate		GitHub Desktop

Table 9.1: Dedicated resources to the project separated into groups.

9.3 Costs breakdown

This Section describes the costs associated with each of the groups of Section 9.2.

Human resources

In this project, they are involved both a junior engineer (the author) and a doctorate (the tutor). The foremost is responsible for doing all the work associated with the project, while the last one is responsible for the supervision and guidance of the engineer. The salary is estimated based on their respective positions. The number of hours dedicated to each phase as well as the total cost per resource can be seen in Table 9.2.

	Phase							
Human resource	1	2	3	4	5	6	Cost per hour[€/h]	Cost [€]
Engineer	110	75	135	145	72	84	15	9315
Doctorate	7	7	4	5	3	4	30	900
							Total Cost	10215

Table 9.2: Number of hours that the human resources dedicate to each phase and the total cost.

Equipment

The costs associated with the equipment can be divided into three groups: electronic equipment, office material and place of work. In this case, the office material and the place of work have a cost of zero, because it was not required to buy anything for the office and the place of work was the Universitat Politècnica de Valencia. About the electronic equipment, as mentioned before, it has only been used as a laptop. It was acquired in 2022 with a cost of 700 €, so the cost per year (assuming 5 years of amortization) is 140 €. All the costs are shown in Table 9.3.

Resource	Cost [€]
Place of work	0
Office material	0
Laptop	140
Total Cost	140

Table 9.3: Costs of the project associated with the equipment.

Software

The software programmes are specified in Section 8.2, being in this case Visual Studio Code and GitHub Desktop. Both of them are free software, so they do not have associated costs.

Total cost

Table 9.4 shows the total cost of the project (net cost), considering an additional cost of 15 % and 6 % of the industrial benefit. It is also required to add the IVA (21 %) to the net cost.

Expense		Cost [€]
Human resources		10215
Equipment		140
Software		0
Net cost		**10355**
Additional expenses	Percentage [%]	Cost [€]
Indirect cost	15	1553.25
Industrial benefit	6	621.3
Cost without IVA		**12529.55**
IVA	21%	2631.20
	Total Cost	**15160.76**

Table 9.4: Summary of the costs associated with the project.

As shown in Table 9.4 the total budget required for the present project is THIRTEEN THOUSAND FOUR HUNDRED NINETY-ONE EURO WITH SIXTY-EIGHT CENTS.

Bibliography

[1] D. Angelov. Top2vec: Distributed representations of topics. *arXiv preprint arXiv:2008.09470*, 2020.

[2] D. Blei, A. Ng, and M. Jordan. Latent Dirichlet Allocation. volume 3, pages 601–608, 01 2001.

[3] R. Egger and J. Yu. A Topic Modeling Comparison Between LDA, NMF, Top2Vec, and BERTopic to Demystify Twitter Posts. *Frontiers in Sociology*, 7, 2022.

[4] M. Grootendorst. Bertopic: Neural topic modeling with a class-based TF-IDF procedure. *arXiv preprint arXiv:2203.05794*, 2022.

[5] A. Hajikhani and A. Suominen. The Interrelation of Sustainable Development Goals in Publications and Patents: A machine learning approach. 2021.

[6] S. Kapadia. Evaluate Topic Models: Latent Dirichlet Allocation (LDA). *Towards Data Science*, 2019.

[7] M. Kostetckaia and M. Hametner. How Sustainable Development Goals interlinkages influence European Union countries' progress towards the 2030 Agenda. *Sustainable Development*, 2022.

[8] D. Kuang, J. Choo, and H. Park. Nonnegative Matrix Factorization for Interactive Topic Modeling and Document Clustering. pages 215–243, 10 2015.

[9] M. LaFleur. Art is long, life is short: An SDG Classification System for DESA Publications. 2019.

[10] Q. Le and T. Mikolov. Distributed Representations of Sentences and Documents. *CoRR*, abs/1405.4053, 2014.

[11] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781*, 2013.

[12] K. Nazemi, M. Klepsch, D. Burkhardt, and L. Kaupp. Comparison of Full-text Articles and Abstracts for Visual Trend Analytics through Natural Language Processing. In *2020 24th International Conference Information Visualisation (IV)*, pages 360–367. IEEE, 2020.

[13] A. Sanchez, O. Garibo, J.A. Conejero, H. Eivazi, E. Rosenberg, J. Garcia, S. Hoyas, R. Vinuesa, and F. Fuso. BLOOM — An AI-based framework for automatic classification of impact on the SDGs. *Proceedings of the 15th International Conference on Theory and Practice of Electronic Governance (ICEGOV 2022). Accepted for publication.*, 2022.

[14] P. Suri and N. Roy. Comparison between LDA & NMF for event-detection from large text stream data. In *2017 3rd International Conference on Computational Intelligence & Communication Technology (CICT)*, pages 1–5. IEEE, 2017.

[15] R. Vinuesa, H. Azizpour, I. Leite, M. Balaam, V. Dignum, S. Domisch, A. Fellander, S.D. Langhans, M. Tegmark, and F. Fuso. The role of artificial intelligence in achieving the Sustainable Development Goals. *Nature communications*, 11(1):1–10, 2020.

[16] D. Zhang, S. Mishra, E. Brynjolfsson, J. Etchemendy, D. Ganguli, B. Grosz, T. Lyons, J. Manyika J.C. Niebles, M. Sellitto, et al. The AI Index 2021 Annual Report. *arXiv preprint arXiv:2103.06312*, 2021.

Appendix A

Codes

Listing A.1: tools.py

```
# module that contains the required functions with specific functionalities such as converting texts from pdf
# to txt or preprocessing input text
import os
from string import punctuation
import subprocess
from turtle import color
import difflib
from typing import List
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.stem.porter import PorterStemmer
import gensim
from gensim.parsing.preprocessing import STOPWORDS
from gensim.models import Phrases
import pickle
import conf
import numpy as np
import matplotlib.pyplot as plt
from difflib import SequenceMatcher

def preprocess_files (folderPath):
    pdfs = [file for file in os. listdir (folderPath) if file .endswith(".pdf")]
    for pdf in pdfs:
        newPdf = standarize_file_name(pdf)
        oldPath = folderPath + pdf
        newPath = folderPath + newPdf
        os.renames(oldPath, newPath)
    # Converts the pdfs to txt
    pdfs2txt(folderPath)


def check_dictionary_valid ( filesDict ):
    # Checks if 2 files have a very close name. This generally avoids having to compare all texts
    for file in filesDict .keys ():
        closestName = difflib .get_close_matches( file , filesDict .keys (), n=2,cutoff=0.8)
        if len(closestName) > 1:
            showStr = "File with name: {} close to {}, should the process continue? (Y/N): ".
            format(file, closestName[1:])
            userInput = input(showStr)
            userInput = userInput.lower()
            if userInput == "y":
                continue
            else:
                raise Exception("Process exited by user...")

def standarize_file_name(file_name , n_iter =3):
```

```
    # removes the rare caracters from the file name
    symbols = [",", " ","-",":",";"]
    newName = file_name.lower()
    for iteration in range(0, n_iter):
        for symbol in symbols:
            newName = newName.replace(symbol, "_")

    return newName

def pdfs2txt(pdfPath:str):
    # Converts all the PDFs located in the $pdfPath$ into txt format
    # @param pdfPath path where the pdfs should be located
    os.environ["COMSPEC"] = r"C:\WINDOWS\system32\WindowsPowerShell\v1.0\powershell.exe"
    bashCommand = "bash pdftotxt.sh {} {}".format(pdfPath, pdfPath)
    subprocess.call(bashCommand, shell=True)

def tokenize_text(text:str, min_word_length:int=3, punctuation:bool=True, lemmatize:bool=True,
        stem:bool=True, stopwords:bool=True, extended_stopwords:bool=True):
    # Tokenizes the input text. First, it applies all the options.
    # @param text Input text to clear and tokenize
    # @param min_word_length Minimum length of the works to keep
    # @param punctuation Remove ASCII punctuation characters with spaces in s
    # @param lemmatize Wordnetlemmatizer
    # @param stem PorterStemmer
    # @param stopwords Remove the frequent stopwords
    # @param extended_stopwords Use the list stop_words.txt

    lemmatizer = WordNetLemmatizer()
    # stemmer = PorterStemmer()

    tokens = gensim.parsing.strip_tags(text)
    if punctuation: tokens = gensim.parsing.strip_punctuation(tokens)
    tokens = gensim.parsing.strip_numeric(tokens)
    tokens = gensim.parsing.strip_non_alphanum(tokens)
    if stem: tokens = gensim.parsing.stem_text(tokens)
    tokens = gensim.parsing.strip_multiple_whitespaces(tokens)
    tokens = gensim.utils.simple_preprocess(tokens, deacc=punctuation, min_len=min_word_length)
    for token, tokenIndex in zip(tokens, range(len(tokens))):
        newToken = token
        if lemmatize: newToken = lemmatizer.lemmatize(newToken)
        # if stem: newToken = stemmer.stem(newToken)
        tokens[tokenIndex] = newToken

    if stopwords:
        set = STOPWORDS
        if extended_stopwords:
            paths = conf.get_paths()
            with open(paths["ref"] + "stop_words.txt", 'r') as f:
                words = f.read().split('_')
                f.close()
            own_set = frozenset(words)
            set = STOPWORDS.union(own_set)
            # tokens = [token for token in tokens if not(token in set)]
        tokens = [token for token in tokens if not token in set]

    # tokens = [token for token in tokens if len(token) > min_word_length]
    return tokens

def standarize_raw_text(text:str):
    # Preprocess a raw text so that all have the same format.
    # @warning It does not tokenize or apply any process for cleaning the text
    # @param text
    outText = text
    outText = outText.lower()
    outText = outText.replace("-", " ").replace("—", " ")
    outText = gensim.parsing.strip_multiple_whitespaces(outText)
```

```
    return outText

def save_obj(obj, path:str):
    pickle.dump(obj, open(path, 'wb'))

def load_obj(path:str):
    obj = pickle.load(open(path, 'rb'))
    return obj

def segmentize_text(text:str, segment_size):
    text_segments = [text]
    textLength = len(text)
    if textLength > segment_size:
        text_segments = []; index = 0
        while(1):
            if index + segment_size > textLength:
                text_segments.append(text[index:])
                break
            else:
                if index + segment_size + 200 > textLength:
                    text_segments.append(text[index:])
                    break
                else:
                    text_segments.append(text[index:(index + segment_size)])
            index += segment_size
    return text_segments

def parse_sdgs_ascii_list (sdgs_ascii : list, append_always:bool=False):
    # Parses a list of SDGs from ascii to int
    # @param sdgs_ascii List of sdgs in ascii -> "[1,2,4]" = SDG1,2,4
    # return sdgs List of sdgs in int -> [1,2,4]
    sdgs = []
    for sdgAscii in sdgs_ascii :
        tmp = [int(sdg) for sdg in sdgAscii[1:-1]. split (',') if len(sdg) > 0]
        if len(tmp) > 0 or append_always: sdgs.append(tmp)
    return sdgs

def save_figure (fig : plt, path:str):
    if os.path.exists(path):
        os.remove(path) # otherwise, old figures are not overwritten
    fig.savefig(path)

def analyze_predict_real_sdgs(real_sdgs, predic_sdgs, path_out="", case_name="default", show=True):
    ok = np.zeros(17); wrong = np.zeros(17)
    for real, predic in zip(real_sdgs, predic_sdgs):
        for rr in real:
            if rr in predic: ok[rr - 1] += 1
            else: wrong[rr - 1] += 1

    label_ticks = range(1,18)
    plt.figure( figsize =(8, 8))
    plt.bar( label_ticks , ok + wrong, color="red")
    plt.bar( label_ticks , ok, color="green")
    plt.xlabel('SDG')
    plt.ylabel("Number_of_times_identified")
    plt.xticks( label_ticks )
    save_figure (plt, path_out + case_name + ".png")
    if show: plt.show()


def count_words(text:str):
    return len(text.split('_'))

def count_texts_per_sdg(sdgs: list [list [int ]]):
    # Gets the count per sdg
    # @param sdgs List[List[int]]
```

```
    # @return List[int(17)], Str (formatted)
    countPerSdg = np.zeros(17)
    for sdgG in sdgs:
        for sdg in sdgG:
            countPerSdg[int(sdg) - 1] += 1
    countPerSdgStr = ["SDG{}:{}".format(int(sdg), int(count)) for sdg, count in zip(range(1,18), countPerSdg)]
    countPerSdgStr = " | ".join(countPerSdgStr)

    return countPerSdg, countPerSdgStr

def count_texts_per_score( sdgs_identified , sdgs_score ):
    countPerSdg = np.zeros(17)
    for sdgG, scoreS in zip( sdgs_identified , sdgs_score ):
        for sdg, score in zip(sdgG, scoreS):
            countPerSdg[int(sdg) - 1] += score
    countPerSdgStr = ["SDG{}:{:.2f}".format(int(sdg), weight) for sdg, weight in zip(range(1,18), countPerSdg)]
    countPerSdgStr = " | ".join(countPerSdgStr)
    return countPerSdg, countPerSdgStr

def count_meanwords_per_sdg(texts:list[str], sdgs: list [ list [int ]]):
    meanWords = np.zeros(17); count = np.zeros(17)
    for text , sdgG in zip(texts, sdgs):
        nWords = count_words(text)
        for sdg in sdgG:
            meanWords[sdg - 1] += nWords
            count[sdg - 1] += 1
    for ii in range(17):
        if count[ ii ] > 0: meanWords[ii] /= count[ii] # the mean
        else: meanWords[ii] = 0
    meanWordsStr = ["SDG{}:{}".format(sdg, int(count)) for sdg, count in zip(range(1,18), meanWords)]
    meanWordsStr = " | ".join(meanWordsStr)

    return meanWords, meanWordsStr

def search_for_repeated_texts (texts : list [str ], ratio :float=0.8):
    textsOut = texts.copy(); textsIn = texts.copy(); it =0
    for text1 in textsOut:
        it += 1
        print("# Checked: {} out of {}".format(it, len(textsOut)))

        for text2, index in zip(textsIn, range(len(textsIn))):
            similarityRatio = SequenceMatcher(text1, text2).ratio()
            if similarityRatio > ratio:
                print("# Similarity {:.2f} between: \r\n Text1: {} \r\n Text2{}".
                      format(similarityRatio, text1, text2))

        textsIn.pop(0) # the first text can be removed...

def get_ok_nok_SDGsidentified(sdgs_labelled: list [ list [int ]], sdgs_identified : list [ list [int ]]):
    ok = np.zeros(17); nok = np.zeros(17)
    for ii in range(len(sdgs_labelled)):
        for sdg in sdgs_labelled [ ii ]:
            if sdg in sdgs_identified [ ii ]: ok[sdg - 1] += 1
            else: nok[sdg - 1] += 1
    return ok, nok

def plot_ok_vs_nok_SDGsidentified(sdgs_labelled : list [ list [int ]], sdgs_identified : list [ list [int ]],
        path_out:str="", show:bool=False):
    ok, nok = get_ok_nok_SDGsidentified(sdgs_labelled, sdgs_identified )

    xlabel = [ii for ii in range(1, 18)]

    plt.figure( figsize =(8, 8))
    for xx in xlabel:
        plt.bar(xx, ok[xx - 1] + nok[xx - 1], width=0.3, alpha=0.5, color='green',)
        plt.bar(xx, ok[xx - 1]                , width=0.3, alpha=1.0, color='green')
```

```
    plt.xticks(xlabel)
    plt.xlabel('SDG')
    plt.ylabel("Number_of_texts")
    # plt.ylim(top=0.5)
    # plt. title ('SDGs to identify: {}'.format(labeledSDGs[textIndex]))
    plt.legend(['Not_identified', 'Identified'])
    if len(path_out) > 4: plt.savefig(path_out)
    if show: plt.show()


def plot_SDGsidentified( sdgs_identified : list [ list [int ]], sdgs_scores : list [ list [float ]],
        path_out:str="", with_score:bool=False, show:bool=False, fontsize:int=14):
    xlabel = [ii for ii in range(1, 18)]
    if with_score:
        counts, countstr = count_texts_per_score( sdgs_identified , sdgs_scores )
    else:
        counts, countstr = count_texts_per_sdg( sdgs_identified )

    plt.figure( figsize =(8, 8))
    plt.bar(xlabel, counts, width=0.3, alpha=1.0, color='green')
    plt.xticks(xlabel)
    plt.xlabel('SDG', fontsize=fontsize)
    if with_score:
        plt.ylabel("Total_weight_(sum_of_individual_score)", fontsize =fontsize)
    else:
        plt.ylabel("Number_of_times_identified", fontsize=fontsize)
    # plt.ylim(top=0.5)
    # plt. title ('SDGs to identify: {}'.format(labeledSDGs[textIndex]))
    plt.tick_params(axis='x', labelsize =fontsize)
    plt.tick_params(axis='y', labelsize =fontsize)
    if len(path_out) > 4:
        if os.path.exists(path_out):
            os.remove(path_out) # otherwise, old figures are not overwritten
        plt.savefig(path_out)
    if show: plt.show()
```

Listing A.2: data.py

```
# File that contains the functions for accesing the required data: training and validation

from cmath import isnan
# from curses import raw
import os
from typing import Dict
from numpy import empty
import pandas as pd
import json
import conf
import tools


def get_sdg_titles (refPath):
    # returns the title of each SDG as a dictionary, with key: SDGx, value = title.
    # for example: "SDG1":"No poverty"
    f = open(refPath + "SDG_titles.json")
    sdgs_title = json.load(f)
    f.close()
    return sdgs_title

def get_sdgs_seed_list (refPath):
    with open(refPath + "seed_list_sdgs.json", 'r') as f:
        text = f.read()
        f.close()
        dict = json.loads(text)
    return list(dict.values())
```

```
# DATASET: the role of artificial intelligence in achieving the sustainable development goals. NATURE PAPER.
# The user can select independently: abstract, keywords, introduction, body or conclusions.
def get_nature_files (abstract=True, kw=False, intro=False, body=False, concl=False):
    paths = conf.get_paths()
    with open(paths["ref"] + "cleaned_database.json", "r") as f:
        json_dump = f.read()
        f.close()
    database = json.loads(json_dump)

    corpus = []; associatedSDGs = []; indexes = []
    for file, index in zip(database, range(len(database))):
        text = ""
        sdgs = database[file]["SDG"]
        if 17 in sdgs:
            continue
        if abstract:
            if len(database[file]["abstract"]) > 50:
                text += "␣" + database[file]["abstract"]
        if kw:
            text += "␣" + database[file]["keywords"]
        if intro:
            text += "␣" + database[file]["introduction"]
        if body:
            text += "␣" + database[file]["body"]
        if concl:
            text += "␣" + database[file]["conclusions"]
        corpus.append(text)
        associatedSDGs.append(sdgs)
        indexes.append(index)
    print("#␣{}␣nature_files␣were_found".format(len(corpus)))
    return [corpus, associatedSDGs, indexes]

def get_nature_abstracts():
    paths = conf.get_paths()
    with open(paths["ref"] + "cleaned_database.json", "r") as f:
        json_dump = f.read()
        f.close()
    database = json.loads(json_dump)

    corpus = []; associatedSDGs = []; indexes = []
    for (file, index) in zip(database, range(len(database))):
        sdgs = database[file]["SDG"]
        if 17 in sdgs:
            continue
        if len(database[file]["abstract"].split('␣')) > 50:
            corpus.append(database[file]["abstract"])
            associatedSDGs.append(sdgs)
            indexes.append(index)
    print("#␣{}␣nature_abstracts␣were_found".format(len(corpus)))
    return [corpus, associatedSDGs, indexes]

def get_nature_abstracts_filtered ():
    paths = conf.get_paths()
    excel = pd.read_excel(paths["ref"] + "test2set_thresholds.xlsx")
    texts = list(excel["text"]); sdgsAscii = list(excel["sdgs"])
    sdgs = tools.parse_sdgs_ascii_list (sdgsAscii)
    return [texts, sdgs]

# DATASET: https://sdgs.un.org/
# - Goals definition
# - Goals progress - evolution section
def get_sdgs_org_files (refPath, sdg_query=-1):
    # Returns an array where each elements consist of an array with the fields :
    # [0] abstract or text related to a SDG, [1]: array with the associated SDGs.
    path = refPath + "sdg_texts.xlsx"
```

```
    df = pd.read_excel(path)
    texts = list(df["text"]); sdgs = tools. parse_sdgs_ascii_list (list(df["sdg"]))

    corpus = []; associatedSDGs = []
    for text, sdg in zip(texts, sdgs):
        if sdg[0] == 17: continue # not included
        if sdg_query > 0:
            if sdg_query == sdg[0]:
                corpus.append(text)
                associatedSDGs.append(sdg)
        else:
            corpus.append(text)
            associatedSDGs.append(sdg)

    nFiles = len(corpus)
    print("# {} sdgs files were found".format(nFiles))
    return [corpus, associatedSDGs]

# DATASET: https://www.kaggle.com/datasets/xhlulu/medal-emnlp
def get_health_care_files (refPath):
    sdgsPath = [refPath + "Extra_files/SDG3/"]
    corpus = []; associatedSDGs = []
    for path in sdgsPath:
        for file in os. listdir (path):
            f = open(path + file, 'r')
            text = f.read()
            f. close ()
            fileSDG = [3]
            corpus.append(text)
            associatedSDGs.append(fileSDG)
    nFiles = len(corpus)
    print("- {} health_care_files (SDG3) were_found".format(nFiles))
    return [corpus, associatedSDGs]

# DATASET: files from scopus classified as related to a sdg previously by the algorithm
def get_previous_classified_abstracts (refPath):
    # returns files that were previously classifies by the algorithm as valid
    absPath = refPath + "Abstracts/"
    abstracts = []
    for file in os. listdir (absPath):
        if file .endswith(".txt"):
            f = open(absPath + file, 'r', encoding="utf-8")
            text = f.read()
            f. close ()
            abstracts.append(text)

# DATASET: https://sdg-pathfinder.org/ files related to each SDG
def get_sdgs_pathfinder(refPath, min_words=150):
    csv = pd.read_csv(refPath + "ds_sdg_path_finder.csv")
    corpus = []; sdgs = []
    for text, sdgsAscii in zip(csv["text"], csv["SDGs"]):
        sdgsInt = [int(sdg) for sdg in (sdgsAscii.replace("[","").replace("]","")). split (",")]
        if 17 in sdgsInt:
            continue
        if len(text. split (' ')) > min_words:
            corpus.append(text)
            sdgs.append(sdgsInt)
    print("- {} texts_in_the_pathfinder_dataset".format(len(corpus)))
    return [corpus, sdgs]

# MANUAL SELECTED files
def get_extra_manual_files (refPath, sdg_query=[], verbose=True):
    # Returns an array where each elements consist of an array with the fields :
    # [0] abstract or text related to a SDG, [1]: array with the associated SDGs.
    sdgsPaths = [refPath + "Manual_selected/"]
    corpus = []; associatedSDGs = []
```

```
    for path in sdgsPaths:
        for file in os.listdir(path):
            filePath = path + file
            if not os.path.isfile(filePath): continue
            f = open(filePath, 'r', encoding='utf8')
            text = f.read()
            f.close()
            fileSDG = []
            for sdg in file.split("_"):
                if sdg.isdigit():
                    if int(sdg) == 17: continue
                    fileSDG.append(int(sdg))
            ok = 0
            if len(sdg_query) > 0:
                for sdg in fileSDG:
                    if sdg in sdg_query: ok += 1
            else: ok = 1

            if ok > 0:
                corpus.append(text)
                associatedSDGs.append(fileSDG)

    nFiles = len(corpus)
    if verbose: print("#_{}_manual_files_were_found".format(nFiles))
    return [corpus, associatedSDGs]

def get_iGEM_files(ref_path, verbose=True):
    path = ref_path + "iGEM_2004_2021/"
    fieldsSeparator = ":::"
    with open(path + "00Header.txt", 'r') as hd:
        text = hd.read()[:-1]; hd.close()
        fields = text.split(fieldsSeparator)

    abstracts = []; extInformation = []; not_valid = []
    for folder in os.listdir(path=path):
        if folder.startswith("iGEM"):
            for file in os.listdir(path=(path + folder)):
                if file.startswith("0"): continue # it is not a valid file
                try:
                    fp = open(path + folder + "/" + file, 'r', encoding='utf8')
                    text = fp.read()[:-1]; fp.close()
                    fieldsValue = text.split(fieldsSeparator)
                    data = dict()
                    for fieldValue, fieldName in zip(fieldsValue, fields):
                        data[fieldName] = fieldValue

                    # append the data to the lists if OK
                    if data['Application'] == 'Accepted' and len(data['Abstract'].split('_')) > 10:
                        data["Abstract"] = data["Abstract"].encode("ascii", "ignore")
                        abstracts.append(data["Abstract"])
                        extInformation.append(data)
                except:
                    not_valid.append(path + folder + "/" + file)

    if verbose:
        print('##_{}_accepted_texts_with_abstract_were_found'.format(len(abstracts)))
        for file in not_valid:
            print('#_Revise:_' + file)

    return [abstracts, extInformation]

def get_aero_files(ref_path:str, verbose:bool=True) -> pd.DataFrame:
    path = ref_path + "Aero/"

    authors = []; titles = []; years = []; citations = []; links = []
    countries = []; abstracts = []; keywords = []
```

```
    for csv_name in os.listdir(path):
        csv = pd.read_csv(path + csv_name)
        authors += list(csv["Authors"])
        titles += list(csv["Title"])
        years += list(csv["Year"])
        citations += list(csv["Cited_by"])
        links += list(csv["Link"])

        def extract_country(affiliation :str) -> str:
            if not isinstance(affiliation, str): return "" # to protect
            ind = affiliation.find(";")
            if not ind == -1: affiliation = affiliation[0:ind]
            country = affiliation.split(',')[-1]

            if verbose: print(' Affiliation : {} -> Country: {}'.format(affiliation, country))
            return country

        for affiliation in list(csv["Affiliations"]):
            country = extract_country(affiliation)
            countries.append(country)

        abstracts += list(csv["Abstract"])
        keywords += list(csv["Index_Keywords"])

    df = pd.DataFrame()
    df["authors"] = authors; df["titles"] = titles; df["years"] = years; df["citations"] = citations
    df["links"] = links; df["countries"] = countries; df["abstracts"] = abstracts; df["keywords"] = keywords
    return df

def update_datasets():
    # Updates the datasets that can be used for training, validation or analysis
    print("# Updating datasets...")

    paths = conf.get_paths()
    raw_orgFiles, sdgs_orgFiles = get_sdgs_org_files(paths["SDGs_inf"])
    raw_extraFiles, sdgs_extra = get_extra_manual_files(paths["ref"],
                                                        sdg_query=[] # queries all the sdgs, not filter
                                                        )
    raw_natureShort, sdgs_nature, index_abstracts = get_nature_abstracts()
    raw_natureExt, sdgs_natureAll, index_full = get_nature_files(abstract=True, kw=True, intro=True,
        body=True, concl=True)

    df_aero = get_aero_files(ref_path=paths["test"], verbose=False)
    raw_aero = list(df_aero["abstracts"])

    # 1. Clears all the texts, standarizing them
    print("# 1. Clearing_texts...")
    corpus = raw_orgFiles + raw_extraFiles + raw_natureShort + raw_natureExt
    nFiles = len(corpus); nFilesAero = len(raw_aero)
    corpus += raw_aero
    sdgs = sdgs_orgFiles + sdgs_extra + sdgs_nature + sdgs_natureAll + [[] for ii in range(nFilesAero)]
    identifiers = ["org" for ii in range(len(raw_orgFiles))] \
                + ["manual_extra" for ii in range(len(raw_extraFiles))] \
                + ["nature_abstract" for ii in range(len(raw_natureShort))] \
                + ["nature_all" for ii in range(len(raw_natureExt))] \
                + ["aero" for ii in range(nFilesAero)]
    years = [-1 for ii in range(nFiles)] + list(df_aero["years"])
    citations = [-1 for ii in range(nFiles)] + list(df_aero["citations"])
    countries = ["" for ii in range(nFiles)] + list(df_aero["countries"])
    keywords = ["" for ii in range(nFiles)] + list(df_aero["keywords"])
    print("# Total_number_of_texts_in_datasets:", len(identifiers))

    # 2. Generates stem and lem datasets
    print("# 2. Creating_datasets...")
    stand_texts = []; lem_texts = []; lem_stem_texts = []
    for text in corpus:
```

```
            standarized = tools.standarize_raw_text(text) # all the texts should be based on the standarized version

            stand_texts.append(standarized)
            lem_texts.append(" ".join(tools.tokenize_text(standarized, min_word_length=3, punctuation=True,
                lemmatize=True, stem=False, stopwords=True, extended_stopwords=True)))
            lem_stem_texts.append(" ".join(tools.tokenize_text(standarized, min_word_length=3, punctuation=True,
                lemmatize=True, stem=True, stopwords=True, extended_stopwords=True)))

    # 3. Stores the datasets
    outPath = "datasets/"
    print("# 3. Storing datasets in " + outPath + "...")
    df = pd.DataFrame()
    df["standard"] = stand_texts
    df["lem"] = lem_texts
    df["lem_stem"] = lem_stem_texts
    df["sdgs"] = sdgs
    df[" identifier "] = identifiers
    df["years"] = years
    df[" citations "] = citations
    df["countries"] = countries
    df["keywords"] = keywords

    df.to_csv(outPath + "dataset.csv")

    dc = dict()
    dc["standard"] = stand_texts
    dc["lem"] = lem_texts
    dc["lem_stem"] = lem_stem_texts
    dc["sdgs"] = sdgs
    dc[" identifier "] = identifiers
    dc["years"] = years
    dc[" citations "] = citations
    dc["countries"] = countries
    dc["keywords"] = keywords

    with open(outPath + 'dataset.json', 'w') as outfile:
        json.dump(dc, outfile)

def get_dataset(requires_update:bool=False, filter:list =[]):
    # Gets the dataset where all the texts are stored
    # @return Dictionary with all the data:
    # "standard" -> raw texts
    # "lem" -> lemmatized texts
    # "lem_stem" -> lemmatized + stemmed texts
    # "sdgs" -> associates sdgs to each text
    # "identifier" -> source of each text. "org", "manual_extra", "nature_abstract", "nature_all"
    if requires_update: update_datasets()

    datasetPath = "datasets/dataset.json"
    print("# Opening " + datasetPath)
    with open(datasetPath, "r") as f:
        json_dump = f.read()
        f.close()

    dataset = json.loads(json_dump)

    if len(filter) > 0:
        newDataset = dict(); keys = dataset.keys()
        for key in keys: newDataset[key] = [] # empty initializations

        for id, index in zip(dataset[" identifier "], range(len(dataset[" identifier "]))):
            if id in filter:
                for key in keys: newDataset[key].append(dataset[key][index])
        dataset = newDataset

    return dataset
```

```
# update_datasets()
# db = get_dataset()
```

Listing A.3: model_nmf.py

```
from cmath import isnan
from signal import valid_signals
import tools
import pandas as pd
import numpy as np
from sklearn.decomposition import NMF
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import TfidfVectorizer
import warnings
warnings.filterwarnings('ignore')

# Class associated to the Non-Negative Matrix classifier.
class NMF_classifier:
    paths=[]
    model=[]
    vectorizer=[]
    topics_association=[]
    train_data=[]
    verbose=False

    def __init__(self, paths, verbose=False):
        self.paths = paths
        self.verbose = verbose

    def train(self, train_data, n_topics, ngram, min_df=1, max_iter=2000, l1=0.0, alpha_w=0.0, alpha_h=0.0):
        self.train_data = train_data
        # the corpus should be preprocessed before
        self.vectorizer = TfidfVectorizer(min_df=min_df, encoding='utf-8', ngram_range=ngram)
        vectorized_data = self.vectorizer.fit_transform(train_data[0])
        self.model = NMF(n_components=n_topics, random_state=5, verbose=False, max_iter=max_iter,
                         alpha_W=alpha_w, alpha_H=alpha_h, l1_ratio=l1)
        self.model.fit(vectorized_data)
        a=2



    def print_stopwords(self, path_csv=""):
        stopwords = list(self.vectorizer.stop_words_)
        df = pd.DataFrame()
        df["stopwords"] = stopwords
        if len(path_csv) > 4:
            df.to_csv(path_csv)

    def test_model(self, corpus, associated_SDGs, score_threshold=0.2, segmentize=-1, filter_low=False,
        path_to_plot="", path_to_excel="", normalize=True, expand_factor=1.0):
        rawSDG = []; rawSDGseg = []; pred_sdgs = []
        predictedSDGs = []; realSDGs = []
        valids = []; validsAny = []
        texts = []
        statsGlobal = []
        countPerSDG = np.zeros(17)
        countWellPredictionsPerSDG = np.zeros(17)
        maxSDG = 0.0
        for text, labeled_sdgs in zip(corpus, associated_SDGs):
            if segmentize > 0:
                # then the documents are divided
                text_segments = tools.segmentize_text(text, segment_size=segmentize)
                raw_sdgs = np.zeros(17)
                for segment in text_segments:
                    raw_sdgs += self.map_text_to_sdgs(segment, filter_low=filter_low,
```

```
                    normalize=normalize, expand_factor=expand_factor)
                raw_sdgs /= len(text_segments)
                raw_sdgs_seg = raw_sdgs
            else:
                raw_sdgs_seg = np.zeros(17)
                raw_sdgs = self.map_text_to_sdgs(text, filter_low = filter_low , normalize=normalize,
                    expand_factor=expand_factor)

            predic_sdgs = [list(raw_sdgs).index(sdgScore) + 1 for sdgScore in raw_sdgs if sdgScore > score_threshold]
            validSingle = False; ii = 0
            for sdg in labeled_sdgs:
                countPerSDG[sdg - 1] += 1
                if sdg in predic_sdgs:
                    validSingle = True
                    ii += 1
                    countWellPredictionsPerSDG[sdg - 1] += 1
            valid = False
            if ii == len(labeled_sdgs):
                valid = True
            maxLocal = max(raw_sdgs)
            if maxLocal > maxSDG: maxSDG = maxLocal

            raw_sdgsAscii = ["x{}: {:.2f}".format(xx, topic) for topic, xx in zip(raw_sdgs, range(1,18))]
            raw_sdgsAscii = "|".join(raw_sdgsAscii)

            raw_sdgsAsciiseg = ["x{}: {:.2f}".format(xx, topic) for topic, xx in zip(raw_sdgs_seg, range(1,18))]
            raw_sdgsAsciiseg = "|".join(raw_sdgsAsciiseg)

            stats = [min(raw_sdgs), np.mean(raw_sdgs), max(raw_sdgs)]
            statsAscii = "[{:.2f}, {:.2f}, {:.2f}]".format(stats[0], stats[1], stats[2])

            rawSDG.append(raw_sdgsAscii)
            rawSDGseg.append(raw_sdgsAsciiseg)
            statsGlobal.append(statsAscii)
            predictedSDGs.append(predic_sdgs)
            realSDGs.append(labeled_sdgs)
            texts.append(text)
            valids.append(valid)
            validsAny.append(validSingle)
            pred_sdgs.append(predic_sdgs)

        oks = [ok for ok in valids if ok == True]
        oksSingle = [ok for ok in validsAny if ok == True]
        perc_valid_global = len(oks) / len(valids) * 100; perc_valid_any = len(oksSingle) / len(valids) * 100
        print("- {:.2f} % valid_global, {:.2f} % valid_any, of {} files".format(perc_valid_global,
            perc_valid_any, len(valids)))
        print('Max found: {:.3f}'.format(maxSDG))

        if len(path_to_excel) > 0:
            df = pd.DataFrame()
            df["text"] = texts
            df["labeled_sdgs"] = realSDGs
            df["sdgs_association"] = rawSDG
            df["sdgs_segmentated"] = rawSDGseg
            df["stats"] = statsGlobal
            df["predict_sdgs"] = predictedSDGs
            df[" all_valid "] = valids
            df["any_valid"] = validsAny
            df.to_excel(path_to_excel)

        return [rawSDG, perc_valid_global, perc_valid_any, maxSDG, pred_sdgs]

    def map_model_topics_to_sdgs(self, n_top_words, normalize=True, path_csv=""):
        # Maps each new topic of the general NMF model to an specific SDG obtained from training 17 models
        nTopics = self.model.n_components
```

```
        self.topics_association = np.zeros((nTopics, 17))
        for text, labeled_sdgs in zip(self.train_data[0], self.train_data[1]):
            topicScores = self.infer_text(text)
            for topicIndex, score in zip(range(nTopics), topicScores):
                for sdg in labeled_sdgs:
                    tmp = np.zeros(17)
                    tmp[sdg - 1] = 1
                    self.topics_association[topicIndex] += score * tmp
        sum_per_topic = np.zeros(17)
        for ii in range(nTopics):
            if normalize:
                norm_topics = self.topics_association[ii] / sum(self.topics_association[ii])
                topics_to_delete = norm_topics < 0.1
                for nn, delete, index in zip(norm_topics, topics_to_delete, range(len(norm_topics))):
                    if delete:
                        norm_topics[index] = 0.0
                ss = sum(norm_topics)
                if ss > 0: norm_topics = norm_topics / ss
                self.topics_association[ii] = norm_topics

                sum_per_topic += self.topics_association[ii]
            if self.verbose:
                listAscii = ["x{}:{:.3f}".format(xx, sdg) for xx, sdg in zip(range(1,18),
                    self.topics_association[ii])]
                print('Topic{:2d}: '.format(ii), '|'.join(listAscii))
        listAscii = ["x{}:{:.2f}".format(xx, sdg) for xx, sdg in zip(range(1,18), sum_per_topic)]
        if self.verbose:
            print('GLOBAL: ' + '|'.join(listAscii))

        if len(path_csv) > 4:
            # Then the mapping result is stored in a csv
            dfMap = pd.DataFrame()
            for ii in range(nTopics):
                listAscii = ["{:.2f}".format(sdg) for sdg in self.topics_association[ii]]
                dfMap["topic{}".format(ii)] = listAscii
            dfMap = dfMap.transpose()
            dfMap.columns = ["SDG{}".format(ii) for ii in range(1,18)]
            dfMap.to_csv(self.paths["out"] + "NMF/" + "association_matrix.csv")

            df = pd.DataFrame()
            topic_words_ascii = []
            for ii in range(nTopics):
                terms, scores = self.get_topic_terms(ii, topn=n_top_words)
                scoreTerm = ["{:.3f}:{}".format(sc, tm) for sc, tm in zip(scores, terms)]
                topic_words_ascii.append(scoreTerm)

            # all the top SDGs with an score > 0.1 are considered for that topic
            for topicIndex in range(nTopics):
                topSDGs = sorted(self.topics_association[topicIndex], reverse=True)
                title = []
                for sdg in topSDGs:
                    if sdg < 0.1: break
                    sdgIndex = list(self.topics_association[topicIndex]).index(sdg)
                    ass_sdg = self.topics_association[topicIndex][sdgIndex]
                    title.append("{:.2f}*SDG{}".format(ass_sdg, sdgIndex + 1))
                title = ",".join(title)
                df[title] = topic_words_ascii[topicIndex]

            df.to_csv(path_csv)

    def map_text_to_sdgs(self, text, filter_low =True, normalize=True, expand_factor=1.0):
        query_words_vect = self.vectorizer.transform([text])
        topicFeats = self.model.transform(query_words_vect)[0]

        sdgs_score = np.zeros(17)
        for topicScore, topicIndex in zip(topicFeats, range(len(topicFeats))):
```

```
            sdgs_score += topicScore * self.topics_association[topicIndex]
        sdgs_score *= expand_factor

        if filter_low:
            raw_sdgs_filt = sdgs_score < 0.05
            for prob, index, filt in zip(sdgs_score, range(len(sdgs_score)), raw_sdgs_filt):
                if filt:
                    prob = sdgs_score[index]
                    sdgs_score[index] = 0.0
                    sdgs_score += prob * sdgs_score / sum(sdgs_score)

        if normalize:
            sdgs_score = sdgs_score / sum(sdgs_score)

        return sdgs_score

    def map_text_to_topwords(self, text, top_n):
        query_words_vect = self.vectorizer.transform([text])
        topicFeats = self.model.transform(query_words_vect)[0]

        words_collection = []
        for topicScore, topicIndex in zip(topicFeats, range(len(topicFeats))):
            words, scores = self.get_topic_terms(topicIndex, top_n)
            for word, score in zip(words, scores):
                words_collection.append((word, score * topicScore))

        def sort_method(elem):
            return elem[1]

        words_collection.sort(key=sort_method, reverse=True)
        return words_collection[:top_n]

    def infer_text(self, text):
        query_words_vect = self.vectorizer.transform([text])
        topicScores = self.model.transform(query_words_vect)[0]
        return topicScores

    def get_topic_terms(self, topic, topn):
        # Returns the n_top_words for each of the n_topics for the topic that is queried
        feat_names = self.vectorizer.get_feature_names_out()
        words_ids = self.model.components_[topic].argsort()[:-topn - 1:-1]
        words = [feat_names[key] for key in words_ids]
        scores = [self.model.components_[topic][key] for key in words_ids]
        return words, scores
```

Listing A.4: model_lda.py

```
from signal import valid_signals
import tools
import pandas as pd
import numpy as np
from gensim.models import LdaModel
from sklearn.preprocessing import normalize
import tools
import warnings
warnings.filterwarnings('ignore')

# Class associated to the Latent-Dirichlet Allocation model
class LDA_classifier(LdaModel):
    paths=[]
    topics_association=[]
    dict=[]
    verbose=False
    train_data=[]
```

```
def set_conf(self, paths, dict, verbose=False):
    self.paths = paths
    self.dict = dict
    self.verbose = verbose

def test_model(self, corpus, sdgs, path_to_plot="", path_to_excel="", only_bad=False, score_threshold=3.0,
    only_positive=False,    segmentize=-1, filter_low=False, expand_factor=1.0, normalize=True):
    rawSDG = []; rawSDGseg = []
    predictedSDGs = []
    realSDGs = []
    valids = []
    validsAny = []
    texts = []
    statsGlobal = []
    pred = []
    countPerSDG = np.zeros(17)
    countWellPredictionsPerSDG = np.zeros(17)
    maxSDG = 0.0
    for text, labeled_sdgs in zip(corpus, sdgs):
        if segmentize > 0:
            # then the documents are divided
            text_segments = tools.segmentize_text(text, segment_size=segmentize)
            raw_sdgs = np.zeros(17)
            for segment in text_segments:
                raw_sdgs += self.map_text_to_sdgs(segment, only_positive=only_positive,
                    filter_low=filter_low, normalize=normalize, expand_factor=expand_factor)
            raw_sdgs /= len(text_segments)
            raw_sdgs_seg = raw_sdgs
        else:
            raw_sdgs_seg = np.zeros(17)
            raw_sdgs = self.map_text_to_sdgs(text, only_positive=only_positive, filter_low=filter_low,
             normalize=normalize, expand_factor=expand_factor)

        # raw_sdgs *= expand_factor

        maxLocal = max(raw_sdgs)
        if maxLocal > maxSDG: maxSDG = maxLocal

        predic_sdgs = [list(raw_sdgs).index(sdgScore) + 1 for sdgScore in raw_sdgs if sdgScore > score_threshold]
        validSingle = False; ii = 0
        for sdg in labeled_sdgs:
            countPerSDG[sdg - 1] += 1
            if sdg in predic_sdgs:
                validSingle = True
                ii += 1
                countWellPredictionsPerSDG[sdg - 1] += 1
        valid = False
        if ii == len(labeled_sdgs):
            valid = True

        if (only_bad and not(valid)) or not(only_bad):
            raw_sdgsAscii = ["x{}: {:.2f}".format(xx, topic) for topic, xx in zip(raw_sdgs, range(1,18))]
            raw_sdgsAscii = "|".join(raw_sdgsAscii)

            raw_sdgsAsciiseg = ["x{}: {:.2f}".format(xx, topic) for topic, xx in zip(raw_sdgs_seg, range(1,18))]
            raw_sdgsAsciiseg = "|".join(raw_sdgsAsciiseg)

            stats = [min(raw_sdgs), np.mean(raw_sdgs), max(raw_sdgs)]
            statsAscii = "[{:.2f}, {:.2f}, {:.2f}]".format(stats[0], stats[1], stats[2])

            rawSDG.append(raw_sdgsAscii)
            rawSDGseg.append(raw_sdgsAsciiseg)
            statsGlobal.append(statsAscii)
            predictedSDGs.append(predic_sdgs)
            realSDGs.append(labeled_sdgs)
            texts.append(text)
```

```
            pred.append(predic_sdgs)
        valids.append(valid)
        validsAny.append(validSingle)

    oks = [ok for ok in valids if ok == True]
    oksSingle = [ok for ok in validsAny if ok == True]
    perc_valid_global = len(oks) / len(valids) * 100; perc_valid_any = len(oksSingle) / len(valids) * 100
    print("- {:.2f} % valid_global, {:.2f} % valid_any, of {} files".format(perc_valid_global,
        perc_valid_any, len(valids)))
    print('Max_found: {:.3f}'.format(maxSDG))

    if len(path_to_excel) > 0:
        df = pd.DataFrame()
        df["text"] = texts
        df["labeled_sdgs"] = realSDGs
        df["sdgs_association"] = rawSDG
        df["sdgs_segmentated"] = rawSDGseg
        df["stats"] = statsGlobal
        df["predict_sdgs"] = predictedSDGs
        df[" all_valid "] = valids
        df["any_valid"] = validsAny
        df.to_excel(path_to_excel)


    return [rawSDG, perc_valid_global, perc_valid_any, maxSDG, pred]

def print_summary(self, top_words, path_csv=""):
    nTopics = len(self.get_topics())
    topicsWords = [[] for ii in range(nTopics)]
    dfTopics = pd.DataFrame()
    words_prob = self.show_topics(num_topics=nTopics, num_words=top_words, log=False, formatted=False)
    for topicIndex in range(nTopics):
        distribution = words_prob[topicIndex]
        for elem in distribution[1]:
            topicsWords[topicIndex].append("{:.3f}:{}".format(elem[1], elem[0]))
        topicName = "Topic{}".format(topicIndex)
        dfTopics[topicName] = topicsWords[topicIndex]

    topic = 0
    maxTopicsPerLine = 7
    while(1):
        if topic + maxTopicsPerLine > nTopics:
            print(dfTopics.iloc[:, topic:])
            break
        else:
            print(dfTopics.iloc[:, topic:(topic + maxTopicsPerLine)])
        topic += maxTopicsPerLine

    if len(path_csv) > 0:
        try:
            dfTopics.to_csv(path_csv)
        except:
            print('CSV_IS_OPENED..._ABORTING_TOPICS_EXPORT')

def map_model_topics_to_sdgs(self, train_data, top_n_words=30, path_csv="", normalize=False, verbose=False):
    # maps each internal topic with the SDGs. A complete text associated to each specific SDG is fetched.
    # Then each topic is compared with each text and the text-associated sdg with the maximum score is selected
    # as the SDG.
    self.train_data = train_data
    nTopics = len(self.get_topics())
    self.topics_association = np.zeros((nTopics, 17))
    for text, labeled_sdgs in zip(train_data[0], train_data[1]):
        topics, probs = self.infer_text(text)
        for (topicIndex, score) in zip(topics, probs):
            # if subScore < meanSub: continue
            tmp = np.zeros(17)
```

```
            for sdg in labeled_sdgs:
                tmp[sdg - 1] = 1
            self.topics_association[topicIndex] += score * tmp
    sum_per_topic = np.zeros(17)
    for ii in range(nTopics):
        if normalize:
            norm_topics = self.topics_association[ii] / sum(self.topics_association[ii])
            for nn in norm_topics:
                if nn < 0.1:
                    norm_topics[list(norm_topics).index(nn)] = 0.0
            norm_topics = norm_topics / sum(norm_topics)
            self.topics_association[ii] = norm_topics

            sum_per_topic += self.topics_association[ii]
        if verbose:
            listAscii = ["x{}:{:.3f}".format(xx, sdg) for xx, sdg in zip(range(1,18),
                self.topics_association[ii])]
            print('Topic{:2d}: '.format(ii), '|'.join(listAscii))
    listAscii = ["x{}:{:.2f}".format(xx, sdg) for xx, sdg in zip(range(1,18), sum_per_topic)]
    if verbose:
        print('GLOBAL: ' + '|'.join(listAscii))

    listSDGsOut = '|'.join(listAscii)
    if len(path_csv) > 4:
        dfMap = pd.DataFrame()
        rows = []
        sum_per_sdg = np.zeros(17)
        for ii in range(nTopics):
            listAscii = ["x{}:{:.3f}".format(xx, sdg) for xx, sdg in zip(range(1,18), self.topics_association[ii])]
            sum_per_sdg += self.topics_association[ii]
            rows.append('|'.join(listAscii))
        sum_ascii = ["x{}:{:.3f}".format(xx, sdg) for xx, sdg in zip(range(1,18), sum_per_sdg)]
        rows.append('|'.join(sum_ascii))
        dfMap["topics_association_map"] = rows
        dfMap.to_excel(self.paths["out"] + "LDA/" + "topics_map.xlsx")

        np.savetxt(self.paths["out"] + "LDA/" + "topics_map.csv", self.topics_association, delimiter=",")

        # Then the mapping result is stored in a csv
        df = pd.DataFrame()
        topics_words = []
        for ii in range(nTopics):
            topics_words.append(self.get_topic_terms(ii, topn=top_n_words))
        topic_words_ascii = [[] for ii in range(nTopics)]
        for words in topics_words:
            topicIndex = topics_words.index(words)
            for word in words:
                topic_words_ascii[topicIndex].append("{:.3f}:{}".format(word[1], self.dict.id2token[word[0]]))

        # all the top SDGs with an score > 0.1 are considered for that topic
        for topicIndex in range(nTopics):
            topSDGs = sorted(self.topics_association[topicIndex], reverse=True)
            title = []
            for sdg in topSDGs:
                if sdg < 0.1: break
                sdgIndex = list(self.topics_association[topicIndex]).index(sdg)
                ass_sdg = self.topics_association[topicIndex][sdgIndex]
                title.append("{:.2f}*SDG{}".format(ass_sdg, sdgIndex + 1))
            title = ",".join(title)
            df[title] = topic_words_ascii[topicIndex]

        df.to_csv(path_csv)

    return [sum_per_topic, listSDGsOut]

def map_text_to_sdgs(self, text, min_threshold=0, only_positive=True, filter_low=True, normalize=True,
```

```
        expand_factor=1.0):
        text = self.convert_text(text)

        topics, probs = self.infer_text(text)
        sdgs = np.zeros(17)
        for topic, prob in zip(topics, probs):
            if (prob < min_threshold) or (prob < 0 and only_positive): continue
            sdgs += prob * self.topics_association[topic]
        sdgs *= expand_factor
        if filter_low:
            raw_sdgs_filt = sdgs < 0.05
            for prob, index, filt in zip(sdgs, range(len(sdgs)), raw_sdgs_filt):
                if filt:
                    prob = sdgs[index]
                    sdgs[index] = 0.0
                    sdgs += prob * sdgs / sum(sdgs)

        if normalize:
            sdgs = sdgs / sum(sdgs)

        return sdgs

    def map_text_to_topwords(self, text, top_n):
        if isinstance(text, str): text = text.split('␣')
        elif isinstance(text, list): text = text
        else: raise ValueError('Text_type_is_not_valid')

        words_collection = []
        topics, probs = self.infer_text(text)
        for topic, prob in zip(topics, probs):
            words = self.get_topic_terms(topic)
            for pair in words:
                wrd = self.dict.id2token[pair[0]]
                score = pair[1]
                words_collection.append((wrd, score * prob))

        def sort_method(elem):
            return elem[1]

        words_collection.sort(key=sort_method, reverse=True)
        return words_collection[:top_n]

    def infer_text(self, text):
        bow = self.dict.doc2bow(text)
        result = self.get_document_topics(bow, minimum_probability=None, minimum_phi_value=None)
        topics = [elem[0] for elem in result]
        probs = [elem[1] for elem in result]
        return [topics, probs]

    def convert_text(text):
        if isinstance(text, str): text = text.split('␣')
        elif isinstance(text, list): text = text
        else: raise ValueError('Text_type_is_not_valid')
        return text
```

Listing A.5: model_top2vec.py

```
from signal import valid_signals
import pandas as pd
import numpy as np
from sklearn.preprocessing import normalize
import tools
import warnings
from top2vec import Top2Vec
warnings.filterwarnings('ignore')
```

```
# Class associated to the Top2Vec model
class Top2Vec_classifier:
    paths=[]
    model=[]
    train_data=[]
    topics_association=[]
    verbose=False

    def __init__(self, paths, verbose=False):
        self.paths = paths
        self.verbose = verbose

    def set_conf(self, paths, dict, verbose=False):
        self.paths = paths
        self.dict = dict
        self.verbose = verbose

    def train(self, train_data, embedding_model="doc2vec", ngram=True, method="learn", workers=8,
        min_count=2, tokenizer=False):
        # trains the model based on the training files
        # @param train_files corpus of documents as a list of strings
        # @param method "fast-learn", "learn" or "deep-learn"
        # @param workes number of parallel workers
        # @param min_count minimum number of documents where a word must be to be valid
        self.train_data = train_data
        corpus = train_data[0]

        self.model = Top2Vec(documents=corpus, embedding_model=embedding_model, min_count=min_count,
            ngram_vocab=ngram, speed=method, workers=workers, document_chunker="sequential",
            use_embedding_model_tokenizer=tokenizer)

        self.print_model_summary()

    def test_model(self, corpus, associated_SDGs, path_to_plot="", path_to_excel="", only_bad=False,
                   score_threshold=3.0, only_positive=False, filter_low=False, expand_factor=1.0,
                   version=1, normalize=True, normalize_threshold=0.25):
        rawSDG = []; rawSDGseg = []
        predictedSDGs = []
        realSDGs = []
        scoresSDGs = []
        valids = []
        validsAny = []
        texts = []
        statsGlobal = []
        countPerSDG = np.zeros(17)
        countWellPredictionsPerSDG = np.zeros(17)
        probs_per_sdg = [[] for ii in range(1,18)]
        maxSDG = 0.0

        numTopics = self.model.get_num_topics()
        stat_topics = numTopics
        for text, sdgs in zip(corpus, associated_SDGs):
            raw_sdgs, predic, score, raw_topicsScores = self.map_text_to_sdgs(text, score_threshold=score_threshold,
                only_positive=only_positive, version=version, expand_factor=expand_factor, filter_low=filter_low,
                normalize=normalize, normalize_threshold=normalize_threshold)

            maxLocal = max(raw_sdgs)
            if maxLocal > maxSDG: maxSDG = maxLocal

            validSingle = False; ii = 0
            for sdg in sdgs:
                countPerSDG[sdg - 1] += 1
                probs_per_sdg[sdg - 1].append(raw_sdgs[sdg - 1])
                if sdg in predic:
                    validSingle = True
                    ii += 1
```

```
                countWellPredictionsPerSDG[sdg - 1] += 1
            valid = False
            if ii == len(sdgs):
                valid = True

            if (only_bad and not(valid)) or not(only_bad):

                raw_sdgsAscii = ["x{}: {:.3f}".format(xx, topic) for topic, xx in zip(raw_sdgs, range(1,18))]
                raw_sdgsAscii = "|".join(raw_sdgsAscii)
                rawSDG.append(raw_sdgsAscii)

                stats = [min(raw_sdgs), np.mean(raw_sdgs), max(raw_sdgs)]
                statsAscii = "[{:.2f}, {:.2f}, {:.2f}]".format(stats[0], stats[1], stats[2])
                statsGlobal.append(statsAscii)
                predictedSDGs.append(predic)
                realSDGs.append(sdgs)
                scoresSDGs.append(score)
                texts.append(text)
            valids.append(valid)
            validsAny.append(validSingle)

        oks = [ok for ok in valids if ok == True]
        oksSingle = [ok for ok in validsAny if ok == True]
        perc_global = len(oks) / len(valids) * 100
        perc_single = len(oksSingle) / len(valids) * 100
        print("- {:.2f} % valid_global, {:.3f} % valid_any, of {} files".format(perc_global, perc_single,
            len(valids)))
        print('Max_found: {:.3f}'.format(maxSDG))

        for probs, index in zip(probs_per_sdg, range(len(probs_per_sdg))):
            probs_per_sdg[index] = np.mean(probs_per_sdg[index])

        if len(path_to_excel) > 0:
            df = pd.DataFrame()
            df["text"] = texts
            df["real"] = realSDGs
            df["sdgs_association"] = rawSDG
            df["stats"] = statsGlobal
            df["prediction"] = predictedSDGs
            df["scores"] = scoresSDGs
            df[" all_valid "] = valids
            df["any_valid"] = validsAny
            df.to_excel(path_to_excel)

        return [perc_global, perc_single, probs_per_sdg, maxSDG, predictedSDGs]

    def map_model_topics_to_sdgs(self, path_csv="", normalize=False, version=1):
        # maps each internal topic with the SDGs. A complete text associated to each specific SDG is fetched.
        # Then each topic is compared with each text and the text-associated sdg with the maximum score is selected
        # as the SDG.
        if version == 1:
            nTopics = self.model.get_num_topics()
            topic_sizes, topics_num = self.model.get_topic_sizes()
            self.topics_association = np.zeros((nTopics, 17))
            for text, labeled_sdgs in zip(self.train_data[0], self.train_data[1]):
                topics, probs = self.infer_text(text)
                for topicIndex, score in zip(topics, probs):
                    # if subScore < meanSub: continue
                    if score < 0.0: continue
                    for sdg in labeled_sdgs:
                        tmp = np.zeros(17)
                        tmp[sdg - 1] = 1
                        self.topics_association[topicIndex] += score * tmp

            sum_per_topic = np.zeros(17)
            for ii in range(nTopics):
```

```
            if normalize:
                norm_topics = self.topics_association[ii] / sum(self.topics_association[ii])
                for nn in norm_topics:
                    if nn < 0.1:
                        norm_topics[list(norm_topics).index(nn)] = 0.0
                norm_topics = norm_topics / sum(norm_topics)
                self.topics_association[ii] = norm_topics

                sum_per_topic += self.topics_association[ii]
            if self.verbose:
                listAscii = ["x{}:{:.3f}".format(xx, sdg) for xx, sdg in zip(range(1,18),
                    self.topics_association[ii])]
                print('Topic{:2d}: '.format(ii), '|'.join(listAscii))
        if self.verbose:
            listAscii = ["x{}:{:.3f}".format(xx, sdg) for xx, sdg in zip(range(1,18), sum_per_topic)]
            final_sum = 'Sum_total: ' + '|'.join(listAscii)
            print(final_sum)
    else:
        sum_per_topic = np.zeros(17)
        nTopics = self.model.get_num_topics()
        topic_sizes, topics_num = self.model.get_topic_sizes()
        self.topics_association = np.zeros((nTopics, 17))
        for topicIndex in range(nTopics):
            numDocs = topic_sizes[topicIndex] # all the associated documents to that topic
            documents, document_scores, document_ids = self.model.search_documents_by_topic(topic_num=topicIndex,
                num_docs=numDocs)
            if normalize:
                document_scores = document_scores / sum(document_scores)

            sdgs = np.zeros(17)
            for docId, score in zip(document_ids, document_scores):
                labeled_sdgs = self.train_data[1][docId]
                for sdg in labeled_sdgs:
                    sdgs[sdg - 1] += score * 1

            if normalize:
                norm_topics = sdgs / sum(sdgs)
                for nn in norm_topics:
                    if nn < 0.1:
                        norm_topics[list(norm_topics).index(nn)] = 0.0
                norm_topics = norm_topics / sum(norm_topics)
                sdgs = norm_topics

            self.topics_association[topicIndex] = sdgs
            sum_per_topic += sdgs
            if self.verbose:
                listAscii = ["x{}:{:.3f}".format(xx, sdg) for xx, sdg in zip(range(1,18),
                    self.topics_association[topicIndex])]
                print('Topic{:2d}: '.format(topicIndex), '|'.join(listAscii))
        if self.verbose:
            listAscii = ["x{}:{:.3f}".format(xx, sdg) for xx, sdg in zip(range(1,18), sum_per_topic)]
            final_sum = 'Sum_total: ' + '|'.join(listAscii)
            print(final_sum)

    if len(path_csv) > 4:
        dfMap = pd.DataFrame()
        rows = []
        sum_per_sdg = np.zeros(17)
        for ii in range(nTopics):
            listAscii = ["x{}:{:.3f}".format(xx, sdg) for xx, sdg in zip(range(1,18),
                self.topics_association[ii])]
            sum_per_sdg += self.topics_association[ii]
            rows.append('|'.join(listAscii))
        sum_ascii = ["x{}:{:.3f}".format(xx, sdg) for xx, sdg in zip(range(1,18), sum_per_sdg)]
        rows.append('|'.join(sum_ascii))
        dfMap["topics_association_map"] = rows
```

```
        dfMap.to_excel(self.paths["out"] + "Top2vec/" + "topics_map.xlsx")

        #np.savetxt(path_csv, self.topics_association, delimiter=",")

        # Then the mapping result is stored in a csv
        topic_words, word_scores, topic_nums = self.model.get_topics()
        df = pd.DataFrame()

        topic_words_ascii = [[] for ii in range(nTopics)]
        for words, scores, topicIndex in zip(topic_words, word_scores, range(nTopics)):
            for word, score in zip(words, scores):
                topic_words_ascii[topicIndex].append("{:.3f}:{}".format(score, word))

        # all the top SDGs with an score > 0.1 are considered for that topic
        for topicIndex in range(nTopics):
            topSDGs = sorted(self.topics_association[topicIndex], reverse=True)
            title = []
            for sdg in topSDGs:
                if sdg < 0.1: break
                sdgIndex = list(self.topics_association[topicIndex]).index(sdg)
                ass_sdg = self.topics_association[topicIndex][sdgIndex]
                title.append("{:.2f}*SDG{}".format(ass_sdg, sdgIndex + 1))
            title = ",".join(title)
            df[title] = topic_words_ascii[topicIndex]
        df.to_csv(path_csv)

    return [sum_per_topic, final_sum]

def map_text_to_sdgs(self, text, score_threshold, only_positive=False, version=1, expand_factor=3,
    filter_low =True, normalize=True, normalize_threshold=0.25):
    if version == 1: # then map with topics
        numTopics = self.model.get_num_topics()
        topics_words, word_scores, probabilities, topic_nums = self.model.query_topics(text,
            num_topics=numTopics)
        predictSDGs = np.zeros(17)
        for topicIndex, topicScore in zip(topic_nums, probabilities):
            if only_positive and (topicScore < 0): break
            predictSDGs += topicScore * self.topics_association[topicIndex]
    elif version == 2: # then map with documents
        maxDocs = len(self.train_data[0])  # number of texts with which it was trained
        documents, probabilities, doc_ids = self.model.query_documents(text, num_docs=maxDocs)
        predictSDGs = np.zeros(17)
        for docIndex, docScore in zip(doc_ids, probabilities):
            if only_positive and (docScore < 0): break
            sdgs = np.zeros(17)
            labeled_sdgs = self.train_data[1][docIndex]
            for sdg in labeled_sdgs:
                sdgs[sdg - 1] = 1
            predictSDGs += docScore * sdgs
    # POST-PROCESSING OF THE MEASURES
    predictSDGs *= expand_factor

    if filter_low:
        raw_sdgs_filt = predictSDGs < 0.05
        for prob, index, filt in zip(predictSDGs, range(len(predictSDGs)), raw_sdgs_filt):
            if filt:
                prob = predictSDGs[index]
                predictSDGs[index] = 0.0
                predictSDGs += prob * predictSDGs / sum(predictSDGs)

    if normalize:
        # raw_sdgs_filt = predictSDGs < normalize_threshold
        # for index, filt in zip(range(len(predictSDGs)), raw_sdgs_filt):
        #     if filt:
        #         predictSDGs[index] = 0.0
        predictSDGs = predictSDGs / sum(predictSDGs)
```

```
        top = sorted(predictSDGs, reverse=True)
        sdgs = []; scores = []
        for ii in range(len(top)):
            if top[ii] >= score_threshold:
                sdgs.append(list(predictSDGs).index(top[ii]) + 1)
                scores.append(top[ii])

        return [predictSDGs, sdgs, scores, probabilities]


    def map_text_to_topwords(self, text, top_n):
        topics_words, word_scores, topic_nums = self.model.get_topics()

        words_collection = []
        topics, probs = self.infer_text(text)
        for topic, prob in zip(topics, probs):
            for word, score in zip(topics_words[topic], word_scores[topic]):
                words_collection.append((word, score * prob))

        def sort_method(elem):
            return elem[1]

        words_collection.sort(key=sort_method, reverse=True)
        return words_collection[:top_n]


    def infer_text(self, text):
        nTopics = self.model.get_num_topics()
        t1, t2, topic_scores , topic_nums = self.model.query_topics(text, nTopics)
        return [topic_nums, topic_scores]

    def print_model_summary(self):
        # print('####### Model summary:')
        print('- Number_of_topics:', self.model.get_num_topics())

    def get_topics_from_model(self, model, n_top_words):
        # Returns the n_top_words for each of the n_topics with which a model has been trained
        word_dict = dict()
        topicsRaw = model.show_topics(num_topics=model.num_topics, num_words=n_top_words)
        topicsParsed = []
        for topic in topicsRaw:
            topicStr = topic[1]
            words = []
            for comb in topicStr.split('+'):
                coef, word = comb.split('*')
                coef = float(coef)
                word = word.replace("'",'')
                words.append([coef, word])
            topicsParsed.append(words)
        return topicsParsed
```

Listing A.6: model_bertopic.py

```
from signal import valid_signals
import tools
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tools
import warnings
from bertopic import BERTopic
warnings.filterwarnings('ignore')


class BERTopic_classifier:
```

```
paths=[]
model=[]
nTopics=[]
 topics_association =[]
verbose=False
trainData=[]

def __init__(self, paths):
    self.paths = paths

def set_conf(self, paths, dict, verbose=False):
    self.paths = paths
    self.dict = dict
    self.verbose = verbose

def train_model(self, train_data,
                embedding_model="all-MiniLM-L6-v2", # Others: all-MiniLM-L6-v2, all-MiniLM-L12-v2,
                all-mpnet-base-v2
                n_gram_range=(1,3), # the default parameter is (1,1)
                top_n_words=10, min_topic_size=10, # default parameters
                nr_topics=None, # reduce the number of topics to this number
                diversity=None, # value can be used between 0, 1
                 calculate_probabilities=True,
                 seed_topic_list=None,
                verbose=True
                ):
    # trains the model based on the training files
    # @param train_files corpus of documents as a list of strings
    # @param method "fast-learn", "learn" or "deep-learn"
    # @param workes number of parallel workers
    # @param min_count minimum number of documents where a word must be to be valid
    self.trainData = train_data
    corpus = train_data[0]; associated_sdgs = train_data[1]

    self.model = BERTopic(language="english",
                      embedding_model=embedding_model,
                      top_n_words=top_n_words,
                      min_topic_size=min_topic_size,
                      n_gram_range=n_gram_range,
                      nr_topics=nr_topics,
                       calculate_probabilities=calculate_probabilities,
                       seed_topic_list=seed_topic_list,
                      verbose=verbose)

    topics, probs = self.model.fit_transform(corpus,
                     embeddings=None, # use the sentence-transformer model
                     y=None # target class for semisupervised. not applicable
                     )
    self.nTopics = len(self.model.get_topics())
    if topics.count(-1) > 0: self.nTopics -= 1 # there is 1 outlier topic
    counts_per_topic = ["T{}: ".format(ii) + str(topics.count(ii)) for ii in range(0, self.nTopics)]
    print('## nDocs_per_topic: ' + " | ".join(counts_per_topic))

    self.map_model_topics_to_sdgs(associated_sdgs, topics, probs, normalize=True, verbose=True)

def test_model(self, corpus, associated_SDGs, stat_topics=-1, path_to_plot="", path_to_excel="",
               only_bad=False, score_threshold=3.0, only_positive=True, filter_low=True,
               expand_factor=1.0, normalize=False):
    rawSDG = []; rawSDGseg = []
    predictedSDGs = []; realSDGs = []
    scoresSDGs = []
    valids = []; validsAny = []
    texts = []
    statsGlobal = []
    countPerSDG = np.zeros(17)
    countWellPredictionsPerSDG = np.zeros(17)
```

```
    probs_per_sdg = [[] for ii in range(1,18)]
    maxSDG = 0.0

    topics, probs = self.model.transform(corpus) # transforms the entire corpus

    for text, sdgs, prob in zip(corpus, associated_SDGs, probs):
        raw_sdgs, predic, score = self.map_text_to_sdgs_with_probs(prob, score_threshold=score_threshold,
            only_positive=only_positive, expand_factor=expand_factor, filter_low=filter_low,
            normalize=normalize)

        maxLocal = max(raw_sdgs)
        if maxLocal > maxSDG: maxSDG = maxLocal

        validSingle = False; ii = 0
        for sdg in sdgs:
            countPerSDG[sdg - 1] += 1
            probs_per_sdg[sdg - 1].append(raw_sdgs[sdg - 1])
            if sdg in predic:
                validSingle = True
                ii += 1
                countWellPredictionsPerSDG[sdg - 1] += 1
        valid = False
        if ii == len(sdgs):
            valid = True

        if (only_bad and not(valid)) or not(only_bad):
            raw_sdgsAscii = ["x{}: {:.3f}".format(xx, topic) for topic, xx in zip(raw_sdgs, range(1,18))]
            raw_sdgsAscii = "|".join(raw_sdgsAscii)
            rawSDG.append(raw_sdgsAscii)

            stats = [min(raw_sdgs), np.mean(raw_sdgs), max(raw_sdgs)]
            statsAscii = "[{:.2f}, {:.2f}, {:.2f}]".format(stats[0], stats[1], stats[2])
            statsGlobal.append(statsAscii)
            predictedSDGs.append(predic)
            realSDGs.append(sdgs)
            scoresSDGs.append(score)
            texts.append(text)
        valids.append(valid)
        validsAny.append(validSingle)

    oks = [ok for ok in valids if ok == True]
    oksSingle = [ok for ok in validsAny if ok == True]
    perc_global = len(oks) / len(valids) * 100
    perc_single = len(oksSingle) / len(valids) * 100
    print("- {:.2f} % valid_global, {:.3f} % valid_any, of {} files".format(perc_global,
        perc_single, len(valids)))
    print('Max_found: {:.3f}'.format(maxSDG))

    for probs, index in zip(probs_per_sdg, range(len(probs_per_sdg))):
        probs_per_sdg[index] = np.mean(probs_per_sdg[index])

    if len(path_to_excel) > 0:
        df = pd.DataFrame()
        df["text"] = texts
        df["real"] = realSDGs
        df["sdgs_association"] = rawSDG
        df["stats"] = statsGlobal
        df["prediction"] = predictedSDGs
        df["scores"] = scoresSDGs
        df[" all_valid "] = valids
        df["any_valid"] = validsAny
        df.to_excel(path_to_excel)

    return predictedSDGs, maxSDG, perc_global, perc_single

def map_model_topics_to_sdgs(self, associated_sdgs, topics, probs, path_csv="", normalize=True, verbose=True):
```

```
        # maps each internal topic with the SDGs. A complete text associated to each specific SDG is fetched.
        # Then each topic is compared with each text and the text-associated sdg with the maximum score is selected
        # as the SDG.
        self.topics_association = np.zeros((self.nTopics, 17))
        for sdgs, main_topic, prob in zip(associated_sdgs, topics, probs):
            if main_topic < 0:
                continue # outlier topic is discarded
            tmp = np.zeros(17)
            for sdg in sdgs:
                tmp[sdg - 1] = 1
            for topic_index, prob_topic in zip(range(self.nTopics), prob):
                self.topics_association[topic_index, :] += tmp * prob_topic

        sum_per_topic = np.zeros(17)
        for topic_index in range(self.nTopics):
            if normalize:
                tmp = self.topics_association[topic_index, :]
                norm_topics = tmp / sum(tmp)

                for nn in norm_topics:
                    if nn < 0.1:
                        norm_topics[list(norm_topics).index(nn)] = 0.0
                norm_topics = norm_topics / sum(norm_topics)

                self.topics_association[topic_index, :] = norm_topics

            sum_per_topic += self.topics_association[topic_index, :]
            if verbose:
                listAscii = ["x{}:{:.3f}".format(xx, sdg) for xx, sdg in zip(range(1,18),
                    self.topics_association[topic_index, :])]
                print('Topic{:2d}: '.format(topic_index), '|'.join(listAscii))

        if verbose:
            listAscii = ["x{}:{:.3f}".format(xx, sdg) for xx, sdg in zip(range(1,18), sum_per_topic)]
            final_sum = 'Sum_total: ' + '|'.join(listAscii)
            print(final_sum)

        if len(path_csv) > 4:
            all_topics = topic_model.get_topics()

            raise ValueError('Association_map_to_csv_not_supported')

    def map_text_to_sdgs(self, text, score_threshold, only_positive=False,
                         expand_factor=3, filter_low=True, normalize=False):
        topics, probs = self.model.transform(text)
        [predictSDGs, sdgs, scores] = self.map_text_to_sdgs_with_probs(probs, score_threshold=score_threshold,
            only_positive=only_positive, expand_factor=expand_factor, filter_low=filter_low, normalize=normalize)

        return [predictSDGs, sdgs, scores]

    def map_text_to_topwords(self, text, top_n):
        topics_inf = self.model.get_topics()
        words_collection = []
        topics, probs = self.model.transform(text)
        topics = range(len(probs[0]))
        for topic, prob in zip(topics, probs[0]):
            for pair in topics_inf[topic]:
                wrd = pair[0]; score = pair[1]
                words_collection.append((wrd, score * prob))

        def sort_method(elem):
            return elem[1]

        words_collection.sort(key=sort_method, reverse=True)
        return words_collection[:top_n]
```

```
    def map_text_to_sdgs_with_probs(self, probs, score_threshold, only_positive=False,
                                    expand_factor=3, filter_low=True, normalize=False):
        predictSDGs = np.zeros(17)
        for topicIndex, topicScore in zip(range(self.nTopics), probs):
            if only_positive and topicScore < 0: continue
            predictSDGs += topicScore * self.topics_association[topicIndex]

        predictSDGs *= expand_factor

        if filter_low:
            raw_sdgs_filt = predictSDGs < 0.05
            for prob, index, filt in zip(predictSDGs, range(len(predictSDGs)), raw_sdgs_filt):
                if filt:
                    prob = predictSDGs[index]
                    predictSDGs[index] = 0.0
                    predictSDGs += prob * predictSDGs / sum(predictSDGs)

        if normalize: predictSDGs = predictSDGs / sum(predictSDGs)

        top = sorted(predictSDGs, reverse=True)
        sdgs = []; scores = []
        for ii in range(len(top)):
            if top[ii] >= score_threshold:
                sdgs.append(list(predictSDGs).index(top[ii]) + 1)
                scores.append(top[ii])

        return [predictSDGs, sdgs, scores]

    def print_summary(self, path_csv=""):
        topic_words_scores = [[] for ii in range(self.nTopics)]
        dfTopics = pd.DataFrame()
        all_topics = self.model.get_topics()
        for topic_index in range(self.nTopics):
            topic_words = all_topics[topic_index]
            for topic_tuple in topic_words:
                word = topic_tuple[0]; score = topic_tuple[1]
                word_str = "{:.3f}:{}".format(score, word)
                topic_words_scores[topic_index].append(word_str)

            topicName = "Topic{}".format(topic_index)
            dfTopics[topicName] = topic_words_scores[topic_index]

        if len(path_csv) > 0:
            dfTopics.to_csv(path_csv)
```

Listing A.7: test_nmf.py

```
# Configures the project paths: they can be launched from any code
import sys, os
sys.path.append(os.path.realpath('.'))
import conf
conf.import_paths()

# CONFIGURATION FLAGS
flag_optimize = 0

# Real imports required by the file for work properly
import model_nmf
from logging import error
import data
import conf
import pandas as pd
import numpy as np
import tools

# Loads all the datasets
```

```
paths = conf.get_paths()
ds_train = data.get_dataset(requires_update=False, filter=["org", "manual_extra"])
ds_valid_short = data.get_dataset(requires_update=False, filter=["nature_abstract"])
ds_valid_long = data.get_dataset(requires_update=False, filter=["nature_all"])

path_out = "out/NMF/"

if flag_optimize:
    optimData = pd.read_excel(paths["ref"] + "optimization_nmf.xlsx")
    nTopics = optimData["num_topics"]
    nIterations = optimData["iterations"]
    nature = optimData["nature"]
    stemming = optimData["stemming"]
    score_threshold = optimData["score_threshold"]
    l1 = optimData["l1"]
    alpha_w = optimData["alpha_w"]
    alpha_h = optimData["alpha_h"]

    res_any = []; res_all = []

    for ii in range(len(nTopics)):
        print("# Optimizing case: {}, nTopic: {}, nIterations: {}, nature: {}, Stemming: {},
        L1: {:.2f}, Alphaw: {:.2f}, AlphaH: {:.2f}".format(ii, nTopics[ii], nIterations[ii],
          nature[ii], stemming[ii], l1[ii], alpha_w[ii], alpha_h[ii]))

        if stemming[ii]: type_texts = "lem_stem"
        else: type_texts = "lem"

        # all text should have been processed in the same way
        orgFiles = ds_train[type_texts]; sdgs_org = ds_train["sdgs"]
        natureShort = ds_valid_short[type_texts]; sdgs_natureShort = ds_valid_short["sdgs"]
        natureLong = ds_valid_long[type_texts]; sdgs_natureLong = ds_valid_long["sdgs"]

        if nature[ii]: trainData = [orgFiles + natureShort, sdgs_org + sdgs_natureShort]
        else: trainData = [orgFiles, sdgs_org]

        try:
            print('# Training_model...')
            nmf = model_nmf.NMF_classifier(paths, verbose=True)

            nmf.train(train_data=trainData, n_topics=nTopics[ii], ngram=(1,3), min_df=2,
                max_iter=nIterations[ii],
                    l1=l1[ii], alpha_w=alpha_w[ii], alpha_h=alpha_h[ii])
            nmf.map_model_topics_to_sdgs(n_top_words=50, normalize=True, path_csv=path_out +
                "topics_map{}.csv".format(ii))

            tools.save_obj(nmf, paths["model"] + "nmf{}.pickle".format(ii))

            print('# Testing_model...')
            filter = True; normalize = False

            [rawSDG, perc_valid_global, perc_valid_any, maxSDG, pred_sdgs] = nmf.test_model(corpus=trainData[0],
             associated_SDGs=trainData[1], score_threshold=score_threshold[ii], segmentize=-1,
              filter_low =filter, normalize=normalize,
                        path_to_excel=(path_out + "test_nmf_training{}.xlsx".format(ii)))
            tools.plot_ok_vs_nok_SDGsidentified(trainData[1], pred_sdgs, path_out + "sdgs_train{}.png".format(ii))

            expandFactor = 4
            [rawSDG, perc_valid_global, perc_valid_any, maxSDG, pred_sdgs] = nmf.test_model(corpus=natureShort,
             associated_SDGs=sdgs_natureShort, score_threshold=score_threshold[ii],
                        segmentize=-1, path_to_excel=(path_out + "test_nmf_natureS{}.xlsx".format(ii)),
                        normalize=normalize, filter_low=filter, expand_factor=expandFactor)
            tools.plot_ok_vs_nok_SDGsidentified(sdgs_natureShort, pred_sdgs, path_out +
                "sdgs_test{}.png".format(ii))

        except:
```

```
            print('# Aborting execution of iteration{}'.format(ii))
            perc_valid_any = -1; perc_valid_global = -1

        res_any.append(perc_valid_any); res_all.append(perc_valid_global)

    outData = optimData
    outData["any"] = res_any
    outData["all"] = res_all
    outData.to_excel(path_out + "optimization{}.xlsx".format(ii))
else:
    print('# Using default-user_configuration...')

    type_texts = "lem"
    nTopics = 20; maxIter = 1000; l1 = 0.0; alpha_w = 0.0; alpha_h = 0.0
    score = 0.1


    orgFiles = ds_train[type_texts]; sdgs_org = ds_train["sdgs"]
    natureShort = ds_valid_short[type_texts]; sdgs_natureShort = ds_valid_short["sdgs"]
    natureLong = ds_valid_long[type_texts]; sdgs_natureLong = ds_valid_long["sdgs"]

    trainData = [orgFiles, sdgs_org]

    print('# Training_model...')
    nmf = model_nmf.NMF_classifier(paths, verbose=True)

    nmf.train(train_data=trainData, n_topics=nTopics, ngram=(1,3), min_df=2, max_iter=maxIter,
            l1=l1, alpha_w=alpha_w, alpha_h=alpha_h)
    nmf.print_stopwords(path_out + "stopwords.csv")
    nmf.map_model_topics_to_sdgs(n_top_words=50, normalize=True, path_csv=path_out + "topics_map.csv")

    tools.save_obj(nmf, paths["model"] + "nmf.pickle")

    print('# Testing_model...')
    filter = True; normalize = False; expandFactor = 4.0

    [rawSDG, perc_valid_global, perc_valid_any, maxSDG, pred_sdgs] = nmf.test_model(corpus=trainData[0],
     associated_SDGs=trainData[1], score_threshold=score, segmentize=-1, filter_low=filter, normalize=normalize,
                path_to_excel=(path_out + "test_nmf_training.xlsx"), expand_factor=expandFactor)
    tools.plot_ok_vs_nok_SDGsidentified(trainData[1], pred_sdgs, path_out + "sdgs_train.png")

    [rawSDG, perc_valid_global, perc_valid_any, maxSDG, pred_sdgs] = nmf.test_model(corpus=natureShort,
        associated_SDGs=sdgs_natureShort, score_threshold=score,
                segmentize=-1, path_to_excel=(path_out + "test_nmf_natureS.xlsx"),
                normalize=normalize, filter_low=filter, expand_factor=expandFactor)
    tools.plot_ok_vs_nok_SDGsidentified(sdgs_natureShort, pred_sdgs, path_out + "sdgs_test.png")

    pred_sdgs = pd.DataFrame(pred_sdgs)
    pred_sdgs.to_csv(paths["out"] + "ALL/Individual/pred_test_nmf.csv")
```

Listing A.8: test_all.py

```
# Configures the project paths: they can be launched from any code
from pkgutil import iter_importers
import sys, os
sys.path.append(os.path.realpath('.'))
import conf
conf.import_paths()


# Real imports required by the file for work properly
from logging import error
import data
import conf
import pandas as pd
import model_global
```

```
import numpy as np
import tools


# Loads all the datasets
print('# Loading datasets...')
paths = conf.get_paths()
ds_train = data.get_dataset(requires_update=False, filter=["org", "manual_extra"])
ds_valid_short = data.get_dataset(requires_update=False, filter=["nature_abstract"])
ds_valid_long = data.get_dataset(requires_update=False, filter=["nature_all"])

raw_orgFiles = ds_train["standard"]
raw_natureShort = ds_valid_short["standard"]
raw_natureExt = ds_valid_long["standard"]

orgFiles = ds_train["lem"]; sdgs_org = ds_train["sdgs"]
natureShort = ds_valid_short["lem"]; sdgs_natureShort = ds_valid_short["sdgs"]
natureLong = ds_valid_long["lem"]; sdgs_natureLong = ds_valid_long["sdgs"]

# filters those long texts which are not the same of the abstracts. They are required to be the same
# in order to do the comparisons
xx, yy, index_abstracts = data.get_nature_abstracts()
xx, yy, index_full = data.get_nature_files(abstract=True, kw=False, intro=True,
    body=True, concl=True)
flags_filter = []
for index in index_abstracts:
    flags_filter.append(index_full.index(index))

raw_natureExt = [raw_natureExt[index] for index in flags_filter]
natureLong = [natureLong[index] for index in flags_filter]
sdgs_natureLong = [sdgs_natureLong[index] for index in flags_filter]


print('# Loading models...')
model = model_global.Global_Classifier(paths=paths, verbose=True)
model.load_models()

print('# Testing train dataset...')
predic, scores, predicStr = model.test_model(raw_corpus=raw_orgFiles, corpus=orgFiles,
    associated_SDGs=sdgs_org,
                path_to_plot="",
                path_to_excel=paths["out"] + "All/test_training.xlsx",
                only_bad=False, score_threshold=-1, only_positive=True, filter_low=True)
tools.plot_ok_vs_nok_SDGsidentified(sdgs_org, predic, paths["out"] + "All/" + "sdgs_train.png")


print('# Testing validation dataset (short)...')
predic, scores, predicStr = model.test_model(raw_corpus=raw_natureShort, corpus=natureShort,
    associated_SDGs=sdgs_natureShort,
                path_to_plot="",
                path_to_excel=paths["out"] + "All/test_nature_short.xlsx",
                only_bad=False, score_threshold=-1, only_positive=True, filter_low=True)
tools.plot_ok_vs_nok_SDGsidentified(sdgs_natureShort, predic, paths["out"] + "All/" + "sdgs_test_short.png")


print('# Testing validation dataset (long)...')
predic, scores, predicStr = model.test_model(raw_corpus=raw_natureExt, corpus=natureLong,
    associated_SDGs=sdgs_natureLong,
                path_to_plot="",
                path_to_excel=paths["out"] + "All/test_nature_long.xlsx",
                only_bad=False, score_threshold=-1, only_positive=True, filter_low=True)
tools.plot_ok_vs_nok_SDGsidentified(sdgs_natureLong, predic, paths["out"] + "All/" + "sdgs_test_long.png")
```

Listing A.9: aero_analysis.py

```
# Analysis of the results obtained with the Aero database
```

```
# Configures the project paths: they can be launched from any code
from cProfile import label
from pkgutil import iter_importers
import sys, os
sys.path.append(os.path.realpath('.'))
import conf
conf.import_paths()

# Configuration flags
identify_sdgs = False # true: all the texts are identified, false: it used previous stored data

# Imports required to work properly
from logging import error
import data
import conf
import pandas as pd
import model_global
import numpy as np
import tools
import matplotlib.pyplot as plt
import warnings
import os

print('# Loading aero dataset...')
paths = conf.get_paths()
ds_aero = data.get_dataset(requires_update=False, filter=["aero"])
raw_files = ds_aero["standard"]; files = ds_aero["lem"]

print('# Loading models...')
model = model_global.Global_Classifier(paths=paths, verbose=True)
model.load_models()

if identify_sdgs:
    print('# Identifying SDGs in texts...')
    predic, scores, predicStr = model.test_model(raw_corpus=raw_files, corpus=files, associated_SDGs=[],
                path_to_plot="", path_to_excel=paths["out"] + "All/test_aero.xlsx",
                only_bad=False, score_threshold=-1, only_positive=True, filter_low=True)
    ds_aero["id_sdgs"] = predicStr
    pd.DataFrame(ds_aero).to_excel(paths["out"] + "All/df_test_aero.xlsx")
    print('# Results were updated')

def get_sdgs_scores(row_sdgs:str):
    try:
        elems = row_sdgs.split(',')
        scores = []; sdgs=[]
        for elem in elems:
            scores.append(float(elem.split(':')[0]))
            sdgs.append(float(elem.split(':')[1]))
    except:
        print('# Input: ')
        scores = []; sdgs=[]
    return scores, sdgs

def parse_list ( sdgs_list ):
    scores = []; sdgs = []
    for sdg in sdgs_list :
        sc, sd = get_sdgs_scores(sdg)
        scores.append(sc); sdgs.append(sd)
    return scores, sdgs

ds = pd.read_excel(paths["out"] + "All/df_test_aero.xlsx")
list_scores , list_sdgs = parse_list ( list (ds["id_sdgs"]))

print('# Obtaining total number of SDGs identified')
tools.plot_SDGsidentified( list_sdgs , list_scores , with_score=True, fontsize=14,
```

```
    path_out=paths["out"] + "All/total_weight_sdgs.png")
tools.plot_SDGsidentified( list_sdgs , list_scores , with_score=False, fontsize=14,
    path_out=paths["out"] + "All/sdgs_identified.png")

print('# Obtaining the evolution of the SDGs with the years')
def plot_evolution_with_years( fontsize :int=14):
    plt.figure( figsize =(8, 8))
    positions = [-0.15, -0.05, 0.05, 0.15]
    labels = ["SDG7", "SDG9", "SDG11", "SDG13"]
    colors = ["yellow", "blue", "red", "green"]
    years = range(2017, 2022)
    sdgsPerYear = [[], [], [], []]

    for year in years:
        df = ds.loc[ds['years'] == year]
        list_scores , list_sdgs = parse_list (list(df["id_sdgs"]))
        counts, countstr = tools.count_texts_per_score( list_sdgs , list_scores )

        sdgsPerYear[0].append(counts[6]); sdgsPerYear[1].append(counts[8])
        sdgsPerYear[2].append(counts[10]); sdgsPerYear[3].append(counts[12])
    for ii in range(4):
        xx = 2017
        plt.bar(xx + positions[ii], sdgsPerYear[ii][0], width=0.1, alpha=1, color=colors[ii],
            label=labels[ii])

    for ii in range(4):
        for jj in range(2018, 2022):
            plt.bar(jj + positions[ii], sdgsPerYear[ii][years.index(jj)], width=0.1, alpha=1,
                color=colors[ii])

    plt.xticks(years)
    plt.xlabel('Year', fontsize=fontsize)
    plt.ylabel("Total_weight_(sum_of_individual_score)", fontsize=fontsize)
    plt.tick_params(axis='x', labelsize=fontsize)
    plt.tick_params(axis='y', labelsize=fontsize)
    plt.legend()
    plt.savefig(paths["out"] + "All/evolution_sdgs_year.png")

plot_evolution_with_years( fontsize=14)

print('# Plotting the contribution of the papers < median and above')
citations = list(ds["citations"])
dfOrdered = ds.sort_values(by=['citations'])
median_index = len(list_scores) // 2
print('# The median is: {}'.format(list(dfOrdered["citations"])[median_index]))
list_scores , list_sdgs = parse_list (list(dfOrdered["id_sdgs"]))

print([len( list_scores ), median_index])


def compare_lower_higher_citations( fontsize=14):
    xlabel = [ii for ii in range(1, 18)]
    countsLow, countstr = tools.count_texts_per_score( list_sdgs [:median_index],
        list_scores [:median_index])
    countsHigh, countstr = tools.count_texts_per_score( list_sdgs [median_index:],
        list_scores [median_index:])

    plt.figure( figsize =(8, 8))
    plt.bar(np.array(xlabel[0]) - 0.1, countsLow[0], width=0.2, alpha=1.0, color='green',
        label="Lower_than_the_median")
    plt.bar(np.array(xlabel) - 0.1, countsLow, width=0.2, alpha=1.0, color='green')

    plt.bar(np.array(xlabel[1]) + 0.1, countsHigh[0], width=0.2, alpha=1.0, color='red',
        label="Higher_than_the_median")
    plt.bar(np.array(xlabel) + 0.1, countsHigh, width=0.2, alpha=1.0, color='red')
    plt.xticks(xlabel)
```

```
    plt.xlabel('SDG', fontsize=fontsize)
    plt.ylabel("Total_weight_(sum_of_individual_score)", fontsize=fontsize)
    plt.tick_params(axis='x', labelsize=fontsize)
    plt.tick_params(axis='y', labelsize=fontsize)
    path_out = paths["out"] + "All/lower_higher_citations.png"
    plt.legend()
    if os.path.exists(path_out):
        os.remove(path_out) # otherwise, old figures are not overwritten
    plt.savefig(path_out)

compare_lower_higher_citations(fontsize=14)


print('#_Plotting_pie_charts')
countries = list(ds["countries"])
list_countries = []; count_countries = []
for country in countries:
    if not country in list_countries:
        list_countries.append(country)
        count_countries.append(1)
    else:
        count_countries[list_countries.index(country)] += 1
listC = [[country, count] for country, count in zip(list_countries, count_countries)]

def order_second(elem):
    return elem[1]
listC.sort(key=order_second, reverse=True)
a=2
# import matplotlib.pyplot as plt

# # Pie chart, where the slices will be ordered and plotted counter-clockwise:
# labels = 'Frogs', 'Hogs', 'Dogs', 'Logs'
# sizes = [15, 30, 45, 10]
# explode = (0, 0.1, 0, 0)  # only "explode" the 2nd slice (i.e. 'Hogs')

# fig1, ax1 = plt.subplots()
# ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
#         shadow=True, startangle=90)
# ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

# plt.show()
```