# POLITECNICO DI TORINO

Master's Degree in Data Science & Engineering


Politecnico
di Torino

Master's Degree Thesis

# Log Analysis: Topic Modeling applications on fine-features data processing system


Supervisors

Prof. DANIELE APILETTI

EMANUELE GALLO

Candidate

DAVIDE NAPOLITANO


April 2022

# Summary

Nowadays an ever-increasing number of operations is performed with the help of computer systems. They are everywhere around us and each of them produces a huge quantity of log files for each operation carried out.
In this context, the aim of this project is to study and create an architecture able to read and categorize the logs produced by computer systems based on logs content. The outcomes of this project form the foundation of a subsequent task of anomaly detection, which is highly requested since lets companies to detect malicious attacks or failures and whereby the focus is moved towards the most relevant events.
As previously mentioned, I deal with log files. They contain the sequential and chronological records of the operations done by computer systems. Since new systems are extremely complex, the interaction of their software components causes the registration of a big number of operations. Accordingly, analyse and extract useful information from this incredible amount of textual data is complex and it requires many efforts, because both field expertise and cutting-edge technologies must be involved inside the process. The starting point of this thesis is the LogHub dataset repository, where several computer system log types are collected. Specifically, for this research the focus is on logs coming from "HDFS" and "Spark" distributed systems.
One of the issues of this study is the object manipulated: the log. Differently from common texts like books, logs are short texts with a lot of technical words and unavailing elements like path or web keys. For this reason, the manipulation and the cleaning phases are essential to provide useful information to models adopted. Therefore, the first part of the project is the process of making the data more suitable and it is made by several steps. They are principally based on NLP and Regular Expression crafting, which allow to obtain meaningful words from unstructured logs. Some of these techniques are common when dealing with text mining, like the removal of accents and special characters, the expansion of contractions, the tokenization and the PoS tagging. In addition, detailed studies have been conducted on log structure. Indeed, several recurring patterns, corresponding to meaningless elements, have been identified and, through their removal, it is possible to trace logs back to similar structures, reducing differences.

These operations regard reformatting logs error, since usually they are divided into multiple lines, the applications of the Camel Case Split, due to Object based nature of the systems, the upper-case split for exceptions and the removal of patterns like parentheses with their content and variable value association. Moreover, a personal implementation is proposed to lemmatize words that, due to their rare proposition in common texts, are not properly tagged and conducted to their root.
The final steps in the data processing phase are the removal of words depending on their frequency among documents and the production of multi-grams. The last one, provided a sentence, makes mono-gram, bi-gram and tri-gram, while considering to not create elements that contain the same word multiple times or elements with overlapping meaning.
Provided this data processing system, for both "HDFS" and "Spark" three different datasets are made: ORIGIN+MESSAGE, that considers both the source and the message elements of logs, MESSAGE, that takes only the message part, and SPLIT, a version in the middle that considers both the message and the origin but for the latter only monograms are created, in order to decrease its weight in the algorithms.
Afterward, the topic models are implemented. Concisely, they consist in annotating documents with thematic information and in grouping together documents that share similar contents.
About the chosen algorithms, the project is split into two branches. The first one investigates standard approaches, based on probabilistic or algebraical operations, where the models analysed are LDA, NMF and LSA. The second branch, instead, follows a modern approach based on Neural Network and Feature Embedding. For this part the proposed algorithms are ProdLDA, Top2Vec/BERTopic and, by combining many of their approaches, my custom model called GEAC. About the training phase, the datasets final structure must be considered. They are all made by few short entries, making the job hard for the models. For this reason, for all the models a very extensive finetuning had to be carried out, otherwise wrong or meaningful solutions would have been found. All models have been tested with the CV coherence score. Inside the world of coherence scores, the CV is the most meaningful to evaluate the goodness of topic clusters since it provides results close to the human ones, the ground truth for this task.
One issue with Topic Models is about the quality, since the models cited do not always guarantee that each topic cluster is independent from the others. For this reason, I perform an analysis about the similarity between topic clusters by exploiting the spatial representation of each word inside a topic through the Word2Vec feature embedding model. Different possible solutions have been proposed to achieve this result. One aim to keep the model that obtains the highest CV score but, at the same time, it does not have too similar topics. Another solution tries to achieve a good score with a greedy approach: starting from a good result, it decreases the number of topics, depending on the number of similar topics, until

all the topics are not too similar. The last proposal, instead, merges the similar topic, decreasing as consequence the number of topics and their similarity.
These proposals have been tested with classical models, in order to explore and understand many behaviours. Instead, for the algorithms based on neural network, these solutions have been directly included in the training pipeline, since the different training phase combined with the dataset used, required this reasonings.
As consequence, for this project have been necessary the above considerations. However, it must be noticed that the final aim is not to obtain completely dissimilar topics, but topics that can have a level of similarity which does not exceed a threshold beyond which a human would consider the different topic as one. About the value of the threshold, it has been fixed to 85%, since it allows to discriminate topics that are close but, at the same time, different in few details.
Thanks to this analysis, I can select a model that has a high CV metric score and that is also optimal for what the supervisor might infer regarding similarity.
For what concerns results, the performances in classical models are good for NMF and LDA, while LSA is not able to provide meaningful topics. Some issues related to these models are the high number of similar topics when the best model has a high number of topics, and their predisposition to find generally better models with a small number of topics, making them unusable in these cases. Therefore, the nature of the datasets, combined with their short composition, makes the creation of topics difficult, in particular about logs whose topic is unique and that are made up of few terms.
The introduction of Neural Network models introduces several benefits and improvements. ProdLDA seems to be in line or slightly better than LDA, however its utilization is not straightforward. Indeed, it is common to produce mixtures of topics that do not provides a real meaning, but rather make it more challenging to understand the quality of the models as the metric provides good values due to the fit of topics to multi-sentences.
The introduction of BERT marks a positive turning point. BERTopic allows higher scores, however, it has problems related to its formulation. Indeed, the use of HDBSCAN for clustering involves having at least two elements per group. This is a problem for all logs whose corresponding topic is uniquely associated to them, that in this implementation are common to be mis-labelled as outliers. To overcome this drawback, I propose two solutions: in the first one I take each outlier as a cluster with unitary size, in this way those elements are kept rather than being discarded. However, with this solution, it may happen, when the model makes an error, to consider independent clusters when an existing topic is suitable. Directly from this consideration, I present the second solution. For those logs considered as outliers, before assigning a new label, I check if a match with found topics exists. This control is done by looking at the similarity, by using the same approach already proposed in other algorithms. Considering all models analysed so far, I present my

custom implementation. It is based on the combination of classical models, such as LDA and NMF, and modern approaches, represented by BERT and Word2Vec. Then, I follow a workflow similar to BERTopic, for dimensionality reduction I use an AutoEncoder rather than UMAP, and finally I apply a clustering algorithm to group data. Unlike BERTopic, I use two different types of algorithms: K-Means, a standard model that is based on Euclidean distance, and K-Medoids, a similar model that allows to explore the space through cosine similarity.
In this personal proposal, the results further improve, both in terms of coherence and predictions. Indeed, until now the models had several problems on creating some clusters and on making the right associations, while now it is possible to obtain correct topics for all logs analysed. Overall, at the end of this project, a generic log line can be labelled with an almost black box approach that provides optimal outputs both under human and machine perspectives.

The thesis text is structured as follows. After the short introduction on Chapter 1, Chapter 2 provides a global vision about the Application domain and the dataset. Chapter 3 presents a discussion about textual data analysis. Here are presented all the considerations and the steps done to obtain meaningful features from the unstructured logs.
Chapter 4 introduces the readers to Topic Modelling. Here are first explained the algorithm coming from standard approaches based on Algebra and on Probability, like LDA, LSA and NMF. Then are shown the newest approaches that rely on neural networks and new different features manipulation. They are ProdLDA, Top2Vec/BERTopic and, thanks to the reasoning done on them, my personal proposal GEAC done by combining some of their techniques. On Chapter 5 the results on the datasets proposed coming from the different models are disclosed, showing the pros and the cons and, at the same time trying also to overcome same issues related to some of them.
Finally Chapter 6 summarize the whole project in the conclusion, looking into the key contributions and into possible further works.

# Acknowledgements

I would like to thank all the people who have been close to me during these months, who have given me advice and supported me during the whole journey.
In particular, I want to thank my supervisors, Daniele Apiletti and Emanuele Gallo, who, through the collaboration between the Politecnico di Torino and Reply Data, gave me the possibility to face this project.
I want to thank my co-supervisor Valerio Volpe, who guided me during this journey, offering me advices and suggestions.
I thank my family, without their support and their love it would have been impossible to start and finish this journey.
Finally, I would like to thank all my friends and the people closest to me, who in these difficult years have shared the various moments of my life, always with joy and love.

# Table of Contents

**List of Tables** x

**List of Figures** xii

**Acronyms** xvi

**1 Introduction** 1

**2 Application Domain** 3
- 2.1 What are System Logs . . . 3
- 2.2 What is Log Analysis . . . 4
- 2.3 Natural Language Processing . . . 4
  - 2.3.1 Statistical Methods . . . 5
  - 2.3.2 Neural Networks . . . 5
- 2.4 What is Topic Modeling . . . 6
- 2.5 Dataset - LogHub . . . 6

**3 Textual Data Analysis** 9
- 3.1 Data Cleaning . . . 10
  - 3.1.1 Reformat Log Errors . . . 11
  - 3.1.2 Remove Accented Characters . . . 11
  - 3.1.3 Remove Special Characters and Patterns . . . 12
  - 3.1.4 Camel Case Split . . . 15
  - 3.1.5 Merge Improper words divisions . . . 16
  - 3.1.6 Expand Contractions . . . 16
  - 3.1.7 Tokenization . . . 17
  - 3.1.8 StopWords Removal . . . 17
  - 3.1.9 PoS Tagging . . . 17
  - 3.1.10 Lemming . . . 19
  - 3.1.11 Regex . . . 21
  - 3.1.12 Plural to Singular . . . 22

3.2 Data Manipulation . . . . . 22
3.3 Feature Creation . . . . . 24

**4 Topic Modeling** 27
4.1 Classic Approaches . . . . . 27
4.1.1 LDA . . . . . 28
4.1.2 NMF . . . . . 31
4.1.3 LSA . . . . . 35
4.2 Modern Approaches . . . . . 39
4.2.1 ProdLDA . . . . . 49
4.2.2 Top2Vec & BERTopic . . . . . 52
4.2.3 GEAC . . . . . 58

**5 Results** 61
5.1 Coherence Score . . . . . 61
5.2 Topic Similarity . . . . . 63
5.3 Results . . . . . 66
5.3.1 LDA . . . . . 67
5.3.2 NMF . . . . . 75
5.3.3 LSI . . . . . 83
5.3.4 ProdLDA . . . . . 90
5.3.5 BERTopic . . . . . 93
5.3.6 GEAC . . . . . 102

**6 Conclusion** 115
6.1 Future Implementations . . . . . 116

**Bibliography** 117

# List of Tables


5.1 LDA Results on ORIGIN+MESSAGE Spark dataset . . . . . 68
5.2 LDA Results on SPLIT Spark dataset . . . . . 70
5.3 LDA Results on MESSAGE Spark dataset . . . . . 71
5.4 LDA Results on ORIGIN+MESSAGE HDFS dataset . . . . . 73
5.5 LDA Results on SPLIT HDFS dataset . . . . . 73
5.6 LDA Results on MESSAGE HDFS dataset . . . . . 75
5.7 NMF Results on ORIGIN+MESSAGE Spark dataset . . . . . 76
5.8 NMF Results on SPLIT Spark dataset . . . . . 76
5.9 NMF Results on MESSAGE Spark dataset . . . . . 78
5.10 NMF Results on ORIGIN+MESSAGE HDFS dataset . . . . . 80
5.11 NMF Results on SPLIT HDFS dataset . . . . . 81
5.12 NMF Results on MESSAGE HDFS dataset . . . . . 82
5.13 LSI Results on ORIGIN+MESSAGE Spark dataset . . . . . 83
5.14 LSI Results on SPLIT Spark dataset . . . . . 84
5.15 LSI Results on MESSAGE Spark dataset . . . . . 85
5.16 LSI Results on ORIGIN+MESSAGE HDFS dataset . . . . . 87
5.17 LSI Results on SPLIT HDFS dataset . . . . . 88
5.18 LSI Results on MESSAGE HDFS dataset . . . . . 89
5.19 ProdLDA Results on all Spark datasets . . . . . 91
5.20 ProdLDA Results on ORIGIN+MESSAGE HDFS dataset . . . . . 92
5.21 BERTopic Results on ORIGIN+MESSAGE Spark dataset . . . . . 95
5.22 BERTopic Results on SPLIT Spark dataset . . . . . 97
5.23 BERTopic Results on MESSAGE Spark dataset . . . . . 97
5.24 BERTopic Results on ORIGIN+MESSAGE HDFS dataset . . . . . 99
5.25 BERTopic Results on SPLIT HDFS dataset . . . . . 100
5.26 BERTopic Results on MESSAGE HDFS dataset . . . . . 101
5.27 GEAC Results on ORIGIN+MESSAGE Spark dataset . . . . . 104
5.28 GEAC Results on SPLIT Spark dataset . . . . . 105
5.29 GEAC Results on MESSAGE Spark dataset . . . . . 107
5.30 GEAC Results on ORIGIN+MESSAGE HDFS dataset . . . . . 109

5.31 GEAC Results on SPLIT HDFS dataset . . . . . 111
5.32 GEAC Results on MESSAGE HDFS dataset . . . . . 113

# List of Figures


2.1 Example of an event created by HDFS . . . 3

3.1 Log formatting in case of errors . . . 11
3.2 Brackets Removal . . . 12
3.3 Paths Removal . . . 13
3.4 Words with Numbers inside Removal . . . 13
3.5 Values assignment Removal . . . 14
3.6 Single or Double quotes Removal . . . 14
3.7 Upper Case Split. Can be appreciated also the removal of special characters as described in the point before. . . . 15
3.8 Words that presents the Camel Case Capitalizion structure are divided into their inner words . . . 16
3.9 Words split on Camel Case or Upper case are allign with existing words made by their concatenation . . . 16
3.10 Representation of a generic sentence where each word is label with the corresponding PoS . . . 18
3.11 PoS Markov Chain representation . . . 18
3.12 PoS Markov Transaction Matrix . . . 19
3.13 Lemming Example . . . 19
3.14 Data Manipulation implementation - HDFS . . . 23
3.15 Data Manipulation implementation - Spark . . . 23
3.16 N-gram example . . . 24

4.1 LDA graphical model . . . 30
4.2 LSA Decomposition . . . 37
4.3 ANN representation . . . 39
4.4 VAE basic represetation . . . 41
4.5 RT basic representation . . . 43
4.6 Word2Vec Continous Bag of Word and Skip Gram architecture . . . 44
4.7 Attention mechanism representation . . . 45
4.8 Transformer Implementation . . . 46

4.9 Scale Dot-Product Attention vs Multi-Head Attention . . . . . 47
4.10 Complete Transformer Architecture . . . . . 47
4.11 Overall pre-training and fine-tuning procedures for BERT . . . . . 48
4.12 Twin Network architectures of Sentence-BERT. On the right, the architecture for the classification is represented, while on the right the one used at inference . . . . . 48
4.13 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. . . . . 52
4.14 Top2Vec UMAP-reduced document vectors . . . . . 55
4.15 Top2Vec dense areas of documents identified by HDBSCAN . . . . 55
4.16 The topic vector is the centroid of the dense are of documents identified by HDBSCAN, which are the purple points. The outliers identified by HDBSCAN are not used to calculate the centroid. . . . 56
4.17 The topic words are the nearest word vectors to the topic vector. . . 57

5.1 Overview over the unifying coherence framework - its four parts and their intermediate results . . . . . 63
5.2 Predictions LDA on Spark - ORIGIN+MESSAGE . . . . . 69
5.3 Predictions LDA on Spark - SPLIT . . . . . 70
5.4 Predictions LDA on Spark - MESSAGE . . . . . 71
5.5 Predictions LDA on HDFS - ORIGIN+MESSAGE . . . . . 72
5.6 Predictions LDA on HDFS - SPLIT . . . . . 74
5.7 Predictions LDA on HDFS - MESSAGE . . . . . 75
5.8 Predictions NMF on Spark - ORIGIN+MESSAGE . . . . . 77
5.9 Predictions NMF on Spark - SPLIT . . . . . 78
5.10 Predictions NMF on Spark - MESSAGE . . . . . 79
5.11 Predictions NMF on HDFS - ORIGIN+MESSAGE . . . . . 80
5.12 Predictions NMF on HDFS - SPLIT . . . . . 81
5.13 Predictions NMF on HDFS - MESSAGE . . . . . 82
5.14 Predictions LSI on Spark - ORIGIN+MESSAGE . . . . . 84
5.15 Predictions LSI on Spark - SPLIT . . . . . 85
5.16 Predictions LSI on Spark - MESSAGE . . . . . 86
5.17 Predictions LSI on HDFS - ORIGIN+MESSAGE . . . . . 87
5.18 Predictions LSI on HDFS - SPLIT . . . . . 88
5.19 Predictions LSI on HDFS - MESSAGE . . . . . 89
5.20 Predictions ProdLDA on Spark - SPLIT . . . . . 91
5.21 Predictions ProdLDA on HDFS - ORIGIN+MESSAGE . . . . . 92
5.22 Example of sentences assigned to existing topics, rather than creating new ones. . . . . 94
5.23 Predictions BERTopic on Spark - ORIGIN+MESSAGE . . . . . 95

5.24 Predictions BERTopic on Spark - SPLIT . . . . . 96
5.25 Predictions BERTopic on Spark - MESSAGE . . . . . 98
5.26 Predictions BERTopic on HDFS - ORIGIN+MESSAGE . . . . . 99
5.27 Predictions BERTopic on HDFS - SPLIT . . . . . 100
5.28 Predictions BERTopic on HDFS - MESSAGE . . . . . 101
5.29 Predictions GEAC on Spark - ORIGIN+MESSAGE . . . . . 103
5.30 Predictions GEAC on Spark - SPLIT . . . . . 106
5.31 Predictions GEAC on Spark - MESSAGE . . . . . 108
5.32 Predictions GEAC on HDFS - ORIGIN+MESSAGE . . . . . 110
5.33 Predictions GEAC on HDFS - SPLIT . . . . . 112
5.34 Predictions GEAC on HDFS - MESSAGE . . . . . 114

# Acronyms

**AI**
Artificial Intelligence

**TC**
Topic Cluster

**LDA**
Latent Dirichlet Allocations

**NMF**
Non-Negative Matrix Factorization

**LSA**
Latent Semantic Analysis

**HDFS**
Hadoop Distributed File System

**NLP**
Natural Language Processing

**SGD**
Stochastic Gradient Descent

**VAE**
Variational AutoEncoder

**AE**
AutoEncoder

**BERT**

Bidirectional Encoder Representations from Transformers

**W2V**

Word2Vec

**D2V**

Doc2Vec

# Chapter 1

# Introduction

Every day in our lives and society, an incredible quantity of data is produced. Being able to analyse and gather useful information from this huge amount of collected data is a very demanding task, because both experts and computational cutting-edge technologies are needed to approach the issue.

Data mining is the process of extracting high-quality information from row datasets and discovering implicit and hidden knowledge from data. This is done by analysing the structures of the datasets to find relevant patterns between the data items, and this involves methods leaning on machine learning, statistics and database systems.

Over the years, data mining has become increasingly important due to the constant generation of large amounts of data. This higher production of data applies also to textual data. From the social networks to digital libraries, from the e-learning platforms to technical-sources, everything today is based on a e-content and, as consequence, the amount of text documents that is daily produced grows continuously.

After collecting data, data mining techniques are applied to textual data, aiming to extract useful information: more in detail, these techniques belongs to text mining, a subsection of data mining that aims to gather useful information from texts. Since textual data are spread among a wide range of applications and among different type of structures, text mining is one of the most challenging activities in the data mining field.

Due to these reasons, text mining techniques, including topic modelling, are currently areas of great interest both for the scientific community and many big tech companies. Indeed, being able to process correctly the data gathered is crucial for many major tech companies, since several services depends on those data. However, text analysing is complex and resource intensive. This occurs because a relevant chunk of data is produced by people, and since the human language differs from person to person and from language to language. Thanks to natural language

processing and analytical methods we can mine texts. These techniques aim to create models and structures from textual sources by means of syntax and semantic analysis, allowing data analysis and investigation.

In the scientific research, different models have been proposed to represent and to retrieve information coming from the textual data. Depending on the techniques, different approached in the scientific literature exist: algebraic models, that represent texts as document-matrices, models relying on probability and the latest models based Deep Learning approaches, in particular on Transformers and AutoEncoders.

Moreover, making this techniques automatic is a challenging task due to the huge production of data that takes place nowadays. Indeed, using data processing techniques would be impossible if human supervision were needed. This is applied also to text mining since every involved process requires specific configurations and parameters. To overcome this problem, innovative technologies and approaches to make text analysis and mining automatic have already been proposed in scientific field, but many additional efforts are needed to improve and to make them more effective.

The goal of this thesis is to implement and to study a framework able to process unstructured systems logs and then, based on the features extracted, to cluster logs documents into cohesive and well separated groups, based on the content of the data.

The framework should be able to be independent, as much as possible, from the work of the analyst, reducing the interaction where needed. In detail, neural language processing techniques will be used to retrieve information from the two data collections, while probabilistic, algebraic, and neural network topic models, combined with a similarity analysis to determine the optimal number of topics, will be used to cluster data.

# Chapter 2

# Application Domain

## 2.1 What are System Logs

In computer science, logs are made up of unstructured text printed by instructions software. The types of events logged by a system can arise from interactions of a user, by internal conditions relating to the execution of the program, by communications received from other systems and much more. The information contained by a message of logs vary from system to system; there are some features, however, that a message almost always contains. The log message, taken as example and represented in Figure 2.1, records an event of specific system and it is characterized by a series of fields: the timestamp (the instant when the event occurred, 2021-02-09 20:46:55), the verbosity level (the level of gravity of the event, eg. INFO), the origin that launched the message and the description of the event in textual form.

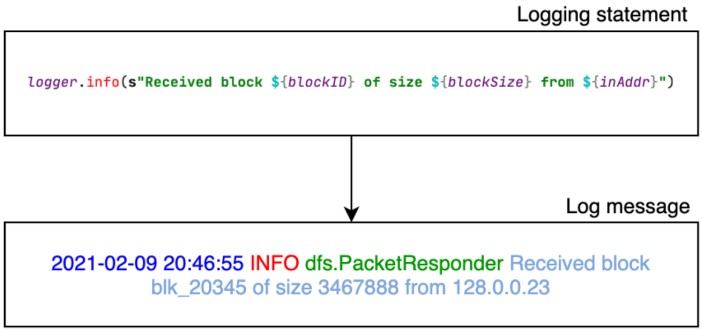



**Figure 2.1:** Example of an event created by HDFS

As regards the timestamp, although it is important to determine the order of events, it is uncertain this reflects the exact instant when the event happened,

especially in modern distributed systems. Usually, the description in textual form is made up of two types of information: a constant textual part that describes the action performed and a variable part that records the value of some parameters. A common practice to facilitate analysis phase is reducing each log message to the event, which corresponds to the fixed part. System logs have always been the best source for checking the status of systems. There are several factors that could be monitored: some hardware components can break, some applications can result in unwanted states, some components may be misconfigured. A science called *log analysis* deals with their interpretation and management.

## 2.2 What is Log Analysis

Log analysis [1] is the process of reviewing, interpreting and understanding records generated by the computer, called logs. The log files are generated by a range of technologies, including network devices, operating systems, applications and more. Thanks to the advent of the Artificial Intelligence and with the advancement of Machine Learning techniques, the extraction of useful information from logs is an increasingly asset for end users. The most frequently methodologies used are summarized below to extract value from logs:

- Collection, centralization and indexing: through installation of a collector, all the logs coming from the different sources that compose the stack are collected in one place and indexed for ease of research and analysis process;
- Monitoring and warning: through Machine Learning techniques, it is possible to implement real-time monitoring techniques aimed at generating alert under certain conditions;
- Dashboard: Real-time reports and custom dashboards work great visual indicators for log controllers.

## 2.3 Natural Language Processing

Natural language processing [2] (NLP) is a sub-field of linguistics, computer science, and artificial intelligence dealing with the interactions between computers and human language, focusing on programming computers to process and analyse natural language data. The goal is to be able to understand the contents of documents, considering the different aspects of human languages. These processes are able to acquire information from documents, in order to categorize documents themselves. Different research fields exist in NLP, including speech recognition, natural language understanding and generation.

In the first steps in NLP, many processing systems were designed by handmade symbolic methods, like coding rules with dictionary lookup. Nowadays, machine-learning systems have many advantages over hand-produced rules. They learn automatically by focusing on the common cases, reducing the time needed by hand written rules, since it is not immediate where the effort should be directed. This approach also reduces errors that can occur when the hand-written rules are too specific, making machine-learning models more robust, and making easier the managing of huge quantity of data as happens these days.

Although the common use of machine learning in NLP, classical methods, like symbolical one, are still commonly used:

- When the amount of training data is insufficient to apply machine learning methods;
- For pre-processing in NLP pipelines, e.g., tokenization;
- For post-processing and transforming the output of NLP pipelines;

### 2.3.1 Statistical Methods

NLP research has relied heavily on the machine-learning paradigm, that uses statistical inference to automatically learn rules through the analysis of large set of documents. Different classes of machine-learning algorithms, which take as input the features extracted from input data, have been used in NLP. Many researches have been led on statistical models, which weight each input feature thanks to soft probabilistic decisions. These models allow to express the relative certainty of many different possible outputs, providing more reliable results when the model is placed inside a larger system. At the beginning, many machine learning algorithms were based on the production of the same if-then set of rules making system, while other algorithms, like hidden Markov models for part-of-speech tagging, were introduced later. Nowadays many speech recognition systems relies on "cache language models", an example of such statistical models, that are generally more robust when given real-world data. After the introduction of Neural Network, statistical methods started to be replaced, however, they continue to be relevant for contexts in which statistical interpretability and transparency are required.

### 2.3.2 Neural Networks

As described above, a drawback of statistical methods is the need of features engineering. In the latest years, the research field has moved from statistical methods towards neural networks for machine learning. Word embeddings is a popular example, it tries to capture semantic properties of words, allowing an

end-to-end learning of a higher-level task rather than relying on a pipeline of different intermediate tasks. In some fields, the introduction of Neural Network changed the design of NLP systems, making this new approaches viewed as a new distinct paradigm.

## 2.4 What is Topic Modeling

In NLP, Topic Modeling [3] indicates a typology of statistical model used to retrieve the topics from textual data. This is known as 'unsupervised' machine learning because it does not require a predefined list of tags or training data that has been previously classified by humans. Topic modelling is a text-mining tool used to discover hidden semantic structures inside textual data. To know by intuition, a document is about a particular topic, anybody would expect particular words to appear in the document more or less frequently: "popcorn" and "movie" will appear more often in documents about cinema, "car" and "bike" will appear in documents about vehicles, "the" and "have" will appear approximately equally in both. Typically a document concerns multiple topics in different proportions; thus, in a document regarding drive-in theater is 10% about vehicles and 90% about cinema, probably there would be about 9 times more cinema words than vehicle words. The output produced by Topic models are clusters of similar words, called topics. A topic model captures mathematically the above intuition by examining a set of documents and, from the statistics found, it figures out what the topics might be, with the relative balance inside documents. For this reason, Topic models are also referred to probabilistic topic models, since statistical algorithms are used to discover the latent semantic structures from texts. At the same time, this view is starting to be overtaken by neural network approaches, where many researchers try to combine probabilistic knowledge with already existing AI models. Topic models implementation is straightforward in the age of information. Due to the huge amount of the written material, they can help to organize and provide information to understand large collections of unstructured text bodies.

## 2.5 Dataset - LogHub

The dataset used in the case study is provided by LogPAI and it is included in LogHub dataset repository collection [4]. This collection contains a set of log datasets produced by various different systems (supercomputers, distributed systems, systems operational etc.). The choice of this dataset over the others is guided by its wide use in literature, both from researchers and companies, and overall it provides logs coming from different sources with a huge variety of contents.

Among the different categories, in this project the focus is on data coming from

"Distributed Systems". In detail, the dataset of HDFS and Spark are taken into consideration.

The logs examined coming from HDFS [5] are produced by a Hadoop cluster consisting of approximately 200 EC2 (Elastic Compute Cloud - Amazon). Apache Hadoop software is an open-source framework that enables distributed storage and processing of large, clustered datasets of computers using simple programming templates. Hadoop is designed to scale up from a single computer to thousands of clustered computers, where each machine provides local computation and storage. This allows Hadoop to efficiently store and process large data sets that they range from gigabytes to petabytes. The Hadoop Distributed File System (in acronym HDFS) is a distributed file system, portable and scalable written in Java for the Hadoop framework. A cluster in Hadoop typically owns one or more name nodes (on which the files metadata reside) and a set of data nodes (on which the files HDFS).

Instead, the Spark logs come from Apache Spark [6], a unified analytics engine for big data processing, with built-in modules for streaming, SQL, machine learning and graph processing. It is widely used to unify the processing of data in batches and real-time streaming, using different languages like Python, SQL, Scala, Java or R. Moreover, it allows to execute fast, distributed ANSI SQL queries for dashboarding and ad-hoc reporting and to perform Exploratory Data Analysis (EDA) on petabyte-scale data without having to resort to downsampling. Finally, in machine learning it is appreciated since allows to scale machine learning algorithms scale to fault-tolerant clusters of thousands of machines. Spark Core [7] is the foundation of the Spark project. It provides distributed task dispatching, scheduling, and basic I/O functionalities. These functionalities are exposed through an application programming interface centered on the RDD abstraction, which mirrors a functional/higher-order model of programming: a "driver" program invokes parallel operations on an RDD by passing a function to Spark, which then schedules the function execution in parallel on the cluster. Currently, thanks to the aforementioned pros, Spark has been widely deployed in industry. In this case, the Spark log dataset was collected by aggregating logs from the Spark system in LogPai lab at Chinese University of Hong Kong, which comprises a total of 32 machines. The logs are aggregated at the machine level, they have a huge size (over 2GB) and they are provided as-is without further modification or labelling, which involve both normal and abnormal application runs.

# Chapter 3

# Textual Data Analysis

As briefly seen with the description of the structure of logs, it is crucial to understand how to properly extract meaningful words from texts plenty of irrelevant elements. To approach this task, I base my analysis on the main NLP techniques combined with some methods created ad-hoc for this type of data. A general preamble must be done to underline that normally this process is really human dependant, making the results subjective, and, as consequence, a supervision must be provided. However, my goal is to reduce as much as possible the human intervention, for this reason the whole data manipulation process has been developed as an almost black box. It takes in input the logs, then the judgement of an expert in NLP is asked where needed, who should provide some feedback about the best choices to implement, and finally the output is saved inside a folder. In any way, to have a better comprehension about what the algorithm does inside, intermediary files are saved, in order to have the comparison between input and output.

The starting point inside this process is to understand how to manage the different parts that make a log record. As described in Chapter 2, the structure of logs is in many cases similar between systems, indeed we find always timestamp, verbosity level, origin and message. The first two can be considered irrelevant to understand the topic since they do not bring any useful information. Different reasonings must be done on the origin and on the message. If the message is essential to understand the content of the record, instead, some considerations on the origin are needed about how and if it is needed. There are different possibilities about how to manage it. Indeed, as first possibility the origin can be discarded, in this way all the contents are collapsed together without making any distinction. So, if different records have the same message but it comes from different origins, we loose any trace of this differentiation. On the other hand, if I keep the origins, the same message will be considered different.

An important source that helps to figure out which direction is better can be found inside the anomaly detection phase. Many algorithms, considered today as

the state of the art for log anomaly detection like DeepLog[8] and LogAnomaly[9], state that their goal is to separate log entries for different tasks in a log file. Following this reasoning the origin can be important to understand the workflow that has produced those key sequences. After considering the different possible cases, I decide to proceed in any way with both the approaches. This conclusion is based on the idea that best approach can be appreciated only after the topic modeling and anomaly detection phases, with their corresponding results.

## 3.1 Data Cleaning

In this first part all the operations regard the deletion of irrelevant elements and about the reduction and the alignment of the different tokens that otherwise would be considered as different objects inside topic models. As described within the introduction to NLP in Chapter 2, the overall goal to process data is to find the best way to transform and to retrieve information from data without making the whole process too related to the data analysed. For this reason, I try to capture the most relevant pattern that are frequent inside logs. To implement it, I opt for many techniques and libraries today considered as the state of the art when dealing with text mining. Since the difficulties in managing this types of dataset, although in the latest year some text mining tools based on neural network have been created, I decide to proceed with more common and consolidated techniques, since this is the first time I deal with this type of data and of structures. In this context, the ground truth of the task is the human supervisor, making necessary to understand each step and the best way to handle the main situations.

In the following list I present the main points implemented to retrieve useful information:

- Reformat Log Errors;
- Remove Accented Characters;
- Remove Special Characters and Patterns;
- Camel Case Split;
- Merge Came Case repetitions;
- Expand Contractions;
- Tokenization;
- StopWords Removal;
- PoS Tagging;

- Lemming;
- Regex;
- Plural to Singular;

Now, I am going to explain more deeply all the processing done inside each of them.

### 3.1.1 Reformat Log Errors

When an error occurs in the system, the file log records the message error, in which part of the code it occurs and for which reason. Many times the part of the code in interest is placed inside a class that is called in cascade by many others. For this reason, when the whole path is printed, this output can be really long and it can be splitted in multiple lines. Indeed, to have a better comprehension about the issues, inside each class programmers print a message linked to the error and then let the error to walk back the call stack until it is caught and handled. So there may be multiple prints with very long messages and it is important to understand how to properly manage them. Many considerations can be done but, in general, only after a check on the topic clusters we can understand which is the best way to manage them. Starting from this, different trials have been done, by cutting all the nested messages, by truncating part of them or by collapsing all inside a single row. After some results, the best approach results in cutting them and keeping only the first row. This can be motivated by considering that, even if there may be words that are relevant with respect to the entire corpus, these words are not linked with the topic of error and so they let the topic model to move towards wrong decisions.

```
java.io.EOFException: End of File Exception between local host is: "mesos-slave-32/127.0.1.1"; destination host is: "mesos-master-1":9000;
     at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
     at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:57)
     at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45)
     at java.lang.reflect.Constructor.newInstance(Constructor.java:526)
     at org.apache.hadoop.net.NetUtils.wrapWithMessage(NetUtils.java:792)
     at org.apache.hadoop.net.NetUtils.wrapException(NetUtils.java:765)
     at org.apache.hadoop.ipc.Client.call(Client.java:1480)
     at org.apache.hadoop.ipc.Client.call(Client.java:1407)
     at org.apache.hadoop.ipc.ProtobufRpcEngine$Invoker.invoke(ProtobufRpcEngine.java:229)
     at com.sun.proxy.$Proxy13.sendHeartbeat(Unknown Source)
     at org.apache.hadoop.hdfs.protocolPB.DatanodeProtocolClientSideTranslatorPB.sendHeartbeat(DatanodeProtocolClientTranslatorPB.java:153)
     at org.apache.hadoop.hdfs.server.datanode.BPServiceActor.sendHeartBeat(BPServiceActor.java:553)
     at org.apache.hadoop.hdfs.server.datanode.BPServiceActor.offerService(BPServiceActor.java:653)
     at org.apache.hadoop.hdfs.server.datanode.BPServiceActor.run(BPServiceActor.java:823)
     at java.lang.Thread.run(Thread.java:745)
```

**Figure 3.1:** Log formatting in case of errors

### 3.1.2 Remove Accented Characters

In this step the processing of texts begins, where I try to make textual data uniform by removing errors and differences. In this point, the process is about

removing possible accented characters. In both spoken and written languages, accented characters are used to emphasize a particular word during pronunciation or understanding and, in some cases, they clarify the meaning of a word, which might be different without the accent. Accents might be also inserted because of someone keyboard default setting or typing style. While their are almost not use in English, it is very common to face accents in a text corpus, in particular in other languages. Hence, I need to make sure that these characters are converted and standardized into ASCII characters. A simple example is the conversion of é into e.

### 3.1.3 Remove Special Characters and Patterns

In almost any texts we can find special characters, described as non-alphanumeric characters, in comments, references, currency numbers etc. These elements add no value to text-understanding and they induce noise into algorithms. Thankfully, regular-expressions can be used to get rid of these characters and numbers.

Other than special characters, inside logs frequent structures are present, that do not bring any useful information. The search of this structure is heavily human dependent since, without adequate functions, classical textual analysis would bring to a data processing where the words inside those structures are kept.

The choice to identify and to delete them comes from several and different analysis. I have tried also to keep the words inside them, but non high quality topic cluster would have obtained, since some structures, although irrelevant, are quite frequent inside the different log entries.

This is the pipeline created to achieve this purpose:

- First of all, all the brackets are deleted with their content since usually inside brackets people put marginal words or proper noun with or without values assignment. So, these elements are discarded. This operation can be appreciated in Figure 3.2.

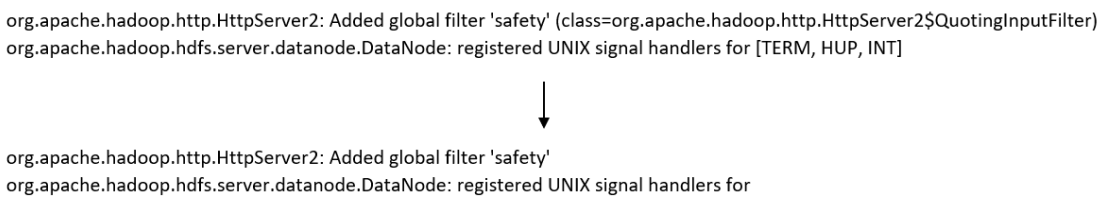



**Figure 3.2:** Brackets Removal

- In system logs name of folders, directories, hyperlinks, websites and many specific structures, that can be generalized as paths, are quite frequent. These structures are quite irrelevant since they do not bring any information about

what is going on related to that log key, but rather they would make that record too specific due to particular names there located.
Although those names could be considered quite rare, this is not true for log files that are focused on some operations related to specific paths, making them quite frequent and causing a poor generalization of the topic models. Indeed the models would collapse those log keys together for those common and shared elements, while the different actions are not considered.
Following this thoughts and after some the trials, I come to the consequence to delete them.

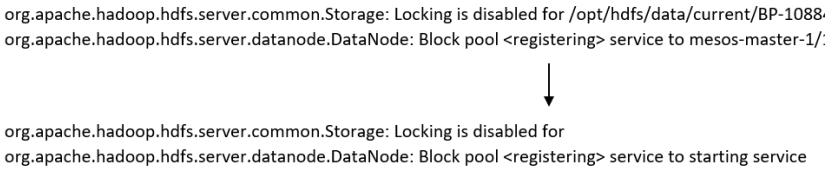



**Figure 3.3:** Paths Removal

- Then the removal of words that contains words is applied. These words are in general keys or specific names used internally in the system. Many times this pattern is connected to the assign of a value to a parameter with the following structures: *words=/:numbers*. In general these words can be considered too specific and so not relevant.

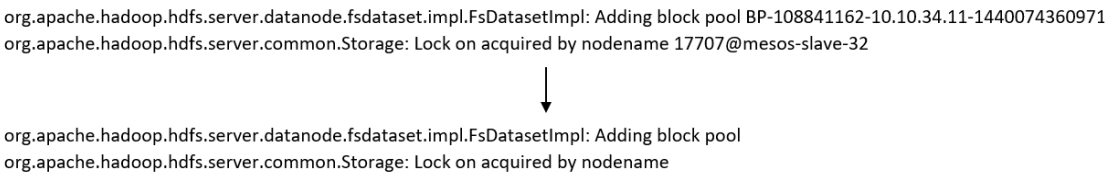



**Figure 3.4:** Words with Numbers inside Removal

- Following the previous point, the assign of values can happen also with string and not only with numbers. The structure is now *words=/:word* and it is discarded for the same aforementioned reason.

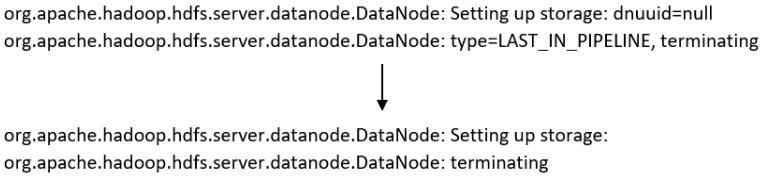



**Figure 3.5:** Values assignment Removal

- Sometime, some words or group of words are put inside quotes or double quotes. Differently from the reasoning on brackets, using quotes is more common to underline rather than make less relevant annotations. For this reason, if this structure is present, the quotes or double quotes are removed. It is important to mention that this is done by avoiding single quotes linked to contractions, that otherwise would be discarded. An example is shown in figure 3.6.


**Figure 3.6:** Single or Double quotes Removal

- Then there comes the removal of special characters. Only characters inside the alphabet plus the single quote are kept, since linked to contractions. So, special elements like punctuation, except apostrophes, are deleted.

- There are some cases, linked with log entries related to errors, where some words are made as the concatenation of two different words: one of the two is all written in upper case. If I would consider this case as single token, its frequency would be really low, but if I would examine as two separated words the result would be different. Indeed, I can extract words that are shared and that can help to group and cluster related words.

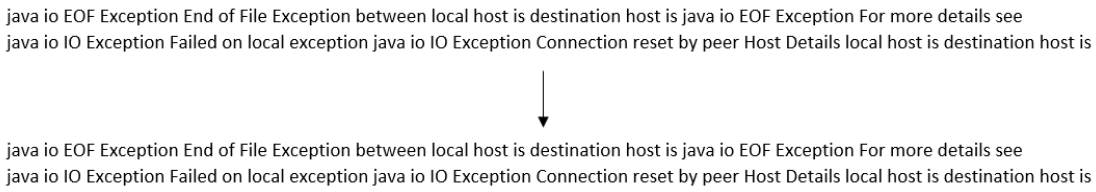



**Figure 3.7:** Upper Case Split. Can be appreciated also the removal of special characters as described in the point before.

All of these analysis are largely done with regular expressions, since the individuation of this structures with other tool would be more difficult to obtain. I have to say that these steps are optional, so if any of them is not necessary, anyone can be easily avoided by specifying the one desired at the beginning. Lastly, there may be other patterns in other logs or datasets, depending on the type of system they coming from.
Thanks to the modular approach of each function, a possibilities is offered to the domain experts that are going to use this tool. Additional regex can be added at the beginning of the program, without the necessity to know and see the other functions.

### 3.1.4 Camel Case Split

As anticipated in the introduction, today many infrastructures run on Object based systems. Hence, many classes and methods use the approach called Camel Case Capitalization. It is the practice of writing phrases without spaces or punctuation, indicating the separation of words with a single capitalized letter, and the first word starting with either case depending if it is a class (capital), or a method (not capital). To avoid keeping this long sequences of words as a single token, I decide to split them using this structures since inner words can also be frequent as single elements. Other than this processing, in this phase all the words are also put to their correspondence lower case.

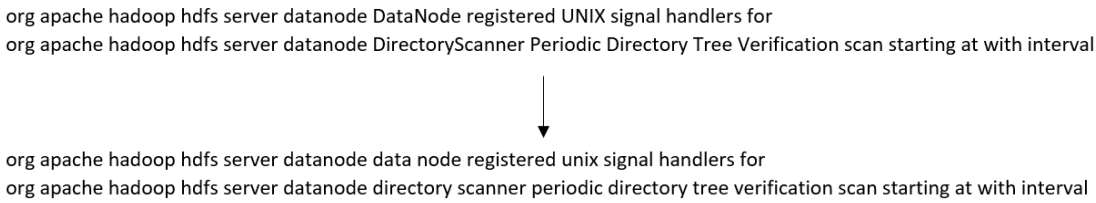



**Figure 3.8:** Words that presents the Camel Case Capitalizion structure are divided into their inner words

### 3.1.5 Merge Improper words divisions

Although the Camel Case and the upper case split implemented are a really powerful tool for this type of dataset, with particular utility to analyze better the origin of the message where are common those structures. Sometimes the split words, if merged, make another word inside the general corpus of the file. In context where it is important to make uniform the words inside the corpus, since many further analysis are based on frequencies and probabilities, it is relevant to collapse those words as a singe element, made by their merging.

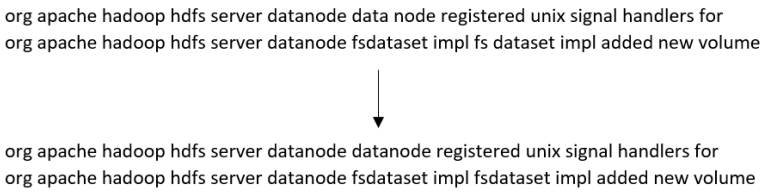



**Figure 3.9:** Words split on Camel Case or Upper case are allign with existing words made by their concatenation

### 3.1.6 Expand Contractions

Contractions are shortened versions of words or syllables. They are created by removing one or more characters from words. In writing, an apostrophe is used to indicate the place of missing letters and in case of English language, contractions often exist in both written or spoken forms. Nowadays, contractions are used by default in many context, including professional and educational ones. Common examples are *do not* to *don't*, *I would* to *I'd*, *you are* to *you're*. Converting each contraction to its original form helps with text standardization, an important step to delete possible differences [10]. For the seek of this project, since the simpler form of the texts, the removal of contractions is done by leveraging a standard set of contractions available in the contractions library [11].

### 3.1.7 Tokenization

Tokenization is a common task in Natural Language Processing. A tokenizer splits unstructured data and natural language text into chunks of information, which can be single words o groups of words and they are considered as discrete elements. The token occurrences in a document can be used directly as a vector representing that document [12]. On the basis of this operation many other NLP processig methods take place. Indeed, many of them require to manipulate elements in order to focus into a small parts rather than entire sentences.

### 3.1.8 StopWords Removal

Among the several tokens produced in the previous step, there are words that needs to be filtered out before processing a natural language and they are called stop words. Actually these are the most common words in any language (like articles, prepositions, pronouns, conjunctions, etc) since added to sentences to make them grammatically correct and they do not provide much information to the text. Common examples of English stop words are “the”, “a”, “an”, “so”, “what” and so on.

Stop words are available in abundance in any human language. By removing these words, we remove the low-level information from text in order to give more focus to the important elements. In other words, the removal of such words does not cause any negative consequences on the models, but it rather helps to avoid poor performances caused if they would be skewed upon those words. This operation is also important since it reduces the dataset size and, as direct consequence, it reduces the training time due to the fewer number of tokens involved in the training.

Inside the project, the list of stop words coming from nltk.corpus [13] is used and this list can further be enhanced by adding or removing custom words based on the situation at hand. In particular, in my case all the stop words related to negative sentiments are kept and similar ones like "not", "nor" or "no" are conduct to the same token in order to avoid many rare tokens rather that one more frequent. In this way, ideally, topics related to negative sentiments can be created.

### 3.1.9 PoS Tagging

Part-of-speech (POS) tagging is a natural language process which aims to categorize words in a text in correspondence with a particular part of speech, taking into consideration the definition of the word and its context [14]. Indeed, this process can not be summarized as a list of words and with the corresponding part-of-speech since many words can represent more than one part-of-speech at different times.

In the figure above, each word has its own lexical term written underneath, however, if all these full terms should be written every time, text analysis quickly

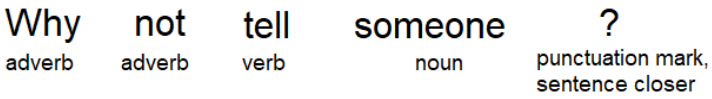



**Figure 3.10:** Representation of a generic sentence where each word is label with the corresponding PoS

become unfeasible, especially with many data. Therefore, the representation of the categories is done by the mean of a short representation referred as "tags". As earlier mentioned, this process of assigning a specific tag to a word is referred to as part-of-speech tagging (PoS tagging). They describe the characteristic structure of lexical terms within a sentence or text and they are used to assumptions about semantics. Many times these informations are used as essential part in text processing, as we will see later.

To assign these tags, some computation are needed, in particular the Markov Chains are the main structure behind. The principal idea, as depicted below, is that the POS tag, that is assigned to the next word, is dependent on the POS tag of the previous word, recurring principle of Markov models.

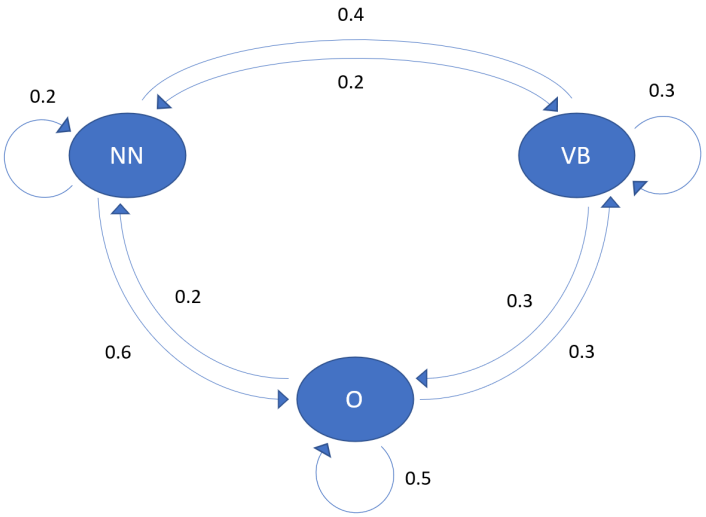



**Figure 3.11:** PoS Markov Chain representation

In this way it is built a direct graph with all the possible paths and the linked transition matrix is obtained, where each cell express the probability to move between the chosen node pair, that inside the graph is represented by a weighted edge. These probabilities come from count occurrences of tag pairs in the training

dataset and can be formulated as:

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}) + \epsilon}{\sum_{j=1}^{J} C(t_{i-1}, t_j) + N * \epsilon} \tag{3.1}$$

| | NN | VB | O |
|---|---|---|---|
| (initial) | 0.4 | 0.1 | 0.5 |
| NN (noun) | 0.2 | 0.2 | 0.6 |
| VB (verb) | 0.4 | 0.3 | 0.3 |
| O (other) | 0.2 | 0.3 | 0.5 |

**Figure 3.12:** PoS Markov Transaction Matrix

### 3.1.10 Lemming

In Spoken and written languages several words are often derived from others. When it happens, as in spoken language, it is called Inflected Language [15]. In grammar, inflection is a process that consists on modifying a word to express different grammatical categories such as tense, case, voice, aspect, person, number, gender, mood, animacy and definiteness. An inflection can express differet grammatical categories with a prefix, suffix or infix, or another internal modification such as a vowel change. As stated by this explanation, it is understandable that inflected words can be conducted to a common root form. Following few examples.

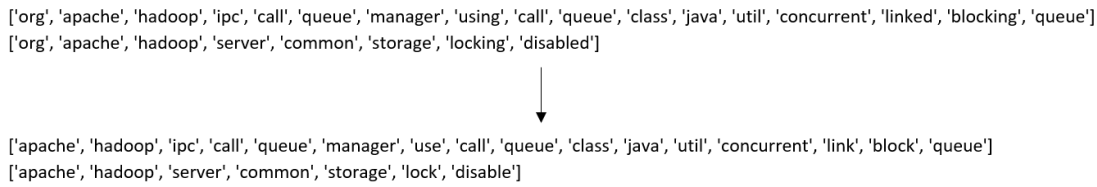



**Figure 3.13:** Lemming Example

In this context, Stemming is the process of reducing inflected words to their root forms by leading a group of words to the same root, i.e the stem, even if

the root does not exist in the Language [16]. The stem is the part of the word where the inflectional affixes is added. So, stemming a word may results in a not-existing word, since they are created by truncating the suffixes or prefixes used. Unlike Stemming, Lemmatization is the linguistic process of grouping together the inflected words such that they can be analysed as a single item, identified by the words root. In Lemmatization root word is called Lemma, a canonical and dictionary form of a set of words belonging to the Language. In order to use a lemmatizer, it is necessary to provide the context where you want to lemmatize by means of POS tags.

Other than this differentiation, it is important to understand which is the best implementation, depending on the situation. Indeed, Stemming is preferable if speed is a requisite but, if the task is strictly focused on language like topic modeling, Lemmatization is preferable since it brings a word to its corresponding lemma rather than a common root hardly to understand. So, several thoughts and trials have been done with both the approaches. My conclusion is that Stemming produces less relevant root, in particular it generalizes too much, deleting words that in reality are different. For this reason, I choose the Lemmization, but some consideration are needed about how it works and how to manage specific cases.

The Lemmatizer used for this project is WordNet [17], a large, freely and publicly available lexical database for the English language, aiming to establish structured semantic relationships between words. It offers lemmatization capabilities as well and it is one of the most commonly used lemmatizers. NLTK [13] offers an interface to it, as for many other approaches. Although, WordNet [17] is one of the best lemmatizer available today, it has several shortcomings for words that are not that common. Indeed, with the two dataset analysed, the words are mainly technical and tech related and, as consequence, the lemmatizer is not able to always produce correct lemmas.

Another issue is leading words to their lemma by using the PoS tags. They hugely influence the lemmatizer and in some situations the proposed lemma is not in line with the human expectations.

From this two issues I try to propose a possible solution. First of all, I decide to process only a subset of PoS tags, in particular the ones related to nouns, verbs and adjectives. All the others are discarded since they are not considered linked to words that provide a significant meaning. Before implementing it, a quality control is performed. As anticipate, sometimes the tags assigned are not correct, so I decide to make interaction with the supervisor to confirm or correct some tags assigned. This could be an expensive process with common type of texts, but, since inside log files many times there are the same entries with just few modifications, the total number of words are feasible to manage. To be even more focused and to restrict the subset of possible wrong tags to check, only the words with a tag related to adverbs are taken into consideration. Indeed, from my studies, this is

the most typical tag error committed in PoS phase. In any case this analysis can be extended to all tags.

Then follows the correction of some tags related to word deriving from verbs. Indeed, depending on the placement inside the sentence, verbs at past participle or ending with *–ing* are considered as adjectives or nouns. In general, this tags are correct but, for my purpose, the differentiation coming from the different lemma is unnecessary for the seek of this task. Indeed, without generalizing with the infinite form of the verb they come from, I would obtain many different words with the same meaning.

Many times it happens the same word has multiple tags and it could cause to obtain different lemmas. So, also with another interaction with a supervisor, we ask to keep only one tag, correcting it or keeping the different tags as been found automatically. After all the corrections on the tags, the lemmization phase arrives. It is applied on all verbs, while instead for nouns and adjectives only the words with a length larger than three characters are kept and lemmatized, since the shorter ones are in general eventual less common stopwords or words not relevant in the context of these datasets. In any case, the supervisor can pass a list of words that must be taken, independently from the analysis done.

Among all this processing, I have to keep in mind that both the Pos Tagger and the Lemmatizer are not perfect, so in some cases the results are completely wrong. For this reason at this point the raw log keys are saved with list of stemmed tokens to understand if the level of quality is good or further considerations are needed. Someone could argue it results in comparing thousands of pairs, but this is not true since, before the PoS Tagging phase, only the unique rows are taken. This step is essential since it reduces the size of the file for over the 99% due to the general composition of log files, where there are the same row but with only constants or specific names that differs. For example the analysed HDFS file goes from almost 200k to 99 unique rows, while the Spark file from over 400k to 45. So, the comparison is way more feasible with fewer rows. With this comparison it is possible to see eventual errors and, to let the algorithm avoid those ones, the supervisor can pass a list of words to keep and a dictionary to use as personal lemmatizer to change words that WordNet [17] is not able to bring to their simpler form.

### 3.1.11 Regex

At the end of the processing, some dirty words are still present as trace of the operations done above. For this reason, I delete these words with regexes in order to capture patterns that do not correspond to any real word. These patterns are made by the concatenation of three or more consonants or vowels followed by other consonants or vowels depending on the case.

### 3.1.12 Plural to Singular

After all the steps done inside the data processing pipeline, there may be some words that are still different depending on their form, if singular or plural. Mainly it happens because of the PoS Tagging and Lemmization phases, that are not capable to analyze correctly all the words. I can find this behaviour in words ending with *-s* and unusual inside the common language, even though specific inside a particular domain. So, the PoS Tagging and Lemmization are not able to properly identify them and so they are not conduced to their root. For this reason and to still minimize the different tokens, if there are words that are different for this behaviour, they are conducted all to their singular form.

## 3.2 Data Manipulation

After the cleaning of data, the extraction of the relevant words is implemented. In this intermediate step, words that are not relevant to distinguish topic are discarded, therefore all the ones whose frequency among documents it too high. Following, the words that are averagely shared are kept only at their original form, i.e the monograms, and, finally, the variations by creating the n-gram are produced for the remaining. Hence, the first analysis is to find the words that can be considered as stop words for the dataset, i.e the words that are so common that they do not bring any information [18, 19, 20], and they are quite frequent inside many rows. The technique is based on the frequency of the words in documents, it is applied a CountVectorizer on a corpus where each line is made by unique words. In this way it's possible to appreciate in how many documents that word can be found. In general, the words that have a really high or really low frequency are considered as stop words, for the reason presented above. Following this principle, I decide to discard the words whose frequency is over the 80% of the documents, since they are not useful to distinguish topics but instead they could be noise for algorithms. Instead, I come to the conclusion to keep the less frequent ones due to the small size of the dataset and since they could be related to a rare document and as consequence to a rare topic. Many other words, instead, have a frequency in the middle are not considered as stop words. After several trial and analysis coming directly from the results of the topic models, those words can still cause some issues with the creation of the topic clusters, creating topics too similar that generalize too much. For this reason those words having a frequency below 80% but higher than 20% are kept only in their variation. It must be considered that, also in this situation, the end user can specify the desired thresholds. Since in many cases it is not easy to know in advantage which are the best values, a plot is printed with the frequencies in order to make easier the choice.

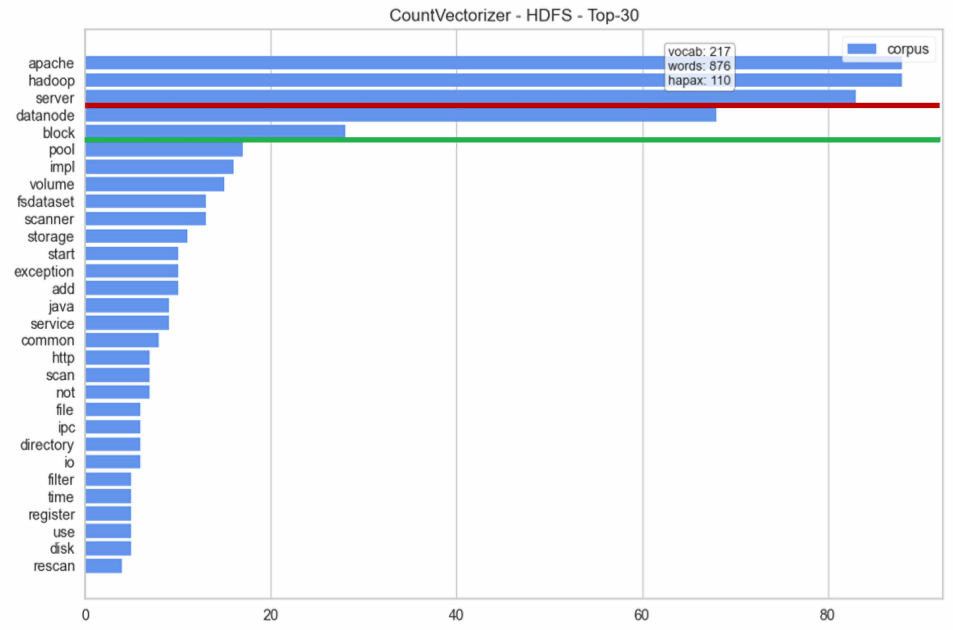



**Figure 3.14:** Data Manipulation implementation - HDFS

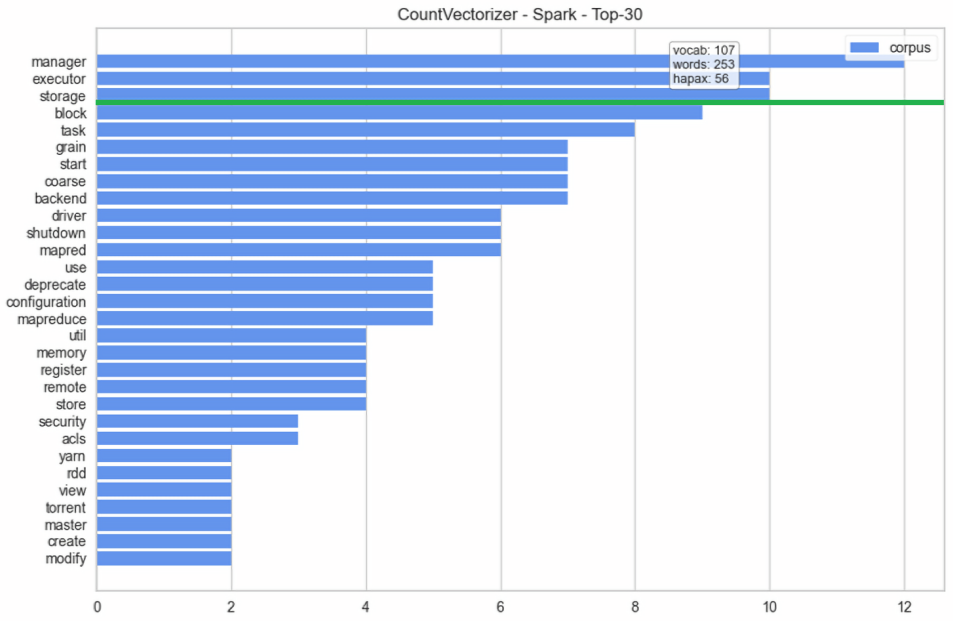



**Figure 3.15:** Data Manipulation implementation - Spark

## 3.3 Feature Creation

The variation implemented to extend the tokens for each row is one of the most known in literature, it is the creation of the n-grams for the tokens of each row, by taking into consideration also the application of the aforementioned frequency rules. In this function, other than keeping the monograms, the bi-grams and the tri-grams of the tokens are created. Longer concatenations are possible but it is common to avoid them, since they would take too many words, creating specific tokens, whose meaning could not be high due to mixed unrelated parts of the sentence. Then, some important rules are applied. With logs, particularly when the origin is considered, some bi-grams could be made as the pair A-B (where A and B are two tokens/words) and then other pairs as B-A are created , i.e the mirror of A-B. This causes a differentiation between them, even if the meaning of the two bi-grams is the same under the human prospective. For this reason, in this situation, the B-A bi-gram are replaced with the A-B form, considered as the ground truth when this check is performed. A similar thought could be done on tri-grams, however, since the meaning can change depending on the combinations of three words and since tri-grams are rarer than bi-grams, I decide to keep them as they are, without particular modifications Another possibility is that both bi-gram and tri-grams have more than one time the same word (so for the bi-gram is made by two times the same words). These ones, taken in isolation, are not relevant since they do not transmit a meaningful message. From this thought, they are not taken into consideration. Overall, the addition of n-gram is crucial for this task since, given the shortage of words in each row due to the nature of data, the expansion of the corpus let the topic model to create better and more significant topics both under the human and machine point of view as explained by the score metric.

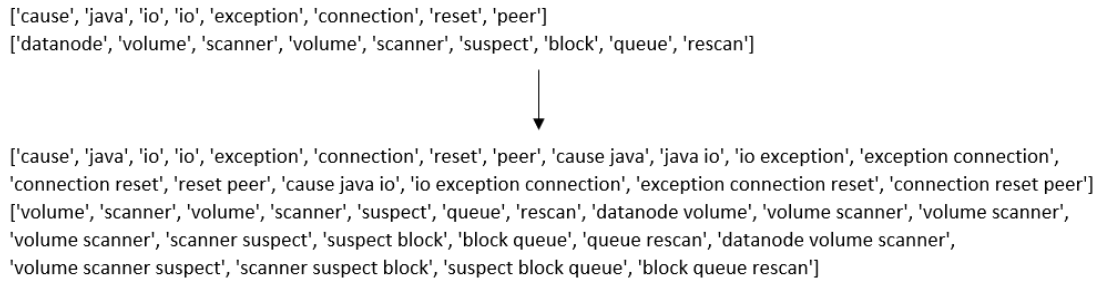



**Figure 3.16:** N-gram example

This study executes all the aforementioned steps to process data, further considerations are proposed later inside the Results section, where more advanced considerations are done on the basis of the results.
It is important to underline how this whole pipeline is created with the objective to have an almost black box where the supervisor can interact but, at the same time, this interaction is limited as much as possible. This is a significant goal since, even if the supervision is always essential when dealing with textual data, a tool able to adequately clean log data is crucial in this context of Topic Modeling.
The goodness of the proposed method is evaluated later by looking at the words inside the topic clusters, the only option to understand the quality of the data. In general, all the variations and the analysis come from hundreds of trials, that allowed to understand the best way to approach a task that is really hard. In final, it must be remembered that several slightly variations could be implemented and that the optimal results keeps a subjective part in the judgement.

# Chapter 4

# Topic Modeling

In this section I am going to explore different Topic Modeling algorithms, explaining them theoretically, to better understand the behaviours in Chapter 5, assigned to the results. As anticipated, there are two main branches about Topic Modeling algorithms. The first one follows more statistical and algebraic way, based on pure mathematics. The second one, instead, chases the actual trend of Deep Learning. Indeed, thanks to the introduction of Neural Networks, we can inspect new ways to accomplish the task of Topic Modelling.

## 4.1 Classic Approaches

In this first section, the classical algorithms are proposed, based on two main distinct concepts. The first one is about conditional probability, used inside LDA; the second one regards matrix decomposition, used in NMF with Matrix Multiplication and in LSI with Singular Values Decomposition.

#### Singular Value Decomposition

Singular value Decomposition (SVD) is a factorization method for matrices. Given a real matrix $M = m \times n$, it is decomposed as $U\Sigma V^T$ where $U = m \times m$ is an orthogonal matrix with the left singular vectors, $V = n$ is the orthogonal matrix with the right singular vectors and $\Sigma$ is a diagonal matrix whose values $\sigma_{i,i}$ are the singular values of $M$.

### 4.1.1 LDA

In natural language processing, the Latent Dirichlet Allocation (LDA) [21] is an generative statistical model, that allows collections of data, such as text corpora, to be described by unobserved groups, i.e topics, that provide an explicit representation of documents. D. Blei, A. Ng and M. Jordan created LDA [22], a model able to describe large collections of data without using and, at the same time, improving existing text modeling approaches, based on dimensionality and feature reduction. Indeed, they structured LDA as a three-level hierarchical Bayesian model, where each item of a collection is a finite mixture over an underlying set of topics. The final objective is to find a probabilistic model of a corpus, that not only assigns high probability to members of the corpus, but also high probability to other similar documents.

Before going into more technical details, we need some information about notation and terminology. Formally, the following terms are defined:

- A *word* is the basic unit of discrete data, defined as an item of the vocabulary indexed by *1, ..., V*.
- A *document* is a sequence of $N$ words denoted by $\mathbf{w} = (w_{i,1}, w_{i,2}, ..., w_{i,N})$ where $w_{i,n}$ is the $n$th word in the $i^{TH}$ sequence.
- A *corpus* is a collection of $M$ documents denoted by $\mathbf{D} = (w_1, w_2, ..., w_N)$

#### Latent Dirichlet Allocation Generative Process

LDA, as text modeling approaches, is based on the assumption and on the utilization of the Bag-of-Words representation. This representation allows documents to be described as sparse vectors, containing the counting of the tokens that occur on each document. This document representation assumes the order of the terms in a document is irrelevant and, at the same time, the documents order in the corpus is not relevant. On these basis, the LDA algorithm takes in consideration the exchangeability property of both terms and documents. LDA is a generative probabilistic model whereby corpora of documents are created. In order to create them, topics and words have to be characterized as probabilistic distributions, since the model draws the elements of the documents based on these probabilities.
LDA asssumes the followig generative process for each document $w_i$ in a corpus $D$:

1. Choose $N \sim Poisson(\xi)$, that represents the distribution of the documents lengths;
2. Choose $\theta \sim Dirichlet(\alpha)$. $\theta$ is vector where each element $\theta_k$ represent the topic proportion for the topic $k$ inside that document;

3. For each of the $N$ words in $w_n$:

   (a) Choose a topic $z_n \sim Multinomial(\theta)$, which denotes the topic assignment.

   (b) Choose a word $w_n$ from $p(w_n|z_n, \beta)$, a multinomial probability conditioned on the topic $z_n$.

In other words, each LDA document is represented as a mixture of topics, where each topic is a probability over the entire vocabulary: this is encoded with $k$-dimensional vector $\beta$. The topic proportion for a document is denoted with $\theta$, also a $k$-dimensional vector, where each element is the topic proportion for that topic inside the document and it is computed using as prior the Dirichlet distribution of parameter $\alpha$. Accordingly, for each word the topic-word assignment is made on the basis of sampling from the document-topic vector $z$, extracted from a Multinomial distribution of parameter $\theta$.

From this explanation, we can see how the model considers provided a fixed $k$, the number of topic, and the parameters $\alpha$ and $\beta$, which respectively represent the prior belief on document-topic distribution and on topic-word distribution. These parameters are the ones of interest when the goal is to obtain the best model.

Given $\alpha$ and $\beta$, the joint multivariate distribution of the topic mixture $\theta$, the set of $N$ topics $z$ and the set of $N$ terms $w$ is given by:

$$p(\theta, z, w|\alpha, \beta) = p(\theta|\alpha) \prod_{n=1}^{N_d} p(z_n|\theta)p(w_n|z_n, \beta) \tag{4.1}$$

Integrating over $\theta$ and summing over $z$, we obtain the marginal distribution of the document:

$$p(w|\alpha, \beta) = \int_\theta p(\theta|\alpha)\left(\prod_{n=1}^{N_D} \sum_{z_{d,n}=1}^{k} p(z_n|\theta)p(w_n|z_n, \beta)\right)d\theta \tag{4.2}$$

while, taking the product of the marginal probabilities of single documents, we obtain the probability of a corpus $D$:

$$p(D|\alpha, \beta) = \prod_{d=1}^{M} \int_\theta \left(\prod_{n=1}^{N} \sum_{z_n} p(z_{d,n}|\theta_d)p(w_{d,n}|z_{d,n}, \beta)\right)p(\theta_d|\alpha)d\theta_d \tag{4.3}$$

where we can appreciate how the topic $z_{d,n}$ depends on the per-document topic proportion $\theta_d$ and the observed word $w_{d,n}$ depends on the topic assignment $z_{d,n}$ and all of the topics $\beta$. These dependencies make the definition of LDA.

Other than the mathematical representation just explained, LDA can be also explained as a probabilistic graphical model.

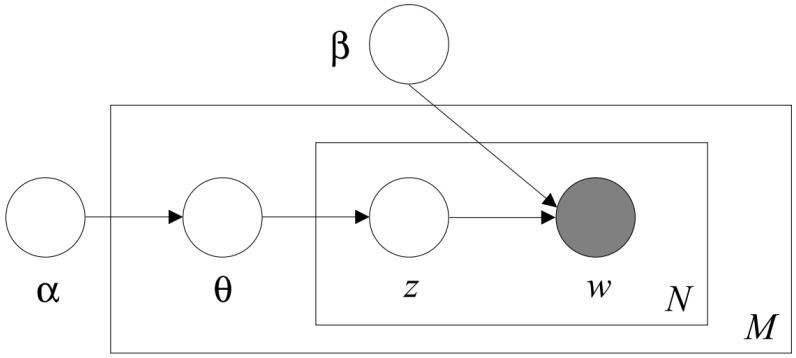



**Figure 4.1:** LDA graphical model

As shown in figure 4.1, there are 3 levels, which go from outside towards inside the corpus, the document and the terms. Starting from the outer level, we can see how $\alpha$ and $\beta$ are sample once for the generative process. By going in the middle level, $\theta$ is obtained from *alpha*, sample for each document, at last, in the most inner box, there are $z$, obtained from $\theta$, and $w$, coming from $z$ and $\beta$.

### Inferential Problems

Other than being a generative model, LDA can be used to do the inference of the posterior distribution of latent variables for a given corpus and then it recovers its structure. The variables that describe the document are the distributions of the topic mixture and the set of $N$ topics $z$:

$$p(\theta, z|w, \alpha, \beta) = \frac{p(\theta, z, w|\alpha, \beta)}{p(w|\alpha, \beta)} \tag{4.4}$$

However, the computation of the integral in the equation 4.2, done to obtain that distribution, is unfeasible and so it is impossible to exactly solve this posterior Bayesian inferential problem. To overtake this issue, several approximate inference algorithms have been presented for the LDA inferential problem. The original Latent Dirichlet Allocation paper itself implements a Variational Bayes approximation, but, over the years, other alternatives, such as the Monte Carlo simulation and Gibbs Sampling, have been proposed to approximate the probability distributions.

### 4.1.2 NMF

Non-negative matrix factorization (NMF) [23] is an algorithm in multivariate analysis and linear algebra, where a matrix $V$ is factorized into two matrices $W$ and $H$, with all three matrices have non-negative elements as property. NMF is included among the state of the art feature extraction algorithms and it is very useful when there are many ambiguous attributes or with weak predictability. By combining attributes, NMF can produce meaningful patterns, topics, or themes.
Due to the factorization, each feature created by NMF is a weighted linear combination of the original attributes, where the weight coefficients represent the weight of each attribute on the feature. In particular, a separate coefficient exist for each numerical attribute and for each distinct value of each categorical attribute. NMF is used in many different fields. In text mining application, like in this project, a document-term matrix is built with the weight of various terms from a set of documents. Then, this matrix is factorized into two different matrices: a term-feature, derived from the contents of the documents, and a feature-document matrix, that describes data clusters of related documents. Other than text mining, thanks to its properties, NMF is hugely adopted in many different fields:

- Astronomy, for dimension reduction of astrophysical signals, and to study common properties of astronomical objects, by the means of spectroscopic and imaging observations;
- Data Imputation for missing data, NMF can take missing data while minimizing its cost function, rather than treating these missing data as zeros;;
- In Bioinformatics, NMF has been successfully applied for clustering gene expression and DNA methylation data. In the cancer mutations analysis, NMF has been used to identify sources of variations, as cell types, tissue composition and tumor clonality;
- Speech denoising when the noise is non-stationary. The key idea, differently from classical statistical approaches, is clean speech signal can be sparsely represented by a speech dictionary, but non-stationary noise cannot. Similarly, non-stationary noise can also be sparsely represented by a noise dictionary, but speech cannot. This is applied by separating the STFT via NMF;

#### Structures

The starting point is a matrix $V$, described by the product of the matrices $W$ and $H$, such that $V = WH$. In this way, each column of $V$ can be computed as the linear combination of the column vectors in $W$, by using the provided coefficients by the columns of $H$. More in detail, the previous formulation can be described as

$v_i = Wh_i$, where $v_i$ is the $i^{TH}$ columns of $v$ and $h_i$ is the $ith$ column vector of the matrix $H$. A NMF important property is the dimension of the factor matrices, that can be lower than the dimension of the product matrix; indeed, NMF generates factors with lower dimensions, compared with the original matrix.
This point is NMF basis, since we can consider each original document as being built from a small set of hidden features, generated by NMF. To better understand those matrices, we can start with $W$: it is the feature matrix where each column, which represents the feature, can be seen as a set of words. Each cell in this set has a value that defines the importance of the corresponding word in that feature: in detail higher is the value, more relevant is that word. Instead, $H$ is the coefficient matrix where each column corresponds to an original document, with a cell value that associates the document rank for a feature. It is possible to reconstruct a document from the input matrix, as the linear combination described above.

### Clustering Properties

NMF has an important clustering property, indeed it is able to automatically group the columns of input data $V$. More specifically, the reconstruction of $V$ as $V \sim WH$ is achieved by finding $W$ and $H$ such that the error function $||V - WH||_F$, subject to $W \geq 0$ and $H \geq 0$, is minimized.
Adding the orthogonality constraint on $H$, i.e $HH^T = I$, the previous minimization is mathematically equivalent to the minimization of K-means clustering. In detail, the matrix $H$ provides the cluster membership, indeed, if $H_{k,j} > H_{i,j}$ for all $i \neq k$, the input data $v_j$ belongs to $k$-th cluster, while $W$ gives the cluster centroids, where the $k$-th column gives the cluster centroid of $k$-th cluster. When the orthogonality constraint $HH^T = I$ is not explicitly imposed, the orthogonality holds on a large extent, and also the clustering property holds on. Clustering is the main objective of most NMF data mining applications.
When the error function used is Kullback–Leibler divergence is an interesting consideration: NMF is identical to the Probabilistic Latent Semantic Analysis [24].

### Online NMF with Outliers

For this problem, different types of solution exist, like the ones based on Approximated NMF, where the columns of $W$ and the number of rows of $H$ are selected such that the product $WH$ will become an approximation to $V$, or the ones based on Convex NMF, where it restricts the columns of $W$ to convex combination of the input data vectors $v_i$. Overall, different types of non-negative matrix factorizations can be implemented, that come from using different cost functions to measure the divergence between $V$ and $WH$, and from the $W$ and/or $H$ matrices regularization. Two of the most common divergence functions are the squared error and the extension of the Kullback–Leibler divergence to positive matrices. Overall, each

divergence is linked to a different NMF algorithm, that minimizes the divergence by iteratively updating rules. Many NMF standard algorithms analyze all the data together, given the whole matrix available from beginning. These algorithms are not usable in applications where data do not fit into memory or where data are provided in streaming fashion.

Following this way, in 2016 R. Zhao and V. Tan proposed an online algorithm for NMF [25], where they consider the presence of outliers: it is called Online Nonnegative Matrix Factorization with Outliers - ONMFO.

The canonical NMF problem can be stated by the minimization problem

$$min_{W \in C, h_i{}_{i=1}^{N} \geq 0} \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} ||v_i - W h_i||_2^2 \tag{4.5}$$

where $C$ is the constraint set for $W$ and $N$ is the number of data samples. When there are outliers inside data, many algorithms unsatisfactorily perform: Robust NMFs have been proposed to overcome this problem. There are two possible categories of solution: the first one suggests to replace the $L2$ loss with another loss measure; the second model suggests the outlier vector $r$ and the associated outlier matrix $R$. In general, these proposals do not admit strong recovery guarantees of the original data matrix, since the new formulations are not convex. Starting from the previous works, the authors explicitly modelled the outlier vectors and they assumed data generation distribution $P$ is time-invariant. They first define the loss function with respect to $v$ and $W$

$$l(v, W) = min_{h \geq 0, r} \frac{1}{2} ||v - Wh - r||_2^2 + \lambda ||r||_1 \tag{4.6}$$

Given a finite set of data samples $v_i$, they defined the empirical loss associated with $v_i$ but, following the convention of the online learning literature, instead of minimizing the empirical loss, they aimed to minimize the expected loss, leading the solution to a non convex stochastic program.

$$min_W \Big[ f(W) = E_v[l(v, W)] \Big] \tag{4.7}$$

To tackle this problem, they leverage the stochastic majorization-minimization framework [26] proposed by J. Mairal in 2013, already used in different previous works on online matrix factorization. This framework decomposes the optimization problem in two step: non-negative encoding and dictionary update. Concretely, at a time instant $t$, the algorithm learns the coefficient vector $h_t$ and the outlier vector $r_t$ based on the newly acquired data sample $v_t$ and the previous dictionary matrix $W_{t-1}$ and in specific it solves the convex optimization problem.

$$(h_t, r_t) = argmin_{h \geq 0, r} \tilde{l}(v_t, W_{t-1}, h, r) \tag{4.8}$$

The initial basis matrix $W_0$ is randomly chosen in $C$, then on the past statistic the basis matrix is updated as

$$W_t = argmin_W \frac{1}{2} tr(W^T W A_t) - tr(W^T B_t) \tag{4.9}$$

where $A$ and $B$ are the sufficient statistics defined as

$$A = \frac{1}{t}\sum_{i=1}^{t} h_i h_i^T, B = \frac{1}{t}\sum_{i=1}^{t}(v_i - r_i)h_i^T \tag{4.10}$$

To solve the last two equations, the authors proposed two solvers based on Proper Generalized Decomposition and on Alternating Direction Method of Multipliers and they followed this algorithm:

**Algorithm 1** Online NMF with outliers (ONMFO)

linenosize=

1: **Input**: Data samples $v_{i,i\in[N]}$, penalty $\lambda$, initial dictionary matrix $W_0$
2: **Initialize** sufficient statistics: $A_0 := 0$, $B_0 := 0$
3: **for** $t = 1$ to $N$ **do**

1. Acquiring a data sample $v_t$
2. Learning the coefficient vector $h_t$ and the outlier vector $r_t$ based on $W_{t-1}$, using the solver, based on PGD or ADMM

$$(h_t, r_t) = argmin_{h\geq 0, r} \tilde{l}(v_t, W_{t-1}, h, r) \tag{4.11}$$

3. Update the sufficient statistics

$$A_t := \frac{1}{t}\Big\{(t-1)A_{t-1} + h_t h_t^T\Big\}, B_t := \frac{1}{t}\Big\{(t-1)B_{t-1} + (v_t - r_t)h_t^T\Big\}, \tag{4.12}$$

4. Learning the dictionary matrix $W_t$ based on $A_t$ and $B_t$, using the solvers based on PGD or ADMM

$$W_t = argmin_W \frac{1}{2} tr(W^T W A_t) - tr(W^T B_t) \tag{4.13}$$

4: **end for**
5: **Output**: Final dictionary matrix $W_N$

This algorithm proved that the sequence of objective values almost surely converge by appealing to quasi-martingale convergence theorem and they also showed the sequence of learned dictionaries converges to the set of stationary points

of the expected loss function.
Overall, this NMF implementation can be easily seen for text mining as follow:

- W is a word-topic matrix;
- H is a topic-document matrix;
- V is an input corpus batch, word-document-matrix;
- A, B the matrices that accumulate information from every consecutive chunk;

So, by replicating the previous algorithm, it can now be rewritten as:

**Algorithm 2** Gensim - Online NMF with outliers (ONMFO)

1: **Initialize** $W$, $A$ and $B$ matrices
2: **Input**: Corpus
3: **Split**: Corpus into batches
4: **for** $v$ in batches **do**

1. Infer $h$:

   Do coordinate gradient descent step to find $h$ that minimizes $||v - Wh||_2^2$
   Bound $h$ so that it is non-negative

2. Update A and B:

   $A = h.dot(h_t)$
   $B = v.dot(h_t)$

3. Update W:

   Do gradient descent step to find $W$ that minimize $\frac{1}{2}tr(W^T W A_t) - tr(W^T B_t)$

5: **end for**
6: **Output**: Final dictionary matrix $W_N$

### 4.1.3 LSA

Latent semantic analysis (LSA) [27] is a natural language processing technique coming from the field of distributional semantics, that analyze relationships between a set of documents and their terms by producing a set of concepts related to the documents and terms. The base assumption of LSA is that words that have similar meaning will occur in similar pieces of text. The principal operation

is the decomposition of the input matrix. It is a Word Count matrix where each rows represent unique words and each columns represent a document, and the mathematical technique called Singular Value Decomposition to reduce the number of rows, while preserving the similarity structure among columns is applied. Afterwards, the documents are compared by the mean of the cosine similarity, that provides values close to 1 for similar documents, while close to 0 for dissimilar one.

### Formulation

As anticipated, LSA uses a sparse document-term matrix which describes the occurrences of terms in documents. different techniques exist to create this matrix, among which the most common are the Bag-of-Word and the TF-IDF. As explained previously in LDA section, Bag-of-Word makes a sparse matrix where in each row there are the frequencies for the features that are inside that row. TF-IDF, instead, weights each element of the matrix to be proportional to the number of times the term appears in each document: in this way, rare terms are up-weighted to reflect their relative importance.

As shown with the previous models, this matrix is shared by standard semantic models, though in many cases it is sufficient keeping the tuples corresponding to the words and their values, since the mathematical properties of matrices are not always used.

After the creation of the occurrence matrix, LSA finds a low-rank approximation to the term-document matrix. About these approximations, there are different reasons about this choice:

- When the original term-document matrix is too large for the computing resources, the approximated low rank matrix is necessary to carry out all the processing.
- The approximation can help with term-document matrix since in many cases are noisy; in this way, the produced approximated matrix is interpreted as a de-noisified matrix.
- The original term-document matrix is overall more sparse with respect to what could be the "true" term-document matrix. Indeed the original matrix shows up only words that are in each document, whereas the focus could be in all words related to each document.

The consequence of the rank lowering is some dimensions are combined and the result depends on many terms: for example $(pizza), (pasta), (sea)$ can be approximate as follow $(1.5638 * pizza + 0.3104 * pasta), (sea)$ The approximation helps to reduce the issue related to synonyms since it is expected that dimensions linked with similar meaning terms are merged. Other than synonymy, this behaviour comes

handful also with polysemy of words. In this last case, components of polysemous words, that point in the "right" direction, are added to the components of words with similar meaning. On the other hand, components that point in other directions are deleted or made smaller than components in the directions corresponding to the intended sense.

About the derivation of LSA, we can take a matrix $X$ where each element $(i, j)$ describes the occurrence of term $i$ in document $j$. The row $t_i^T$ of this matrix is the vector corresponding to a term, giving its relation to each document, each column instead represent a document with all the relations with each term. If we take two term vector $t_i$ and $t_p$, with their dot product $t_i^T t_p$, it is obtained the correlation between terms over the set of documents. As follows, the matrix product $XX^T$ contains all these product with the property $t_i^T t_p = t_p^T t_i$, analogously the matrix $X^T X$ provides the document correlation over terms $d_j^T = d_q = d_q^T d_j$.

The algebrical method can be applied on $X$, method called Singular Value Decomposition that allows to decompone the matrix $X$ in two orthogonal matrices $U$ and $V$ and a diagonal matrix $\Sigma$ such that $X = U\Sigma V$.

With this decomposition the two previous dot matrices done on $X$ can be written as $XX^T = U\Sigma\Sigma^T U^T$ and $XX^T = U\Sigma\Sigma^T U^T$. Since $\Sigma\Sigma^T$ and $\Sigma^T\Sigma$ are diagonal, as consequence $U$ contains the eigenvectors of $XX^T$ while $V$ the eigenvectors of $X^T X$ and both have the same non-zero eigenvalues due to the absence of non-zero entries on both.

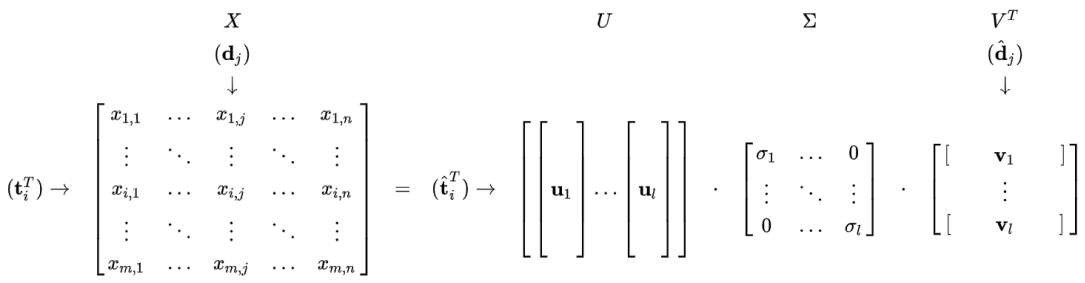



**Figure 4.2:** LSA Decomposition

As summarized in the last figure, the $\sigma$ values are the singular values while the $u$ and $v$ values are correspondingly the left and the right singular vectors. Also from the picture can be appreciated how only the $ith$ row $(\tilde{t})_i^T$ of U contributes to $t_i$ and, at the same time, only the $jth$ column $(d)_j$ of $V^T$ contributes to $d_j$. Hence, if we take the $k$ largest singular values and their corresponding singular vectors from $U$ and $V$, we get rank $k$ approximation of $X$ with minimal possible error that allows to consider term and document vectors as a semantic space. Indeed, the previous row term vector $\tilde{t}_i^T$ has $k$ entries mapping it to a lower-dimensional space and the same thing for $\tilde{d}_j$, which they allow to rewrite the approximation as

$X_k = U_k \Sigma_k V_k^T$.
Provided this last equation, it is possible to do different operations:

- Check the relation between a pair $(j, q)$ of documents by looking at the similarity between the vectors $\Sigma_K \tilde{d_j}$ and $\Sigma_K \tilde{d_q}$ in the lower dimensional space;
- Comparing a pair $(i, p)$ of terms by comparing their vectors $\Sigma_K \tilde{t_i^T}$ and $\Sigma_K \tilde{t_p^T}$ in the lower dimensional space;
- For Topic modeling we can cluster documents and terms using cluster algorithms by the usage of similarity measures;
- Given a query $q$ it is possible to compare it in the lower dimensional space by using the same transformation done on documents $\tilde{d_k} = \Sigma_K^{-1} U_k^T d_j$, obtaining $\tilde{q} = \Sigma_K^{-1} U_k^T q$;
- The reasoning in the previous point can be extended also to terms, obtaining $\tilde{t_i} = \Sigma_K^{-1} V_k^T t_i$;

### Limitations

LSA has also some important drawbacks:

- It might be difficult to interpret the resulting dimensions. For instance, in $[(pizza), (pasta), (flower)]$, approximated as $[(1.3452 * pizza + 0.2828 * pasta), (sea)]$, the $(1.5638 * pizza + 0.3104 * pasta)$ component could be interpreted as "food". However, it is very likely that cases close to $(pizza), (bottle), (sea)$ are approximated as $(1.5638 * pizza + 0.3104 * bottle), (sea)$. So, many times produced results are not understandable under human point of view.
- As stated before, LSA can help with polysemy since if a word has different meaning, it is only considered with a single form due to the used representation. So, the vector representation can be considered as an average of all the word different meanings in the corpus, that can make it difficult for comparison.
- With Bag of words representation it is lost the order of the words. For this reason, the utilization of multi-gram dictionary is a possible solution to find direct and indirect association as well as higher-order co-occurrences among terms.
- LSA assumes that words and documents form a joint Gaussian model, while a Poisson distribution has been observed, causing a mismatch on observed data. Due to this, different alternatives have been proposed to overcome this issues among which probabilistic latent semantic analysis for example.

## 4.2 Modern Approaches

Nowadays, many algorithms, in almost any field of Data Science, are based on Artificial Neural Networks. They introduced many improvements with respect to previous approaches, in particular about feature extraction, letting these improvements to spread and substitute many classical algorithm, considered as the state of the art previously. Artificial neural networks (ANN) [28] are computational models, composed of artificial "neurons", so called since inspired by the biological structure of the brain neural network.
ANNs were theorized in the late 1940s but only recently they have gained a lot of interest due to higher computational power; with the latest GPUs it is indeed possible to model them and carry out the training phase more efficiently. The adoption of neural networks has contributed to achieve high performance in terms of efficiency and precision in many Machine Learning and Data Mining tasks.
The basic principle of neural networks is to emulate biological neurons with artificial ones by means of nodes; each node can transmit a signal to other neurons, as it happens in the synapses of the biological brain. Each node receives values as input coming from the other neurons and emits a real value in output, received in input from other neurons. Typically, the neurons, i.e. the nodes are organized in layer. Each neuron carries out different transformations of the input. The first layer is the

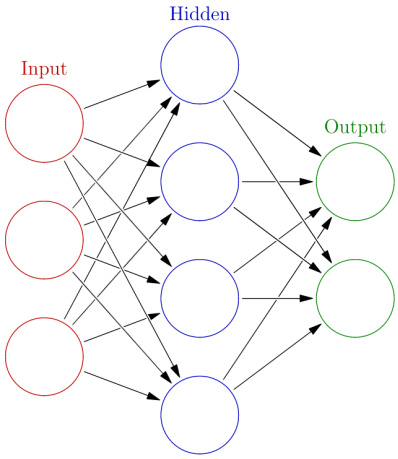



**Figure 4.3:** ANN representation

input layer, the last one is called the output layer and the others are called hidden layers. In their simpler representation, the output of each neuron is calculated by applying an activation function to the weighted sum of its inputs plus a bias. The weights $W$ and the biases $b$ are the parameters of the layer. The calculation of the output is shown below for the $jth$ node given its inputs $x$, applying the *sigmoid*

activation function:

$$y_j = sigmoid(\sum_i w_{ij}x_i + b_j) \tag{4.14}$$

In most cases an Artificial Neural Network is an adaptive system that changes its structure based on external or internal information that flows through the network itself during the learning phase. Through the training, the network takes a series of input examples and a series of expected values; automatically the parameters are progressively adjusted with the aim of minimizing a cost function $C$. The parameters are updated using gradient descent to reduce the classification cost of each instance.

The network learns by dividing the training examples into groups, called batches. The weights are updated with each batch and the cost function is computed based on them, for example with the average of the costs over all the examples in the batch. There are several hyper-parameters that can be fine-tuned to improve the learning of a network: number of hidden layers, which describe the network depth, number of nodes in each layer, activation function used in each layer (sigmoid, softmax, linear, ecc...), structures of the network and many others.

From this brief introduction, now I will explain in detail the architecture used in following models.

### Neural Networks architectures

Concerning the neural networks used inside the different models, the main focus is placed on AutoEncoders [29],[30] and their variations. An AutoEncoder [30] is an Artificial Neural Network used in unsupervised context to learns efficient codings. The encoding is validated and refined by attempting to rebuild the input from the encoding. In dimensionality reduction AutoEncoders are used to learn a data representation by training the network to ignore noisy data.
The basic architecture of AutoEncoder can be described in two main elements: an encoder that maps the input into the code, and a decoder that rebuild the input from the code. Instead of copying directly the input signal, AutoEncoders reconstruct the input approximately, keeping only the most relevant details of the data in the copy.
The simplest model is a feed-forward neural network similar to a multi-layer perceptron, where the input layer and an output layer, which have the same number of nodes, are connected by one or more hidden layers. So, the aim is to reconstruct the inputs by minimizing the difference between the input and the output, instead of making a prediction.
More in detail, AutoEncoder can be represented with two parts, the encoder and the decoder, which can be defined as transitions $\phi : X \to F$ and $\psi : F \to X$ such that $\phi, \psi = argmin_{\phi,\psi}||X - (\psi \circ \phi)X||^2$.

In the case of one hidden layer, the encoder takes the input $x \in R = X$ and maps it to $h = F$ as $h = \sigma(Wx + b)$, where $h$ is usually referred to as code, latent variables, or a latent representation, $\sigma$ is an element-wise activation function and $W$ and $b$ are accordingly the weight matrix and the bias vector. Weights and biases are usually initialized randomly, and then updated iteratively during training through back-propagation. Afterwards, the decoder reconstructs $x'$ from $h$ as $x' = \sigma(W'h + b')$ where $\sigma'$, $W'$ and $b'$ for the decoder could not be related to the corresponding values for the encoder. Considered the coding and the reconstruction equation, AutoEncoders are trained to minimise reconstruction errors, known as the "loss", and its performed through back-propagation of the error.

$$L(x, x') = ||x - x'|| = ||x - \sigma'(W'(\sigma(Wx + b)) + b')||^2 \tag{4.15}$$

The feature vector $\phi(x)$ can be seen as a compressed representation of the input, due to the lower dimensionality of the feature space $F$ with respect to the input space $X$. This implementation is called *undercomplete* AutoEncoders, where the hidden layers are smaller than the input. Instead, if they are larger, they are called *overcomplete* AutoEncoders and they can potentially learn the identity function, becoming useless, even though experimental results found that they might still learn useful features. In the ideal setting, the code dimension and the model capacity are set depending on the complexity of the data distribution to be modeled.

In literature different variations of AutoEncoder exist, and, among them, one of interest is the Variational AutoEncoders (VAE) [31],[32]. In machine learning, a Variational AutoEncoder, also known as VAE, is an Artificial Neural Network architecture belonging to the family of probabilistic graphical models and variational Bayesian methods. Often, It is associated with the AutoEncoder model due to architectural affinity, however they are different since Variational AutoEncoders compress the input information into a constrained multivariate latent distribution (encoding) to reconstruct it as accurately as possible (decoding).

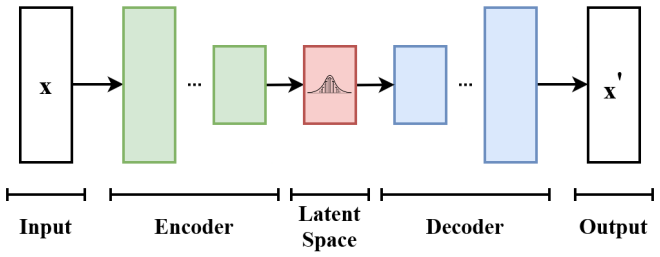



**Figure 4.4:** VAE basic represetation

Given an input dataset $x$, with an unknown probability function $P(x)$ and a multivariate latent encoding vector $z$, the goal is to model the data as a distribution $p_\theta(x)$, with $\theta$ defined as the set of the network parameters. This distribution can be defined as

$$p_\theta(x) = \int_z p_\theta(x,z)dz = \int_z p_\theta(x|z)p_\theta(z)dz \tag{4.16}$$

If we assumes $z$ with a finite dimension and $p_\theta(x|z)$ modeled as a Gaussian distribution, then $p_\theta(x)$ is a mixture of Gaussian. In this way the relationship between input data and its latent representation can be defined, with $p_\theta(z)$ as the prior, $p_\theta(x|z)$ as the likelihood and $p_\theta(z|x)$ as the posterior. However, the computation of the prior is almost in any case unfeasible, for this reason is common to use an approximation $q_\Phi(z|x) \approx p_\theta(z|x)$ to translate the problem into the AutoEncoder domain, in which $p_\theta(x|z)$ is carried by the probabilistic decoder, while the approximation by the probabilistic decoder.

For Variational AutoEncoders the idea is to jointly minimize the generative model parameters to reduce the reconstruction error between the input and the output of the network, and $\Phi$ to have $q_\Phi(z|x)$ as close as possible to $p_\theta(z|x)$.

To define the distance loss, the Kullback-Leibler divergence represent a good option to squeeze $q_\Phi(z|x)$ under $p_\theta(z|x)$.
It can be defined as:

$$D_{KL}(q_\Phi(z|x)||p_\theta(z|x)) = \int q_\Phi(z|x) log \frac{q_\Phi(z|x)}{p_\theta(z|x)} dz$$

$$= \log(\mathrm{p}_\theta(x)) + D_{KL}(q_\Phi(z|x)||p_\theta(z)) - E_{z\sim q_\Phi(z|x)}(log(p_\theta(x|z))) \tag{4.17}$$

If we move $D_{KL}(q_\Phi(z|x)||p_\theta(z))$ to the left side, we can maximize the whole left hand side of the equation in order to minimize the distances between the real posterior and the estimated one. This minimization is equal to minimize the negative log likelihood, which allows to obtain the following loss function, known as evidence lower bound (ELBO):

$$L_{\theta,\Phi} = D_{KL}(q_\Phi(z|x)||p_\theta(z)) - E_{z\sim q_\Phi(z|x)}(log(p_\theta(x|z))) \tag{4.18}$$

The optimal parameters are the ones that minimize this loss function. The problem can be rewritten as

$$\Theta^*, \Phi^* = argmin_{\Theta,\Phi} L_{\Theta,\Phi}, \tag{4.19}$$

providing the advantage to be jointly optimized with respect both $\Theta$ and $\Phi$.
In order to use the ELBO for training, some modifications are necessary both on the formulation and on the structure. Indeed to make the application of back-propagation processes possible, the reparameterization trick (RT) is introduced to

make it differentiable by removing the stochastic sampling. When dealing with the latent space, we can make the assumption to consider it as a set of multivariate Gaussian distribution which describe it as $z \sim q_\Phi(z|x) = N(\mu, \sigma^2)$.
Given a random variable $= N(0, I)$ and $\odot$ defined as the element-wise product, the reparameterization trick modifies the above equation as $z = \mu + \sigma \odot \epsilon$, enabling the training of VAE. Indeed, through this transformation, the probabilistic encoder learns how to map a compressed representation of the input into the two latent vectors $\mu$ and $\sigma$, while the stochastic part is excluded from the updating process since it is injected in the latent space as an external input .

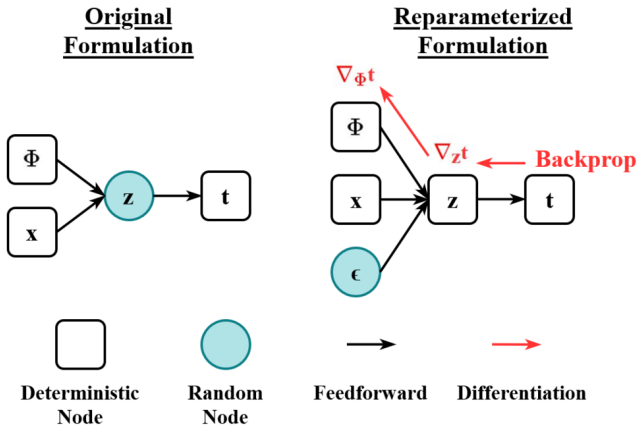



**Figure 4.5:** RT basic representation

### Feature Embedding

Feature transformation is one of the main aspects where textual data is manipulated, since models require a numerical representation to manage those type of data. Although, many different textual processing techniques exist, like the already cited Bag of Word or Term Frequency – Inverse Document Frequency, due to the higher level of difficulty achieved inside many tasks in the latest years, there was a need to improve and enhance textual data transformations. For this reason, after the introduction of neural network, there were introduced new advanced model in which are taken into account many other characteristics. In particular, word embedding is a term used for the representation of words for text analysis. Typically, language models and feature learning techniques provide a real-valued vector that encodes the meaning of the word such that the words that are closer in the vector space are expected to be similar in meaning. Conceptually it involves the embedding from an high dimensionality space per word to a continuous vector space with a

lower dimension. For the seek of this project I am going to explore two different algorithms, Word2Vec [33] and Bert [34], with the correlated sentence-BERT [35].

In 2013 T. Mikolov et al. published **Word2vec** [33][36], a NLP technique that uses a neural network model to learn word associations from a large corpus of text. After training, this model can detect synonymous words or suggest additional words for a partial sentence. As expressed in its name, Word2Vec represents each distinct word with a numerical vector representation that allows to use mathematical function, like the cosine similarity, to check the level of semantic similarity between words.
Word2Vec can be modelled as two different model architectures: the continuous

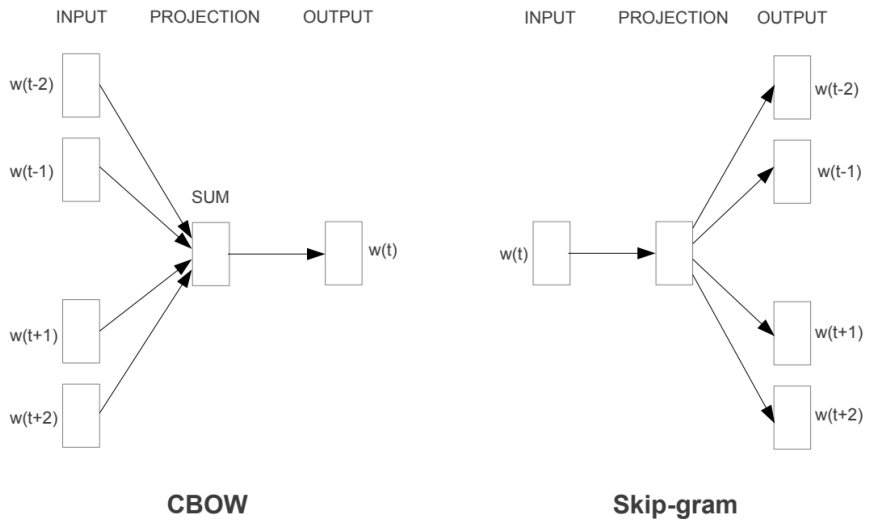



**Figure 4.6:** Word2Vec Continous Bag of Word and Skip Gram architecture

bag-of-words (CBOW) model, that predicts the current word from a window of surrounding context words where the order of context words does not influence prediction, and the continuous skip-gram where model, that uses the current word to predict the surrounding window of context words, weighting nearby context words more heavily than more distant ones. Overall, CBOW is faster, while skip-gram performs better for infrequent words.

Although the useful property to make close similar words, one of the main limitations is words with multiple meanings, relegated into a single shared representation, making hard to handle the concept of polysemy and homonymy.
In order to overcome this problem, recently, contextually-meaningful embeddings such as ELMo[37] and the most recent BERT[34] have been developed. These embeddings use a word's context to disambiguate polysemes thanks to the usage of LSTM and Transformer architectures.

**Bidirectional Encoder Representations from Transformers (BERT)** [34] is a transformer-based machine learning technique for NLP textual data processing. Unlike other language representation models, *"BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task specific architecture modifications".*
The base architecture is the same proposed in the paper "Attention in all you need" [38], the Transformer, a model architecture that completely relies on an attention mechanism to draw global dependencies between input and output, as represented in Figure 4.7.

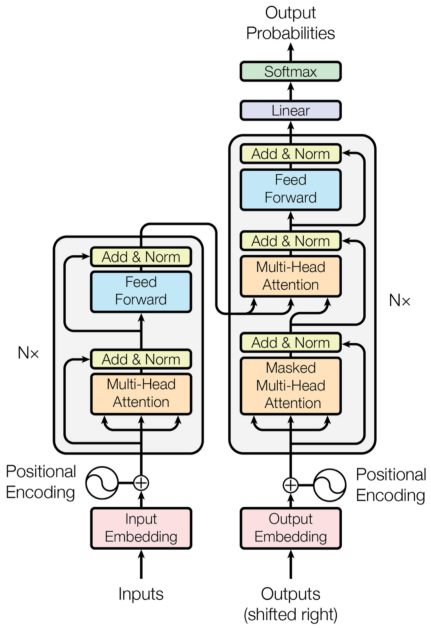



**Figure 4.7:** Attention mechanism representation

The Transformer [39][40] is the first model entirely relying on self-attention to compute representations of its input and output without using Recurrent or Convolutional Neural Network. Self-attention is a sequence-to-sequence operation where there are input vectors $x$ and the corresponding output vectors $y$, both with the same dimension. To produce an output vector, the self attention operation takes a weighted average over all the input vectors $y_i = \sum_j w_{i,j} x_j$, where $j$ indexes over the whole sequence and the weights, that sum to one over $j$, are derived from $x$ as $w_{i,j} = \frac{exp(x_i^T x_j)}{\sum_j exp(x_i^T x_j)}$. Every input vector is used in the self-attention mechanism

for the *Query*, the *Key* and the *Value*. In every role, it is compared to the other vectors to get its own output $y_i$(Query), to get the $j$-th output $y_j$(Key) and to compute each output vector once the weights have been established (Value).

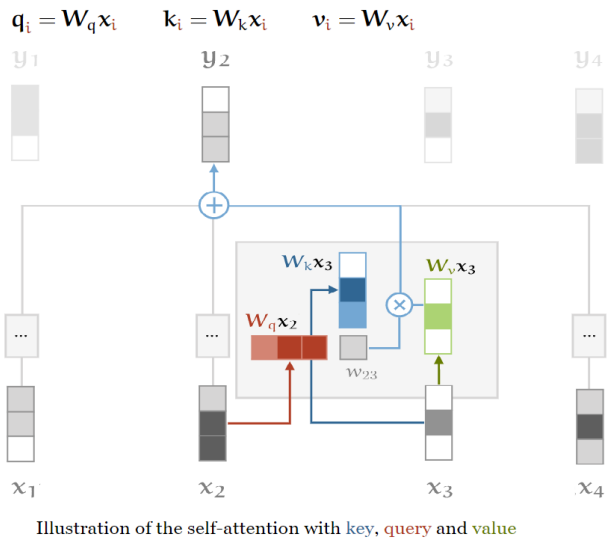



**Figure 4.8:** Transformer Implementation

These three matrices are usually known as $K$, $Q$ and $V$, three learnable weight layers that are applied to the same encoded input. Consequently, due to the derivation from the same input, we can apply the attention mechanism of the input vector with itself, calling it "self-attention". Afterwards the attention scores, a measure about how much focus to place on other words of the input sequence with respect to a word at a certain position, are computed on $Q$, $K$ and $V$ matrices. the attention score is defined as

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{4.20}$$

So far, the attention scores are focused on the whole sentence at a time, causing the production of the same results even if two sentences contain the same words in a different order, when instead we would like to use different segments of the words. To enable this use, several self attention heads are combined, dividing the words vectors into a fixed number $h$ of chunks, and then self-attention is applied on the corresponding chunks, using $Q$, $K$ and $V$ sub-matrices.

Since the model contains no recurrence and no convolution, the model needs some information about the relative or absolute position of the tokens in the sequence must be injected to use the order of the sequence. Hence, it is added

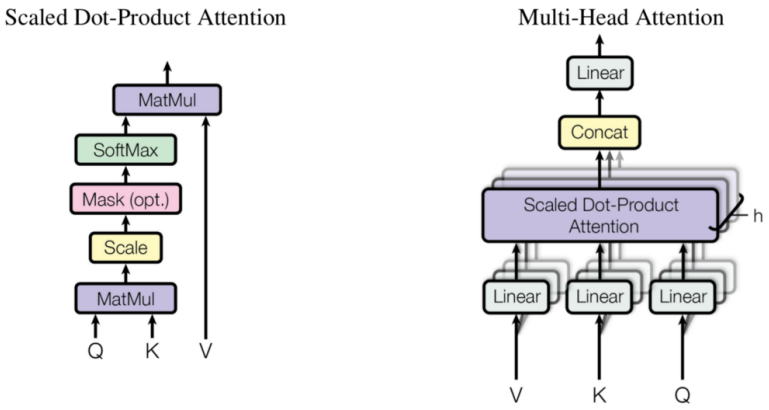



**Figure 4.9:** Scale Dot-Product Attention vs Multi-Head Attention

*positional encodings* to the input embeddings at the beginnning of the encoder and decoder stacks.

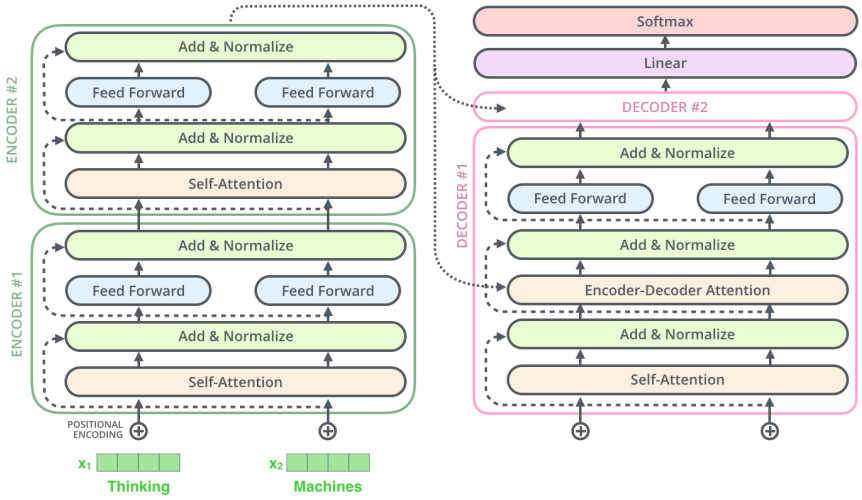



**Figure 4.10:** Complete Transformer Architecture

Provided the description of BERT base structure, BERT is pre-trained in two different mode: Masked Language Model, where some percentage of the input tokens are masked and it has to predict those masked tokens, and Next Sentence Prediction, where provided a group of sentences it determines the order of those sentences. Following, usually a fine-tuning phase is applied, to focus on specific tasks like Question Answering.

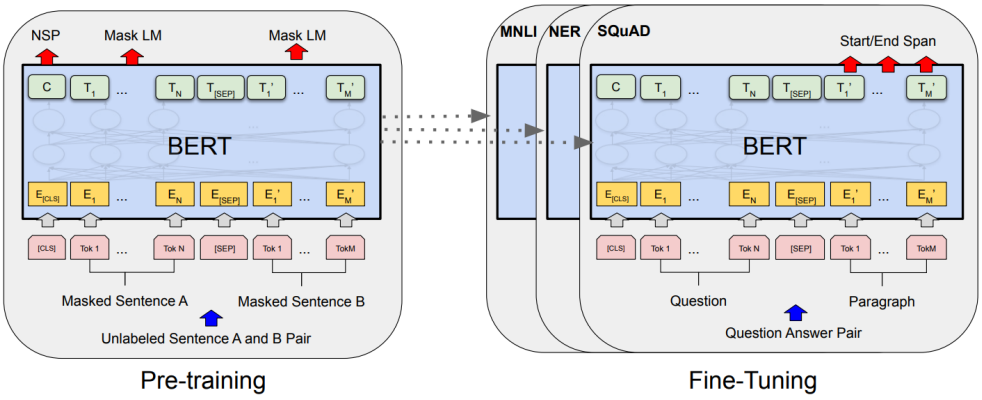



**Figure 4.11:** Overall pre-training and fine-tuning procedures for BERT

**Sentence-BERT (SBERT)** [35] is a modification of the pre-trained BERT network. It uses Siamese and triplet network structures to derive semantically meaningful sentence embeddings, that can be compared using cosine-similarity, allowing better computational performance with the same accuracy of BERT. SBERT is a so-called twin network which allows it to process two sentences in the same way at the same time. These two twins are identical in every parameter (their weight is tied), which allows to think about this architecture as a single model used multiple times.

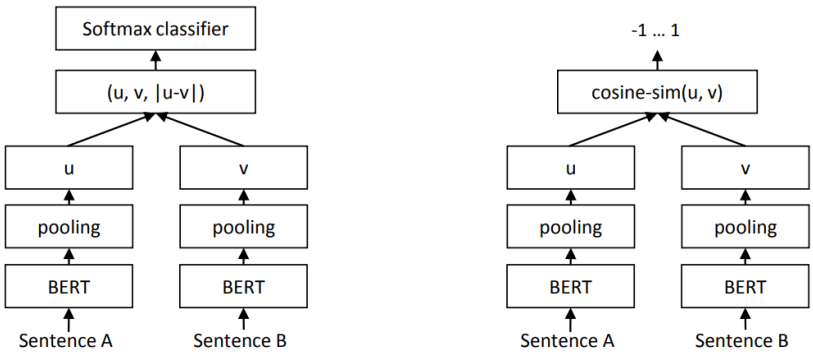



**Figure 4.12:** Twin Network architectures of Sentence-BERT. On the right, the architecture for the classification is represented, while on the right the one used at inference

BERT makes up the base of this model, where a pooling layer has been appended. This pooling layer enables to create a fixed-size representation for input sentences of varying lengths. Since the purpose of creating these fixed-size sentence embeddings is to encode the semantics, the authors fine-tune the network on Semantic Textual

Similarity data with a dataset of over 1.000.000 sentence pairs. To achieve this aim, they trained the network on the classification task of assigning a label to each pair $u$ and $v$ of sentences between *contraddiction*,*entailment* and *neutral*. In detail this is done by using the classification objective function defined as $o = softmax(W_t(u, v, |u - v|))$.

The other approach instead is base on training the network on Triplet Objective function. Given a base sentence $a$, a positive sentence $p$, and a negative sentence $n$, the network is tuned with the triplet loss such that the distance between $a$ and $p$ is smaller than the distance between $a$ and $n$. Mathematically, it is formulated with following loss function $max(||s_a - s_p|| - ||s_a - s_n||+, 0)$, where $s_x$ is the generic sentence embedding and the margin that ensures that $s_p$ is at least closer to $s_a$ than $s_n$.

Both the implementations overcome the previous state of the art methods, making a new baseline for this task. For the implementation SBERT can be used with its framework based on the load of a pre-trained model. Indeed, depending on the task, the most adequated model can be used, made on the best architecture considered by their author.

### 4.2.1 ProdLDA

In 2017, A.Srivastava and C.Sutton proposed ProdLDA [41] tyring to address those issues related to LDA [22], and also to try, at the same time, to obtain better clustered topics. In the section about LDA, we have seen how in the generative model, the marginal likelihood of posterior inference over the hidden variables $\theta$ and $z$ is intractable, due to the coupling between $\theta$ and $\beta$ under the multinomial assumption. An approximation to compute that inference more efficiently in topic models is mean field variational inference, which breaks the coupling between $\beta$ and $z$ by adding the variational parameters $\gamma$ over $\theta$ and $\phi$ over $z$. It enables to substitute the posterior $p(\theta, z|w, \alpha, \beta)$ with the best possible approximation $q(\theta, z|\gamma, \phi) = q_\gamma(\theta) \prod_n q_\phi(z_n)$ that move the optimization problem to minimize that so called *ELBO*

$$L(\gamma, \phi|\alpha, \beta) = D_{KL}\Big[q(\theta, z|\gamma, \phi)||p(\theta, z|w, \alpha, \beta)\Big] - log(p(w|\alpha, \beta)) \tag{4.21}$$

Since the mean field method optimizes each document over an independent set of variational parameters, for LDA this optimization has closed form coordinate descent equations due to the conjugacy between the Dirichlet and Multinomial Distribuitions. This issue limits its flexibility, due to necessity to rely on the ability of the supervisor to derive the closed form updates.

To overcome this issue Auto-Encoding Variational Bayes (AEVB) [31] can be used to reformulate the ELBO as

$$L(\gamma, \phi|\alpha, \beta) = D_{KL}\Big[q(\theta, z|\gamma, \phi)||p(\theta, z|\alpha)\Big] + E_{q(\theta,z|\gamma,\phi)}\Big[log(p(w|z, \theta, \alpha, \beta))\Big] \quad (4.22)$$

where the first part tries to match the variational posterior to the prior, while the second term is the reconstruction term used to ensures the variational posterior choose good values for the data.

Other than re-writing the ELBO, AEVB uses an inference network that computes the variational parameters, provided the data in input. The variational parameters $\gamma$ can be obtained by optimizing the ELBO where, to compute the expectation with respect to $q$, the re-parameterization trick (RT) is used, which defines a variate $U$ and a re-parametrization function $F$ such that $F(U, \gamma)$ has distribution over $q_\gamma$.

Applying AEVB to topic models is not straight forward. Indeed, to use the RT it is needed to determine the re-parameterization function for $q(\theta)$ and $q(z_n)$, while the second issue is the problem of component collapsing.

First of all, the re-parameterization can be problematic with discrete variable like $z$, drawback that in LDA is avoided by summing out the variable $z$, making to deal with only $\theta$ for the sample.
The Dirichlet prior to obtain topic proportions $\theta$ is difficult to handle with AEVB for the RT. However, RT can be used with Gaussian distribution, so the issue can be solved by constructing a Laplace approximation to the Dirichlet prior with the modification of doing in the softmax basis instead of the simplex.
It results in an approximation as a multivariate normal with mean $\mu_i$ and covariance matrix $\Sigma_1$

$$\mu_{1k} = log(\alpha_k) - \frac{1}{K}\sum_i log(\alpha_i), \Sigma_{1kk} = \frac{1}{\alpha_k}\Big(1 - \frac{2}{K}\Big) + \frac{1}{K^2}\sum_i \frac{1}{\alpha_k} \quad (4.23)$$

By bringing this appoximation back to the simplex basis, $p(\theta|\alpha)$ is approximated with a logistic normal distribution of parameter $\mu_1$ and $\Sigma_1$, $LN(\theta|\mu_1, \Sigma_1)$.

With this assumption, the variational objective function can be modified by using the logistic normal variational distribution. Two inference networks $f_\mu$ and $f_\Sigma$ with parameters $\delta$ are defined , then for a document $w$ is defined $q(\theta)$ to be logistic normal with mean $\mu_0 = f_\mu(w, \delta)$ and the covariance matrix $\Sigma_0 = diag(f_\Sigma(w, \delta)$.

With them the ELBO can be rewrited as:

$$L(\Theta) = \sum_{d=1}^{D} \Bigg[ -\Big(\frac{1}{2}\{tr(\Sigma_1^{-1}\Sigma_0) + (\mu_1 - \mu 0)^T \Sigma_1^{-1}(\mu_1 - \mu 0) - K + log(\frac{|\Sigma_1|}{|\Sigma_1|}\}\Big) + \mathrm{E}_{\sim N(0,1)}\Big[w_d^T log(\sigma(\beta)\sigma(\mu_0 + \Sigma_0^{\frac{1}{2}}))\Big]\Bigg] \quad (4.24)$$

where $\Theta$ represents the set of all the model and variational parameters, $w_1 ... w_D$ are the documents and $\sigma(...)$ is the application of the softmax.
In the equation above, the first term is the variation of the KL divergence between the two logistic normal $q$ and $\tilde{p}$ while the second term is the reconstruction error.

As already announced, AEVB has the problem related to component collapsing, which is a particular type of local optimum very close to the prior belief, early on in the training. Since the latent dimensionality of the model is increased, the KL term dominates collapsing the outgoing decoder weights for the components of the latent variable close to the prior and it do not show any posterior divergence.

In this case, this behaviour happens because of the softmax used to produced $\theta$, and it is overtaken by training the network with the ADAM optimizer [42] and high values for both moment weight and learning rate. In additions, the problem the optimizer diverges due to those higher values arises, but also in this case these issues can be avoided by adding batch normalization and dropout units.

Summing up, in LDA the distribution $p(w|\theta, \beta)$ is a mixture of multinomials. A problem with this assumption is that better predictions than the mixed components cannot be made. As consequence, topics that have poor quality and do not correspond well with human judgment are produced. Regarding this issue, ProdLDA model replaces the mixture assumption at the word-level in LDA with a weighted product of experts, which is capable to make sharper predictions, improving in topic coherence.

Hence, the ProdLDA model can be described as Latent Dirichlet Allocation where the topic matrix is not constrained to exist in multinomial simplex prior to mixing. The only changes, done from LDA, are that $\beta$ is unnormalized, and the conditional distribution of $w_n$ is defined as $w_n|\beta, \theta \sim Multinomial(1, \sigma(\beta\theta))$.

The connection to a product of experts is direct, as for the multinomial, a mixture of natural parameters corresponds to a weighted geometric average of the mean parameters. If we consider two $N$ dimensional multinomials with mean vectors $p$ and $q$, the corresponding natural parameters are $\sigma(r)$ for $p$ and $\sigma(s)$ for $q$. If we also take into consideration $\delta \in [0,1]$, we obtain

$$P\Big(x|\delta r + (1-\delta)s\Big) = \prod_{i=1}^{N} \sigma(_i + (1-\delta)s_i)^{x_i} = \prod_{i=1}^{N} [r_i^{\delta} s_i^{1-\delta}]^{x_i} \quad (4.25)$$

In conclusion the ProdLDA model can be described as a product of experts $p(w_n|\theta, \beta) = \prod_k p(w_n|z_n = k, \beta)^{\theta_k}$.

### 4.2.2 Top2Vec & BERTopic

In the world of topic modeling the most widely used methods are LDA and LSA that, despite their popularity, have several weaknesses like knowing the number of topics in advance and poor performances. In the last years, advanced documents and words representations have become popular, since they are able to capture semantics aspects of words and documents. By using these techniques, Top2Vec [43] combines document and word semantic embedding to find topic vectors. In addition, it does not advanced pre-processing and it is able to automatically find the number of topics.

#### Create Semantic Embedding

For the purpose of extracting topics, the author decides to implement a solution based on jointly embedding document and word vectors. Specifically, the embedding is used to project document and words inside a space where distances between document vectors represent semantic associations or dissimilarities and word vectors are close to document vectors they best describe. Thanks to this embedding into a semantic space, it is possible to a continuous representation of topics by computing their equivalent topic vectors.

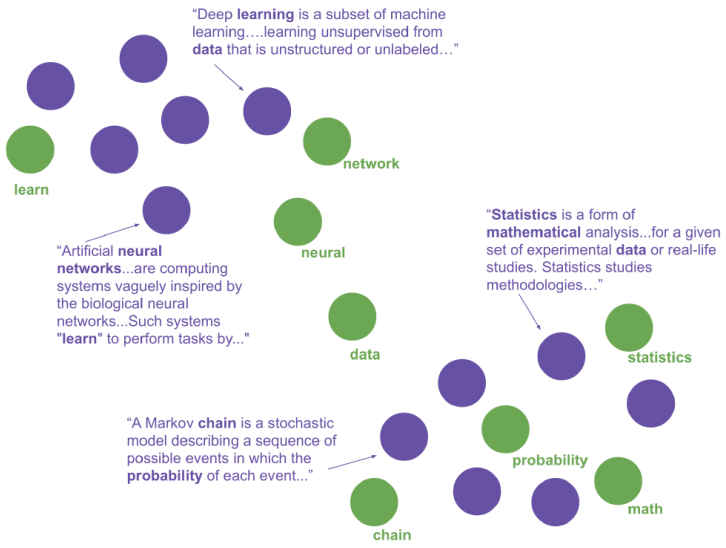



**Figure 4.13:** An example of a semantic space. The purple points are documents and the green points are words. Words are closest to documents they best represent and similar documents are close together.

The joint learning of embedded document and word vectors is done by means of Doc2Vec [44]. In particular, a Distributed Bag of Word (DBOW) model uses the document vector to predict words within a context window in the document, similarly to the mechanism used in the word2vec skip-gram model. The only difference between them is DBOW uses the document vector rather than the context word, allowing to predict the surrounding words in the context window. This similarity allows for the training of both to be interleaved, thus simultaneously learning document and word vectors which are jointly embedded.
The DBOW model consists of a matrix $D_{c,d}$, where $c$ is the number of documents in the corpus and $d$ is the size of the vectors to be learned for each document. Each row of the matrix contains a document vector $d \in R^d$ . It requires a further matrix $W^{'}_{n,d}$ for the context words, whose values come from a Word2Vec model trained on the same vocabulary of $n$ words. Afterwards, for each document $d$ the context vector $w_c \in Wn, d^{'}$ is used to predict the document vector $d$, where this prediction is done by applying the $softmax(w_c D_{c,d})$, that generates a probability distribution over the corpus for each document that has that word. The learning phase is done by using back-propagation, while stochastic gradient descent is used to update each document vector in $D_{c,d}$ and $w_c$ from $W^{'}_{n,d}$ such that the greatest probability $P(d|w)$ is achieved in the probability distribution over the entire corpus.
This process requires document vectors close to word vectors of words inside them, while distant from word vectors of words not belonging to them. As consequence, a semantic space where documents are closest to the words that belonging to them and far from words not place in them is produced. At the same time similar documents are placed close to each other in this space since pulled into the same region by similar words, conversely dissimilar documents are far apart since attracted into different regions of the semantic space by different words.
The resulting semantic space can be described as a continuous representation of topics. This model learns a matrix where each vector describes a $d$ dimensional context word vector for all $n$ words it is trained on, and where each word provides relative similarity to other word vectors in the matrix.
So, the $d$ dimensional embedding space is a continuous representation of topics since the matrix $W^{'}_{n,d}$ can be seen as a linear transformation. Indeed, when the transformation is applied to a $d$ dimensional vector from the embedding space, a $n$ dimensional vector containing the strengths of each of the $n$ words with respect to the point in the $d$ dimensional space is generated. So, the model actually learns how to transform a point $p$ in the $d$ dimensional space into probability distributions over the $n$ words and it is done by computing the $softmax(p\mathring{u}W^{'}_{n,d})$. Hence, any point $p$ in the $d$ dimensional space represents a different topic that can be represented semantically by the nearest word vectors, since the corresponding words have the highest probability in its that topic in the $d$ dimension space.

### Find Number of Topics

The semantic embedding provides the advantage of learning a continuous representation of topics. Indeed, thanks to the joint embedding, documents and words can be represented as positions in the space, where each document vector describes the topic of the document.
As previously anticipated, the word vectors closer to a document vector better describe the document topic. As consequence, in the semantic space, a dense area of documents is associated to an area of highly similar documents, and so it indicates the underlying common topic of the documents.
The centroid of the area can be calculated on the document vectors, thus defining the topic vector. The closest words to this topic vector are the words that best describe it semantically. The main consideration on this formulation is that the number of dense areas is equal to the number of topics.
To create dense cluster in the space, the author implemented a density based clustering called Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) [45]. However, the transformation of documents in high dimensional vectors causes some issues related to the curse of dimensionality, since those vectors are very sparse. This sparsity makes really difficult the creation of dense clusters with poor results and high computational costs.
As possible solution to alleviate this problem, a dimensionality reduction technique called Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP) [46] is used to reduce the semantic space and, as consequence, to use satisfactorily the HDBSCAN cluster algorithms.
The dimensionality reduction applied enables the creation of more accurate clusters of documents. The choice to use UMAP among the many different reduction algorithms lies on its strong theoretical foundations that allow to preserve original local and global structure in the reduced space. Figure 4.14 shows UMAP-reduced document vectors: a lot of global and local structure is preserved in the embedding.

So, the goal of clustering is to find dense areas where high similar documents lie, that identify the underlying topic. With this approach there are some challenges, since document vectors have different density in the space and there are areas where documents are dissimilar, that can be seen as noise. To overcome these issues, HDBSCAN is used to find dense areas of document vectors, since it is designed to handle both noise and variable densities clusters. Indeed, it assigns a label to dense cluster, that later will be used to compute topic vectors, while a noise label to document vectors that are not in a dense cluster as no being descriptive of a possible topic. Figure 4.15 shows an example of dense areas of documents identified by HDBSCAN.

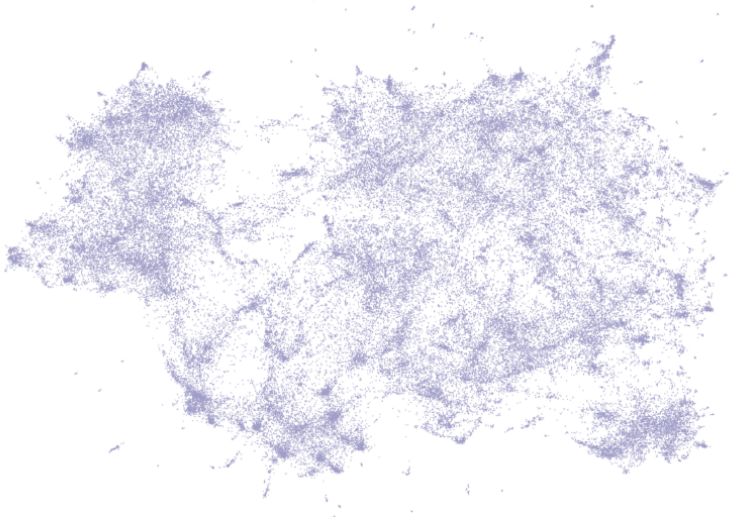


**Figure 4.14:** Top2Vec UMAP-reduced document vectors

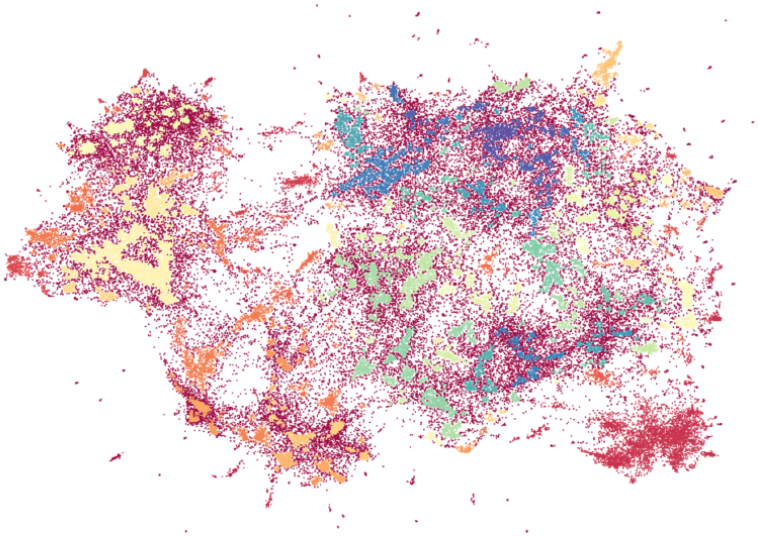


**Figure 4.15:** Top2Vec dense areas of documents identified by HDBSCAN

### Calculate Topic Vectors

Hence, HDBSCAN, on the space reduced by UMAP, identifies the dense cluster of documents by labeling each document in the semantic embedding space as noise or with the label of the dense cluster it belongs. After assigning the label to each cluster, the topic vectors are computed. Among the possible ways, they are obtained by taking the centroid as the arithmetic mean of all document vectors in the same cluster. Figure 4.16 shows a visual example of a topic vector being

calculated from a dense area of documents.

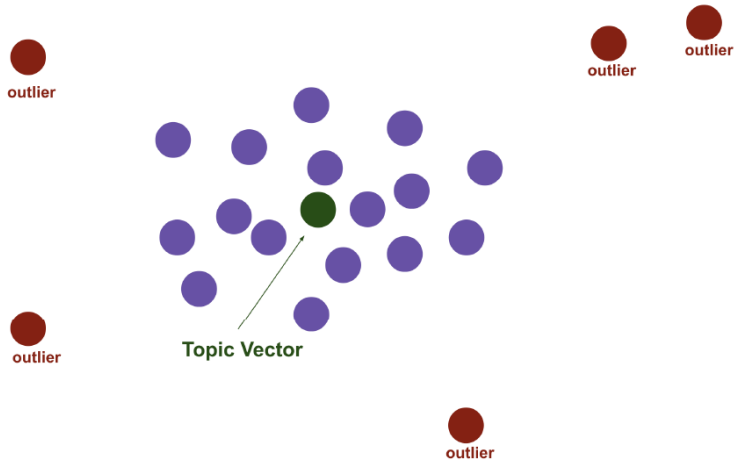



**Figure 4.16:** The topic vector is the centroid of the dense are of documents identified by HDBSCAN, which are the purple points. The outliers identified by HDBSCAN are not used to calculate the centroid.

As already described, in the semantic space a topic is described by its nearest word vectors, where the closest to the topic vector are the ones that represent it semantically and their similarity with respect to the topic vector is represented by the distance of each word vector to the topic vector. So, the most closest words to the topic vector are the most similar to all documents into that area, since the topic vector represents the centroid, and so they summarize the common topic of those document. Common words appear in almost all documents and, as consequence, they often are in a semantic space region equally distant from all documents. As result, the closest words to a topic vector will rarely be stop-words, so making not necessary the stop-word removal. Figure 4.17 shows an example of a topic vector and the nearest words.

### Topic Size and Hierarchical Topic Reduction

The topic and document vectors can be used to compute the size of topics. Document vectors are partitioned by the means of the topic vectors, such that each of them belongs only to the closest topic vector. By doing this association of documents clusters, the size of each topic is computed as the number of documents that belongs to it. Thanks to the representation of topics in the semantic space, there is the huge advantage to reduce the topics to a smaller value, with respect to the one automatically found. This is done by merging the smallest topic by size and its nearest topic vector for semantic meaning with an iterative approach, where the

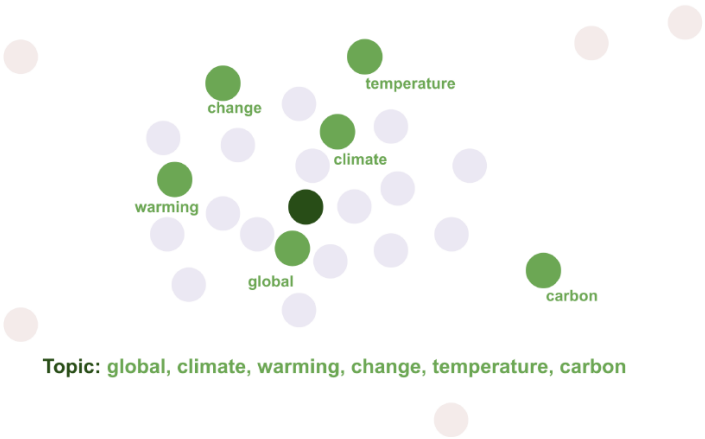



**Figure 4.17:** The topic words are the nearest word vectors to the topic vector.

arithmetic mean, weighted by the topic size, is taken. When a merge is done, the topic sizes are recomputed. Overall, this approach finds the most representative topics of the corpus, as it biases topics with greater size.

### BERTopic

From the idea proposed in Top2Vec, a slightly different algorithm called BERTopic [47] has been developed, based on BERT and transformers embeddings. The main covered steps are the same as in Top2Vec, however two major differences are introduced. The first one is that the layer of Doc2Vec embeddings is replaced: documents are now converted by using BERT, as it extracts different embeddings based on the context of the word. In particular, sentence-BERT is used, since the resulting embeddings have shown to be of high quality and typically they work quite well for document-level embeddings. The next steps are the same done on Top2Vec, with the dimensionality reduction applied with UMAP and the clusters created by the means of HDBSCAN. When the topic creation comes, here we find the second difference. To derive topics from clustered documents, a class-based variant of TF-IDF (c-TF-IDF) has been used. The intuition behind is the same of the classical TF-IDF, where the importance of words between documents are compared, but all documents in a single category are treated as a single document here. It results in a very long document per category, and the resulting TF-IDF would demonstrate the important words in a topic. It also allows to implement the same posterior topic reduction on the output of HDBSCAN, indeed by comparing the c-TF-IDF vectors among topics, the most similar ones are merged, and finally the c-TF-IDF vectors are re-calculated to update the representation of topics.

### 4.2.3 GEAC

At this point, after all introductions on many important models of the literature, I try to explore and to propose my custom implementation. Until now, methods based on probability or on neural network have been studied and proposed: starting from them, I work on making a model that tries to absorb the most relevant characteristics found on them and, at the same time, that tries to overcome all the issues seen both on theory and on experiment results.

This idea is expressed by the name of the model, GensimEncodingAutoencoder-Clustering (GEAC). This name comes from the utilization inside the model of the Gensim library [48], while the Encoding is used to extract features, moving from textual data to numerical data. Instead, the Autoencoder and the Clustering are used to create meaningfully topics, by first reducing the data dimension and finally by applying a cluster algorithm.

More in detail, the basic idea is to use both the probabilistic output produced by Gensim model and the features produced by a word embedding model. Indeed, classical algorithms, based on document representation like Bag-of-Word, can be used to create topics in situations where texts are large and words inside are coherent concerning the topics. So, if words are incoherent, due to the multiple themes addressed or for the low quantity of input data, some extra information can be used to understand the meaning of texts. Another reason lies on the datasets used in this project: they are difficult to handle properly due to their a-typical structure, with both few sentences and few words. Considering the major drawbacks of both classical and modern models, I decide to do a further step by considering them not as separate parts, but trying to connecting them. This is done by the taking into account many different model outputs and by concatenating them to do further operations.

In practice, the probability outputs of LDA or NMF, the word representation coming from BERT and the word vectors produced by Word2Vec are taken. Then, by combining them, I am able to enhance the output of the Gensim model by contextualizing it with the information coming from word embedding. About this concatenation, the three variables delta, epsilon and gamma come into play to weight each of the three components, allowing to create models with different concatenations. This is done to balance the three part to fine-tuning the model.

After the creation of the global vector as the combination of the three distinct ones, it is relevant to reduce the high dimensionality of the final vector that otherwise can create some issues related to the curse of dimensionality. Given the experience with UMAP, although it is a really powerful tool to reduce data dimension, from the implementation of BERTopic comes clear that it is too dependant on the several hyperparameters needed to build it properly. For this reason and to avoid a too expensive fine-tuning in terms of time and resources, I decide to take inspiration

from the idea proposed in ProdLDA to use an AutoEncoder. Indeed, for the dimensionality reduction I choose an AutoEncoder, a standard one in my case rather than a Variational. In this way the focus is only on its dimension, making the whole train way faster.

Following the implementation of a clustering algorithm, on Top2Vec/BERTopic there are some drawbacks related to HDBSCAN, so I decide to explore this step with other two algorithm, K-Means [49] and K-Mededoids [50]. K-Means is one of the most famous algorithm that use the Euclidean distance to compute the closeness between points and cluster. K-Medoids is a similar algorithm to K-Means but, rather than using as the centroid a point which could be not real, it uses the medoid, a real point of the dataset. Thanks to this consideration, K-Medoids is more robust to outliers and, due to the possibility to cover the whole spatial representation of points with different metrics, it is able to achieve better results, depending on the situation.
Overall, both of them find cluster by grouping together vectors that are close to each other, creating, as consequence, topics made by the the words inside the corresponding sentences. Those words and so the vectors are expected to have similar semantic meaning due to the transformation applied to vectorize the words.

Some strict assumptions related to HDBSCAN led me to move towards other clustering algorithms. Indeed, even if it is a really powerful algorithm that, enhancing DBSCAN, is able to explore the hierarchy between topics, in order to change the granularity of detail, the necessity of having at least two points in each cluster and the difficult management of outliers in a context where all sentences are relevant, make it really difficult to use in my specific contex. So, although same possible solution can be explored as we will see later, more practical cluster algorithms are preferable.

With such pipeline I face a problem, indeed growing the number of topics, the model will make independent cluster associated to each single sentence, causing an higher score that I will explain in the next section. In general, it happens since making 1:1 association is for clear reasons a type of division that works in all cases. Nevertheless, making many separate clusters does not take into consideration the meanings of each topic and of the words that compose it. If there are sentences about the same topic, but with slightly different words, under a human point of view it is straightforward to group them together. To reproduce this reasoning and, as consequence, to obtain the desired model, it is implemented a pruning. More in detail, it is taken as the best model the one that achieves the best score but also that ensures that the topic clusters do not exceed a certain level of similarity, fixed by the user. More detailed information about the implementation will be provided later but, in few words, this method allows to avoid to keep separate topics that are similar among themselves.

# Chapter 5

# Results

In this section I am going to analyse and to discuss the output of each model described in Chapter 4, showing how many issues are now overtaken with newer approaches. In detail, about these issues, an analysis to understand the goodness of each topic is performed. Indeed, a common problem with Topic Models regards the quality, since the cited models do not always guarantee independence between topic clusters. For this reason, I perform an analysis about the similarity between topics by exploiting the spatial representation of each word inside a topic through Feature Embedding. Inside this chapter, the used score metric is presented and explained; it is an important point inside the world of Topic Modeling, since using a score able to express a judgement in line with a possible valuation done by a human is essential.

## 5.1 Coherence Score

Dealing with Topic Modeling, different possible metrics exist to evaluate the models with their corresponding outputs. The perplexity measure is considered by many being the starting point. Perplexity is an intrinsic evaluation metric, widely used for language model evaluation. It captures the behaviour of a model when it faces new data and it is measured as the normalized log-likelihood. However, recent studies [51] have shown that perplexity and human judgment are always not correlated, but rather sometimes even slightly anti-correlated. This limitation motivated many works to model the human judgment, arriving at Topic Coherence. Topic Coherence measures [52] score a single topic by measuring the level of semantic similarity between words in the topic. These measurements help to distinguish between meaningful topics and topics that are artifacts [53]. Regarding the meaning of coherence, a set of sentences is coherent if they support each other, thus, sharing similar themes. An example of a coherent fact set is “pizza is an italian food”,

“pizza comes from Naples”, "spaghetti and meatballs is an American food”, where they share a thematic, in this case food.

Many coherence measures exist, presented over the years. Here a short list with the principal ones:

- $C_v$ measure is based on a sliding window, one-set segmentation of the top words and an indirect confirmation measure that uses normalized point-wise mutual information (NPMI) and the cosine similarity [54];
- $C_p$ is based on a sliding window, one-preceding segmentation of the top words and the confirmation measure of Fitelson's coherence; [55][54];
- $C_{uci}$ measure is based on a sliding window and the point-wise mutual information (PMI) of all word pairs of the given top words [52];
- $C_{umas}$ is based on document co-occurrence counts, a one-preceding segmentation and a logarithmic conditional probability as confirmation measure [56];
- $C_{npmi}$ is an enhanced version of the $C_{uci}$ coherence using the normalized point-wise mutual information (NPMI) [57];
- $C_a$ is based on a context window, a pairwise comparison of the top words and an indirect confirmation measure that uses normalized point-wise mutual information (NPMI) and the cosine similarity[54].

Among them, today $C_v$ is the most widely used, since it provides results close to the human ones. For this reason it is main metric used inside this project. Before moving forwards, here a quick explaination.

Roder et all [54] proposed a framework of coherence measures made by four parts, where it can be constructed existing measures as well as unexplored ones.

The first part is about the Segmentation of word subsets. Following [58], coherence of a word set measures the support level between two subsets. The segmentation of a word set $W$ produces a set of pairs of subsets of $W$. The definition of a subset pair consists in two parts, where the first one is the subset for which the support by the second part of the pair is determined when computing the confirmation measures. Different pair typology exist, like made of single words with UCI coherence or where one or both subsets contain more words.

Follows the part of probability estimation, where the authors moved from Boolean document, which estimates the probability of words as the number of documents in which they occur divided by the total number of documents, to a Boolean sliding window. It determines word count using a sliding window that

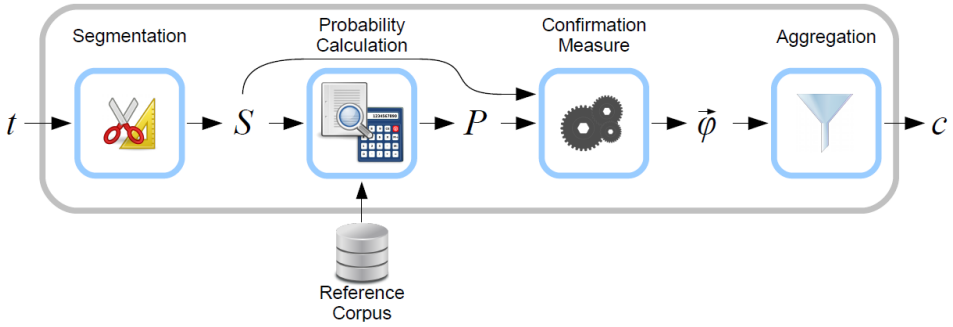



**Figure 5.1:** Overview over the unifying coherence framework - its four parts and their intermediate results

moves over the documents one word token per step, where each step defines a new virtual document.

A confirmation measure takes a single pair $S = (W', W^*)$ of words or word subsets as well as the corresponding probability to compute how strong the conditioning word set $W^*$ support $W'$. Among the possible different confirmation measures, they follow [57], where the largest correlation to human topic coherence ratings were found when the elements of the vectors are normalized PMI (NMPI).

$$m_{nlr}(S_i) = \frac{log\left(\frac{P(W',W^*)+\epsilon}{P(W')*P(W^*)}\right)}{-log(P(W',W^*)+\epsilon)} \tag{5.1}$$

The last step is made by the Aggregation, indeed all confirmations $\phi$ of all subset pairs $S_i$ are aggregated to a single coherence score by taking their arithmetic mean. The final score is inside a range between 0 and 1, whit acceptable values that are above 0.6/0.7 usually. In the results table below, the scores are reported with the confidence interval computed on 10 run with a confidence level of 0.95.

## 5.2 Topic Similarity

Topic modeling have several issues, some of them already mentioned. Among these issues, an important one is the quality of the created topic. Since the number of topics is not known in advance, extensive searches are necessary. For this project, due to the nature of the datasets, the most logical approach is to propose a level of search that goes from the distinct number of sources of the logs, until the number of unique logs provided as inputs to models. This is the consequence of the type of text analysed because, differently from long documents, where each one could be made by different topics, with logs we have short texts that makes explicit a

single operation done inside the system. This operation is strictly linked to a topic, underlined also by the fact that log texts are not long and so the few used words convey the message directly.

From these reasonings, if the number of topics on which the model is trained is higher, with respect to the real number of topics inside a corpus, the topic clusters may overlap each other. This would make the assigned label to each sentence less significant, since very similar things would be considered different. The same issues happens even if the number of topics is lower compared to the real number of topics, since the topic cluster composition depends on the model used. In particular, this behaviour is quite frequent on classical algorithms, where I ascertain that they have many issues with short sentences.

So, it is important to obtain a model where the topics do not overlap and, to guarantee it, I move the focus on Topic Similarity. At the same time, the idea is quite simple but powerful, after I created the topic clusters, I look for the words inside each of them and I check the similarity between them. In order to implement this check, I exploit the spatial representation of each word inside a topic through Feature Embedding. In detail, I have opted for a Word Embedding approach, based on a Word2Vec skip-gram model. As anticipated, in this project I deal with a very short corpus, so training and fine-tuning a feature embedding model from scratch would make the train too expensive. For these reason, I took the pre-trained model coming from Google [33] and I transfer its learning on a new model that I do further train to represent words that are not inside Google model. Provided this model, the words of all the topics are vectorized. Then, by taking topic clusters in pairs, I look for the similarity. Among the different measures, I opt for the cosine similarity, since it measures the similarity between two vectors of an inner product space since it considers not only the distance between point, but also the angle between them.

$$sim(x, y) = \frac{x \cdot y}{||x||||y||} \tag{5.2}$$

In this implementation the similarity measures between topics are obtained by taking the cosine similarity of the average vectorial representation of the words vector coming from Word2Vec. Then, if pairs have similarity above a provided threshold, I consider that topic model not optimal. Hence, it is really important to choose the right threshold, considering it is quite human dependant. Different values can be more appropriate depending on the input sentences. In analysed cases in this project, a good value have been found in 85% of similarity, since it is able to distinguish topics that are close but, at the same time, different for few details, like actions or subjects of those actions. Higher values have been tested but, in general, they result in too similar topics, while lower values make too strict the cut, without allowing topics with words in common. Given the similarity of the topics, I focus

on obtaining a model without similar clusters. So, on the topic models training phase, I look for a model whose combination of parameters guarantees it has both a high metric score and it is also optimal on the base of what a supervisor could infer, by looking directly at each cluster.

The choice about the best model is done on three different analysis:

1. The first one consists on decreasing the number of topic, depending on how many are similar, with a greedy approach, so with this new value I look recursively for the best model. For example, If the best model has 20 topics and 3 of them are correlated, by removing 2 topics I look for a model that could not have any correlated topics. If it still has similar topics, I recursively decrease the actual number of topics, depending on the number of similar topics, until I obtain an acceptable model;

2. The second option, instead, tries to exploit the metric score used by finding the model with the highest score which does not contain similar topics;

3. Finally, it is done an analysis by merging the similar topic between themselves. The idea is that if two or more topics are very similar between themselves, we can take them together. So, all the sentences that should be labelled differently with those topics labels, they will be label in the same way;

All these operations have been performed on the classical model, trying to compensate some poor generalizations and results. On the Topic Models based on Neural Network, instead, the second option inside the training pipeline has been directly implemented, since required. In this way, the best output model provides directly not too similar topics.

By anticipating the behaviours, for the first two implementations sometimes I obtain similar or a bit lower score results, but more meaningful for the supervisor. Indeed, the coherence metric normally is fine to obtain a general comprehension of the model, but, at the same time, in more precise situations the supervision of a user is required. Instead, with the last one I obtain many time better results, and it happens since the merge operation is done by taking the word in common and then by choosing among the remaining the ones that are more similar to ones already considered. However, the goodness depends also on the initial number of topics, indeed, if the number is very high and over the half of topic clusters are correlated, this operation would not provide good results in case of the real number of topics is high.

To represent this concept of similarity, in the results tables below I directly show the number of topics to be dropped in order to avoid similar topics, preventing to represent all the groups of similar topics. This value refers to the mean of the number of topics to drop when computing the Confidence Intervals for the

coherence score; I explicitly avoid to compute the confidence interval for this value since otherwise the computations would have been too expensive due to the higher number of trial for the similarity analysis.

## 5.3 Results

In this section the results produced by the different algorithms are presented. Each of them have been trained exploring all the hyper-parameter on different versions of the datasets. In details, as anticipated on Chapter 3, for each source (Spark and HDFS) different versions of the dataset are made: ORIGIN+MESSAGE and MESSAGE.

Besides them, during the hundreds of trials operated, it has been found that the absence of the ORIGIN has a negative impact on the classical algorithms and, at the same time, the ORIGIN can act as noise, producing models with too generic topic clusters. Indeed, if many logs have the same source origin, the classical models will be prone to cluster together those sentences even if the message is different. For this reasons, it has been also tried a modified version of the datasets, based on ORIGIN+MESSAGE. It is called SPLIT and it tries to reduce the impact of the ORIGIN part by taking only the monogram, avoiding to reduce the dataset to the same version of MESSAGE, that has fewer words. This additional version is an insight of possible considerations that could be implemented in future to better balance the dataset and to produce, as consequence, better topic models.

Another important consideration must be done on the metric. Indeed, by default, the Topic Modelling Gensim library [48], used in this project, computes the coherence on the top 20 words. By thinking on common text types, this is considered a good trade-off, however due to the nature of my data, in some cases the coherences computed on the top 10 and 15 words for each topic clusters are also needed. This reasoning is crucial since, on the outputs produced by the algorithms, the first words have more strength inside the topic, while the following are less relevant. So, the expected behaviour is the coherence on the top 10 words will be higher than the coherence on the top 15. As consequence, this last one will be higher than the one on the top 20. By applying this triple confirmations, it is possible to obtain a model that provides useful outputs. During the experiments, it has been found essential since, without doing this operation, in some cases it is possible to obtain a model with an high coherence on the top 20 words, which is not linked with higher score on the top 10 and 15 words. This happens since the coherence can provide good score when the output is a mixture of words that matches many input sentences. Hence, the topic clusters can be seen as a mix of different word that are good for different sentences.

The results are proposed both on terms of coherence and on predictions. Regarding the predictions, for both Spark and HDFS a subset of log that represents different possible cases and behaviour have been chosen, aiming to cover all situations. To better understand predictions quality, I have used a color schema, that goes from red, which indicates wrong predictions, to yellow, the output neither bad or good, and to green, which encodes a good label. The shades in the middle of these three colors are used to indicate intermediate labels. About this color mapping, it must be remember that the associations are handmade after a judgment based on the experiences coming from several trials. So, these links can not be perfect and subjected to the readers.

About the datasets, all the models have been trained on the output coming from the data processing phase, while, regarding the parameters, the range for the number of topics goes from the number of distinct sources, respectively 10 for Spark and 12 for HDFS, to the number of distinct rows in input. I remember that, even if log files are made by thousands or millions of lines, they can be restricted to hundreds or even less lines by considering the shared structures as already explained. In this case, for Spark the maximum number of lines is 45, while for HDFS is 99. With respect to the results, coherences and predictions come from the best model without similar topics for the neural network based models, while from the best model for classical algorithms, aiming to understand the similarities of topics.

Concerning the implementations, the classical models have been implemented by using the Gensim library [48], one of the best source available today for document representation. The neural network models have been implemented by means of different libraries. ProdLDA has been developed by starting from OCTIS [59], a recent library focused on Topic Modeling. BERTopic [47], instead, has been proposed starting from the code made available from the author on his GitHub page. Finally, my custom model GEAC has been based on BERT [35], Gensim [48] and Word2Vec [33] for the feature creation, while the AutoEncoder has been developed by the means of Keras [60]. For final clustering phase, K-Means comes from the scikit-learn libreary [61] and K-Medoids from scikit-learn-extra [62], a library focused on implementing interesting models still not included in the standard library. Regarding the machines, for the classical models the cluster provided by Smart Data PoliTO [63] in the maximal configuration (up to 48 CPU cores and 120 GB of RAM) is used, instead the models based on neural network have been trained on a Nvidia Tesla P100 provided by the Google Colaboratory service [64][65].

### 5.3.1 LDA

LDA, as the other classical Topic Modelling algorithms, takes as input a corpus, structured as a Bag of words, and the corresponding dictionary. The hyperparameters needed to fine-tune a model are $K$, the number of topics, $\alpha$, the

prior belief on the document-topic distribution, and $\beta$, the prior belief on topic-word distribution. For $K$, i.e the number of topics, the select range is the one presented above. The ranges for $\alpha$ and $\beta$ are the almost the same, the share part is $[0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, symmetric, auto]$, in addition for $\alpha$ is added *asymmetric*. About these three non-numeric values, *symmetric* and *asymmetric* retrieves the value from the number of topics, respectively they are $\frac{1}{K}$ and $\frac{1}{topicindex+\sqrt{K}}$. *Auto* instead tries to learn an asymmetric prior from the corpus.

### Spark

As we can see from Table 5.1, the highest score during training is obtained with a number of topics of 31, that resembles a possible human evaluation. Indeed, since I start with a very restricted number of distinct entries, I do not expect to obtain a number of topics too far from the number of input sentences. However, the model has a very high number of similar topics. By applying the three similarity analysis to overcome this issue, it is possible to obtain a better value when the correlated topics can be considered as ones.

| Results - Spark - ORIGIN+MESSAGE | | | |
|---|---|---|---|
| Result Type | Hyperparameters | Coherence | To drop |
| Best Coherence Model | K=31, $\alpha = 0.4, \beta = 0.6$ | 0.6953±0.0351 | 13 |
| Best Model Analysis 1 | K=18, $\alpha = 0.05, \beta = 0.7$ | 0.5805±0.0228 | 0 |
| Best Model Analysis 2 | K=12, $\alpha = 0.9, \beta = 0.5$ | 0.6575±0.0261 | 0 |
| Best Model Analysis 3 | K=18, $\alpha = /, \beta = /$ | 0.7262±0.0278 | 0 |

**Table 5.1:** LDA Results on ORIGIN+MESSAGE Spark dataset

The best model without similar topics is obtained with 18 topics, causing, as consequence, the production of topic clusters that do not match properly the sentences. The other two analysis, instead, do not produce any improvements. On the predictions we can see how the best model is able to cluster similar logs, however, some of them, especially the ones with few rare words, are not labelled correctly. The other observable behaviour has been already anticipated, i.e. the origin causes the collapsing of different logs in a topic cluster defined by the words inside the ORIGIN part of the logs.

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 0 | broadcast, broadcast_torrent, broadcast_read, start, variable, read, read_broadcast_variable, torrent, broadcast_variable, executor_start | broadcast.TorrentBroadcast: Reading broadcast variable 9 took 73 ms | broadcast, torrent, broadcast, read, broadcast, variable, take, broadcast_torrent, broadcast_torrent, broadcast_read, broadcast_read, broadcast_variable, variable_take, torrent_broadcast_read, read_broadcast_variable, broadcast_variable_take |
| 0 | broadcast, broadcast_torrent, broadcast_read, start, variable, read, read_broadcast_variable, torrent, broadcast_variable, executor_start | broadcast.TorrentBroadcast: Started reading broadcast variable 9 | broadcast, torrent, broadcast, start, read, broadcast, variable, broadcast_torrent, broadcast_torrent, broadcast_start, start_read, broadcast_read, broadcast_variable, torrent_broadcast_start, broadcast_start_read, start_read_broadcast, read_broadcast_variable |
| 24 | find, partition_not, cache, compute, not_find, not, partition_not_find, manager_partition, not_find_compute, cache_manager | spark.CacheManager: Partition rdd_2_6 not found, computing it | cache, partition, not, find, compute, cache_manager, manager_partition, partition_not, not_find, find_compute, cache_manager_partition, manager_partition_not, partition_not_find, not_find_compute |
| 11 | store, memory, memory_store, store_block, storage_memory_store, storage_memory, memory_store_block, store_value_memory, store_value, block_store_value | storage.MemoryStore: Block broadcast_9_piece0 stored as bytes in memory (estimated size 5.2 KB, free 5.2 KB) | memory, store, store, byte, memory, storage_memory, memory_store, store_block, store_block, store_byte, byte_memory, storage_memory_store, memory_store_block, block_store_byte, store_byte_memory |
| 11 | store, memory, memory_store, store_block, storage_memory_store, storage_memory, memory_store_block, store_value_memory, store_value, block_store_value | storage.MemoryStore: Block broadcast_9 stored as values in memory (estimated size 8.8 KB, free 13.9 KB) | memory, store, store, value, memory, storage_memory, memory_store, store_block, store_block, store_value, value_memory, storage_memory_store, memory_store_block, block_store_value, store_value_memory |
| 11 | store, memory, memory_store, store_block, storage_memory_store, storage_memory, memory_store_block, store_value_memory, store_value, block_store_value | storage.MemoryStore: MemoryStore cleared | memory, store, memory, store, clear, storage_memory, memory_store, memory_store, memory_store, store_clear, storage_memory_store, memory_store_clear |
| 27 | deprecate, task, deprecate_mapred, mapred, use_mapreduce, configuration_deprecate_mapred, deprecate_use_mapreduce, deprecate_use configuration_deprecate, use | mapred.SparkHadoopMapRedUtil: attempt_201706091116_ 0204_m_000033_8190: Committed | mapred, hadoop, mapred, util, commit, mapred_hadoop, mapred_hadoop, mapred_util, util_commit, hadoop_mapred_util, mapred_util_commit |
| 0 | broadcast, broadcast_torrent, broadcast_read, start, variable, read, read_broadcast_variable, torrent, broadcast_variable, executor_start | slf4j.Slf4jLogger: Slf4jLogger started | logger, start, logger_start |

**Figure 5.2:** Predictions LDA on Spark - ORIGIN+MESSAGE

In the case of SPLIT dataset, the behaviour is different for the best model. Rather than providing a model with a big value for $K$ and many similar topics, in this situation the best model provides a smaller value for $K$ of 17, with 2 similar topics. The outputs coming the three analysis are instead in line with the previous situation, so better only for the third analysis. An important difference can be appreciated with the outputs. In this version of the dataset, I have less words inside each sentence, increasing the difficulties of LDA when it deals with small datasets [66][67]. Indeed, many predictions are comparable, however, some of them are now labeled incorrectly sometimes. In addition this observation, it is also relevant to see that the topic clusters are less clean than before, with less coherent words. This behaviour is also underlined by the values of the coherence with top 10 and top 15, they are lower or similar than the one with the top 20 words.

| Results - Spark - SPLIT | | | |
|---|---|---|---|
| Result Type | Hyperparameters | Coherence | To drop |
| Best Coherence model | K=17, $\alpha = 0.4, \beta = 1$ | 0.6948±0.0236 | 2 |
| Best Model Analysis 1 | K=15, $\alpha = 1, \beta = 0.01$ | 0.6121±0.0257 | 0 |
| Best Model Analysis 2 | K=15, $\alpha = 1, \beta = 0.01$ | 0.6121±0.0257 | 0 |
| Best Model Analysis 3 | K=15, $\alpha = /, \beta = /$ | 0.7238±0.0226 | 0 |

**Table 5.2:** LDA Results on SPLIT Spark dataset

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 0 | memory, store, coarse, grain, backend, broadcast, start, driver, output, task | broadcast.TorrentBroadcast: Reading broadcast variable 9 took 73 ms | broadcast, torrent, broadcast, read, broadcast, variable, take, read_broadcast, broadcast_variable, variable_take, read_broadcast_variable, broadcast_variable_take |
| 0 | memory, store, coarse, grain, backend, broadcast, start, driver, output, task | broadcast.TorrentBroadcast: Started reading broadcast variable 9 | broadcast, torrent, broadcast, start, read, broadcast, variable, start_read, read_broadcast, broadcast_variable, start_read_broadcast, read_broadcast_variable |
| 6 | store, memory, grain, backend, coarse, start, broadcast, driver, output, shutdown | spark.CacheManager: Partition rdd_2_6 not found, computing it | cache, partition, not, find, compute, partition_not, not_find, find_compute, partition_not_find, not_find_compute |
| 15 | memory, store, coarse, backend, grain, start, output, driver, task, broadcast | storage.MemoryStore: Block broadcast_9_piece0 stored as bytes in memory (estimated size 5.2 KB, free 5.2 KB) | memory, store, store, byte, memory, block_store, store_byte, byte_memory, block_store_byte, store_byte_memory |
| 15 | memory, store, coarse, backend, grain, start, output, driver, task, broadcast | storage.MemoryStore: Block broadcast_9 stored as values in memory (estimated size 8.8 KB, free 13.9 KB) | memory, store, store, value, memory, block_store, store_value, value_memory, block_store_value, store_value_memory |
| 15 | memory, store, coarse, backend, grain, start, output, driver, task, broadcast | storage.MemoryStore: MemoryStore cleared | memory, store, memory, store, clear, memory_store, store_clear, memory_store_clear |
| 3 | deprecate, task, mapred, use, deprecate_use_mapreduce, deprecate_use configuration, use_mapreduce, mapreduce, use_mapreduce_task | mapred.SparkHadoopMapRedUtil: attempt_201706091116_ 0204_m_000033_8190: Committed | mapred, hadoop, mapred, util, commit |
| 5 | remote, shutdown, provider, remote_daemon, actor, terminator, daemon, start, proceed_flush_remote, flush_remote | slf4j.Slf4jLogger: Slf4jLogger started | logger, start, logger_start |

**Figure 5.3:** Predictions LDA on Spark - SPLIT

In the dataset composed by only messages, we can see how the results are in general lower compared to previous values. This result is expected, since the already mentioned problem in the SPLIT dataset is further increased, due to the fewer words in each line. This behaviour is recurrent for all the classical models, for this

reason further considerations and comparisons with the models based on Neural Network will be relevant. For what concern the predictions, the produced clusters contain words coming from different thematics. So, even if the prediction is almost acceptable, it can been directly seen how the meaning is not easily understandable.

| Results - Spark - MESSAGE | | | |
|---|---|---|---|
| Result Type | Hyperparameters | Coherence | To drop |
| Best Coherence model | K=43, $\alpha = 0.3, \beta = 0.5$ | 0.6582±0.0097 | 27 |
| Best Model Analysis 1 | K=16, $\alpha = 1, \beta = 0.01$ | 0.5715±0.0243 | 0 |
| Best Model Analysis 2 | K=40, $\alpha = 1, \beta = 0.1$ | 0.6382±0.0127 | 0 |
| Best Model Analysis 3 | K=16, $\alpha = /, \beta = /$ | 0.5450±0.0206 | 0 |

**Table 5.3:** LDA Results on MESSAGE Spark dataset

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 40 | create, directory, take, broadcast_variable, variable, read_broadcast_variable, broadcast, read_broadcast, broadcast_variable_take, variable_take | broadcast.TorrentBroadcast: Reading broadcast variable 9 took 73 ms | read, broadcast, variable, take, read_broadcast, broadcast_variable, variable_take, read_broadcast_variable, broadcast_variable_take |
| 14 | start, broadcast, read_broadcast_variable, read_broadcast, read, variable, broadcast_variable, start_read_broadcast, start_read, shutdown_hook | broadcast.TorrentBroadcast: Started reading broadcast variable 9 | start, read, broadcast, variable, start_read, read_broadcast, broadcast_variable, start_read_broadcast, read_broadcast_variable |
| 6 | start, remote_start, remote, find, listen_address, start_listen, start_listen_address, not_find_compute, address, partition_not | spark.CacheManager: Partition rdd_2_6 not found, computing it | partition, not, find, compute, partition_not, not_find, find_compute, partition_not_find, not_find_compute |
| 41 | byte, stage, task_stage, driver, finish, finish_task, stage_byte_result, result_send_driver, task_stage_byte, send_driver | storage.MemoryStore: Block broadcast_9_piece0 stored as bytes in memory (estimated size 5.2 KB, free 5.2 KB) | block, store, byte, memory, block_store, store_byte, byte_memory, block_store_byte, store_byte_memory |
| 9 | block, block_store, memory, store, block_store_value, store_value_memory, value_memory, value, store_value, rdd | storage.MemoryStore: Block broadcast_9 stored as values in memory (estimated size 8.8 KB, free 13.9 KB) | block, store, value, memory, block_store, store_value, value_memory, block_store_value, store_value_memory |
| 34 | remote, memory, store, memory_store, shutdown, remote_daemon, daemon, flush_remote, proceed_flush_remote, shutdown_proceed | storage.MemoryStore: MemoryStore cleared | memory, store, clear, memory_store, store_clear, memory_store_clear |
| 39 | task, partition, mapred_task, mapreduce_task, use_mapreduce_task, task_partition, mapreduce, mapred, use, deprecate | mapred.SparkHadoopMapRedUtil: attempt_201706091116_ 0204_m_000033_8190: Committed | attempt, commit, attempt_commit |
| 14 | start, broadcast, read_broadcast_variable, read_broadcast, read, variable, broadcast_variable, start_read_broadcast, start_read, shutdown_hook | slf4j.Slf4jLogger: Slf4jLogger started | logger, start, logger_start |

**Figure 5.4:** Predictions LDA on Spark - MESSAGE

### HDFS

The behaviour with HDFS is very similar to Spark. Indeed, the shortcomings are similar, but the results increase due to the higher quantity of input data. Nevertheless, the persistence presence of problems suggests that more advanced algorithms are needed to accomplish this task.
From Table 5.4, on ORIGIN+MESSAGE dataset the highest score during training is obtained with a number of topic equal to 91, a value acceptable with respect to a plausible human judgement. However, that model has a very high number of similar topics, as well as Spark. About the three similarity analysis, the trends reflect a similar behaviour as in table 5.1, but this time also the third analysis provides worse results. Instead, the first analysis produces a final output too low to be acceptable.

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 62 | exception, eof, java, io, io_eof_exception, java_io, io_eof, java_io_eof, eof_exception, host_destination | Caused by java.io.EOFException | cause, java, io, eof, exception, cause_java, java_io, io_eof, eof_exception, cause_java_io, java_io_eof, io_eof_exception |
| 64 | io, exception, host, java, io_exception, java_io, local, host_destination, exception_connection_reset, connection_reset_peer | Caused by java.io.IOException: Connection reset by peer | cause, java, io, io, exception, connection, reset, peer, cause_java, java_io, io_exception, exception_connection, connection_reset, reset_peer, cause_java_io, io_exception_connection, exception_connection_reset, connection_reset_peer |
| 62 | exception, eof, java, io, io_eof_exception, java_io, io_eof, java_io_eof, eof_exception, host_destination | java.io.EOFException | java, io, eof, exception, java_io, io_eof, eof_exception, java_io_eof, io_eof_exception |
| 14 | fsdataset_impl, fsdataset, impl, datanode_fsdataset, datanode_fsdataset_impl, add, pool, block_pool, volume, replica | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Total time to add all replicas to map: 0ms | fsdataset, impl, fsdataset, impl, total, time, add, replica, map, datanode_fsdataset, fsdataset_impl, fsdataset_impl, fsdataset_impl, impl_total, total_time, time_add, add_replica, replica_map, datanode_fsdataset_impl, fsdataset_impl_total, impl_total_time, total_time_add, time_add_replica, add_replica_map |
| 14 | fsdataset_impl, fsdataset, impl, datanode_fsdataset, datanode_fsdataset_impl, add, pool, block_pool, volume, replica | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Total time to scan all replicas for block pool BP-108841162-10.10.34.11-1440074360971: 8ms | fsdataset, impl, fsdataset, impl, total, time, scan, replica, pool, datanode_fsdataset, fsdataset_impl, fsdataset_impl, fsdataset_impl, impl_total, total_time, time_scan, scan_replica, replica_block, block_pool, datanode_fsdataset_impl, fsdataset_impl_total, impl_total_time, total_time_scan, time_scan_replica, scan_replica_block, replica_block_pool |
| 69 | http, filter, block_scanner, scanner, mortbay, user, http_add, filter_static, static_user, filter_context | org.mortbay.log: jetty-6.1.26 | mortbay |
| 69 | http, filter, block_scanner, scanner, mortbay, user, http_add, filter_static, static_user, filter_context | org.mortbay.log: Logging to org.slf4j.impl.Log4jLoggerAdapter (org.mortbay.log) via org.mortbay.log.Slf4jLog | mortbay, log, mortbay_log |

**Figure 5.5:** Predictions LDA on HDFS - ORIGIN+MESSAGE

| Results - HDFS - ORIGIN+MESSAGE | | | |
|---|---|---|---|
| Result Type | Hyperparameters | Coherence | To drop |
| Best Coherence model | K=91, $\alpha = 0.1, \beta = 0.6$ | 0.7109±0.0240 | 47 |
| Best Model Analysis 1 | K=44, $\alpha = 0.4, \beta = 0.5$ | 0.5872±0.0162 | 0 |
| Best Model Analysis 2 | K=22, $\alpha = 0.2, \beta = 1$ | 0.6996±0.0383 | 0 |
| Best Model Analysis 3 | K=42, $\alpha = /, \beta = /$ | 0.7974±0.0219 | 0 |

**Table 5.4:** LDA Results on ORIGIN+MESSAGE HDFS dataset

In the case of SPLIT dataset, the best model provides a lower value of $K$ and coherence score is higher with respect to the best model on ORIGIN+MESSAGE, however, there still are many similar topic, even if less than ORIGIN+MESSAGE, as for Spark. Contrary to the same implementation done on Spark, here the drawbacks are more visible since the final number of topics, obtained as best $K$ minus the number of topics to drop, is too low in proportion to the same results on Spark and with respect the possible real number of topics.
The outputs coming the three similarity analysis are in line with the previous situation, with the same reasoning done before and with lower results. Nevertheless, as can be appreciated from Figure 5.6, the predictions seem to be more reasonable compared to ORIGIN+MESSAGE. Indeed, for this case, the implementation to weight differently the words coming from the ORIGIN seem to pay off. An interesting observation can be done about the data processing implemented to extract meaningful words from raw sentences. In the $6^{TH}$ row can be appreciated how, since the presence of a single token, it is not discarded but rather kept after the removal of unnecessary characters. In this way, I can keep all input logs, even those that with standard processing would be discarded, since it is essential to keep all elements.

| Results - HDFS - SPLIT | | | |
|---|---|---|---|
| Result Type | Hyperparameters | Coherence | To drop |
| Best Coherence model | K=43, $\alpha = 0.4, \beta = 1$ | 0.7432±0.01887 | 25 |
| Best Model Analysis 1 | K=16, $\alpha = 0.2, \beta = 1$ | 0.6679±0.0288 | 0 |
| Best Model Analysis 2 | K=20, $\alpha = 0.3, \beta = 0.7$ | 0.6756±0.0419 | 0 |
| Best Model Analysis 3 | K=17, $\alpha = /, \beta = /$ | 0.6877±0.0286 | 0 |

**Table 5.5:** LDA Results on SPLIT HDFS dataset

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 7 | exception, eof, java, io, java_io, io_eof, io_eof_exception, java_io_eof, eof_exception, host_destination | Caused by java.io.EOFException | cause, java, io, eof, exception, java_io, io_eof, eof_exception, java_io_eof, io_eof_exception |
| 26 | fsdataset, impl, io, add, pool, block_pool, exception, volume, replica, time | Caused by java.io.IOException: Connection reset by peer | cause, java, io, io, exception, connection, reset, peer, java_io, io_exception, exception_connection, connection_reset, reset_peer, io_exception_connection, exception_connection_reset, connection_reset_peer |
| 7 | exception, eof, java, io, java_io, io_eof, io_eof_exception, java_io_eof, eof_exception, host_destination | java.io.EOFException | java, io, eof, exception, java_io, io_eof, eof_exception, java_io_eof, io_eof_exception |
| 26 | fsdataset, impl, io, add, pool, block_pool, exception, volume, replica, time | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDataset Impl: Total time to add all replicas to map: 0ms | fsdataset, impl, fsdataset, impl, total, time, add, replica, map, total_time, time_add, add_replica, replica_map, total_time_add, time_add_replica, add_replica_map |
| 26 | fsdataset, impl, io, add, pool, block_pool, exception, volume, replica, time | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Total time to scan all replicas for block pool BP-108841162-10.10.34.11-1440074360971: 8ms | fsdataset, impl, fsdataset, impl, total, time, scan, replica, pool, total_time, time_scan, scan_replica, replica_block, block_pool, total_time_scan, time_scan_replica, scan_replica_block, replica_block_pool |
| 37 | supergroup, mortbay, start, balance, msg, signal, receive, open, thread, transfer | org.mortbay.log: jetty-6.1.26 | mortbay |
| 15 | http, filter, static, add, user, add_filter_static, add_filter, filter_static_user, user_filter, static_user | org.mortbay.log: Logging to org.slf4j.impl.Log4jLoggerAdapter (org.mortbay.log) via org.mortbay.log.Slf4jLog | mortbay, log, mortbay_log |

**Figure 5.6:** Predictions LDA on HDFS - SPLIT

Considering the last dataset of messages, this time the results are higher than in the other two datasets and the number of topics found is acceptable, even by removing the similar topics. The applications of the three analysis provide good results, although they are not improved. Indeed, even if the coherence scores decrease with respect to the best model, they are still comparable with the best one. Also the number of topics is acceptable, excluded the second analysis which choose a model with too low number of topics.
The predictions seem to confirm this trend, allowing to match sentences with clusters that have same themes. The behaviour, similar to SPLIT, is to create topics made by words that humanly are not coherent, but that allows to provide a more reasonable label. Therefore, even if the assignments are better that in ORIGIN+MESSAGE, this result is still not ideal for a real case implementation.

Overall, LDA provides good results, in particular among classical algorithms. However, it has some drawbacks since it is not able to find all topics and it generalizes too much when in texts irrelevant but common words are found.

| Results - HDFS - MESSAGE | | | |
|---|---|---|---|
| Result Type | Hyperparameters | Coherence | To drop |
| Best Coherence model | K=98, $\alpha = 0.3, \beta = 1$ | 0.7837±0.0199 | 47 |
| Best Model Analysis 1 | K=78, $\alpha = 0.4, \beta = 0.4$ | 0.7248±0.0154 | 0 |
| Best Model Analysis 2 | K=52, $\alpha = 0.4, \beta = 0.7$ | 0.7533±0.0124 | 0 |
| Best Model Analysis 3 | K=78, $\alpha = /, \beta = /$ | 0.7723±0.0164 | 0 |

**Table 5.6:** LDA Results on MESSAGE HDFS dataset

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 20 | exception, host_destination, host, eof_exception, eof, io_eof_exception, java_io_eof, io_eof, java_io, io | java.io.EOFException | java, io, eof, exception, java_io, io_eof, eof_exception, java_io_eof, io_eof_exception |
| 88 | io, exception, host, io_exception, local, host_destination, java_io, java, connection_reset, connection_reset_peer | java.io.IOException: Connection reset by peer | java, io, io, exception, connection, reset, peer, java_io, io_exception, exception_connection, connection_reset, reset_peer, io_exception_connection, exception_connection_reset, connection_reset_peer |
| 66 | pool, block_pool, add, volume, replica, time, add_replica, map, add_replica_map, replica_map | Total time to add all replicas to map: 0ms | total, time, add, replica, map, total_time, time_add, add_replica, replica_map, total_time_add, time_add_replica, add_replica_map |
| 66 | pool, block_pool, add, volume, replica, time, add_replica, map, add_replica_map, replica_map | Total time to scan all replicas for block pool BP-108841162-10.10.34. 11- 1440074360971: 8ms | total, time, scan, replica, pool, total_time, time_scan, scan_replica, replica_block, block_pool, total_time_scan, time_scan_replica, scan_replica_block, replica_block_pool |
| 89 | file, supergroup, lock, balance, jetty, msg, schedule, server, open, format | jetty-6.1.26 | jetty |
| 75 | filter, user, log, add, add_filter, static_user_filter, context, filter_static_user, user_filter_context, filter_static | Logging to org.slf4j.impl.Log4jLoggerAdapter (org.mortbay.log) via org.mortbay.log.Slf4jLog | log |
| 30 | report, take, command, finalize_command, slow, msec, sent, report_sent, msec_generate, get | org.apache.hadoop.hdfs.server. datanode.DataNode: Slow manageWriterOsCache took 5299ms (threshold=300ms) | slow, manage, writer, os, cache, take, slow_manage, manage_writer, writer_os, os_cache, cache_take, slow_manage_writer, manage_writer_os, writer_os_cache, os_cache_take |

**Figure 5.7:** Predictions LDA on HDFS - MESSAGE

### 5.3.2 NMF

For this project NMF has been trained with several combinations of parameters. Differently from LDA, NMF accepts various transformations of the input data. To explore this possibility, I have tried both Bag-of-Words and Tf-Idf to transform data, however the best result has been achieved with the first proposal.

Provided these transformed data, the models have been fine-tuned by searching

both the number of topics $K$, inside the usual range, and *kappa*, the parameter that regulate the gradient descent step size.

### Spark

| Results - Spark - ORIGIN+MESSAGE | | | |
|---|---|---|---|
| Result Type | Hyperparameters | Coherence | To drop |
| Best Coherence Model | K=22, kappa=0.1 | 0.8072±0.0186 | 5 |
| Best Model Analysis 1 | K=16, kappa=0.1 | 0.7590±0.0216 | 0 |
| Best Model Analysis 2 | K=16, kappa=0.1 | 0.7590±0.0216 | 0 |
| Best Model Analysis 3 | K=17, kappa=/ | 0.8129±0.0453 | 0 |

**Table 5.7:** NMF Results on ORIGIN+MESSAGE Spark dataset

In the table regarding ORIGIN+MESSAGE, It can be noticed how NMF is able to provided higher coherences with respect to LDA. However, at the same time the number of topics is smaller. This behaviour happens for the same reasons already explained. Rather the providing many clusters good both for the origin and the message, due to the higher relevance of the words inside the origin, which they are more common if that origin occur many times, the algorithm will create a single cluster focused on the origin. At the same time, the lower values of $K$ are a sign of the poor generaliziotions done by the algorithm. These issues are perfectly represented in the picture about predictions where this behaviour is underlined by the rows in yellow. For the more difficult ones, like the last two, NMF is still not able to obtain the corresponding topic clusters. For what concerns the similarity analysis, only the third approach based on merging similar topics provides better results.

| Results - Spark - SPLIT | | | |
|---|---|---|---|
| Result Type | Hyperparameters | Coherence | To drop |
| Best Coherence model | K=14, kappa=0.05 | 0.7497±0.0196 | 0 |
| Best Model Analysis 1 | K=14, kappa=0.05 | 0.7497±0.0196 | 0 |
| Best Model Analysis 2 | K=14, kappa=0.05 | 0.7497±0.0196 | 0 |
| Best Model Analysis 3 | K=14, kappa=/ | 0.7497±0.0196 | 0 |

**Table 5.8:** NMF Results on SPLIT Spark dataset

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 11 | broadcast, broadcast_torrent, broadcast_read, variable, broadcast_variable, torrent, read_broadcast_variable, read, torrent_broadcast_start, broadcast_start | broadcast.TorrentBroadcast: Reading broadcast variable 9 took 73 ms | broadcast, torrent, broadcast, read, broadcast, variable, take, broadcast_torrent, broadcast_torrent, broadcast_read, broadcast_read, broadcast_variable, variable_take, torrent_broadcast_read, read_broadcast_variable, broadcast_variable_take |
| 11 | broadcast, broadcast_torrent, broadcast_read, variable, broadcast_variable, torrent, read_broadcast_variable, read, torrent_broadcast_start, broadcast_start | broadcast.TorrentBroadcast: Started reading broadcast variable 9 | broadcast, torrent, broadcast, start, read, broadcast, variable, broadcast_torrent, broadcast_torrent, broadcast_start, start_read, broadcast_read, broadcast_variable, torrent_broadcast_start, broadcast_start_read, start_read_broadcast, read_broadcast_variable |
| 14 | deprecate, partition, deprecate_mapred, task_partition, configuration_deprecate, configuration, use, deprecate_use, configuration_deprecate_mapred, use_mapreduce | spark.CacheManager: Partition rdd_2_6 not found, computing it | cache, partition, not, find, compute, cache_manager, manager_partition, partition_not, not_find, find_compute, cache_manager_partition, manager_partition_not, partition_not_find, not_find_compute |
| 16 | memory_store, memory, store, storage_memory_store, storage_memory, store_block, memory_store_block, start_capacity, store_start_capacity, memory_store_start | storage.MemoryStore: Block broadcast_9_piece0 stored as bytes in memory (estimated size 5.2 KB, free 5.2 KB) | memory, store, store, byte, memory, storage_memory, memory_store, store_block, store_block, store_byte, byte_memory, storage_memory_store, memory_store_block, block_store_byte, store_byte_memory |
| 16 | memory_store, memory, store, storage_memory_store, storage_memory, store_block, memory_store_block, start_capacity, store_start_capacity, memory_store_start | storage.MemoryStore: Block broadcast_9 stored as values in memory (estimated size 8.8 KB, free 13.9 KB) | memory, store, store, value, memory, storage_memory, memory_store, store_block, store_block, store_value, value_memory, storage_memory_store, memory_store_block, block_store_value, store_value_memory |
| 16 | memory_store, memory, store, storage_memory_store, storage_memory, store_block, memory_store_block, start_capacity, store_start_capacity, memory_store_start | storage.MemoryStore: MemoryStore cleared | memory, store, memory, store, clear, storage_memory, memory_store, memory_store, memory_store, store_clear, storage_memory_store, memory_store_clear |
| 6 | remote, remote_start, start, terminator_shutdown, shutdown_remote_daemon, terminator_shutdown_remote, remote_terminator_shutdown | mapred.SparkHadoopMapRedUtil: attempt_201706091116_0204_m_000033_8190: Committed | mapred, hadoop, mapred, util, commit, mapred_hadoop, mapred_hadoop, mapred_util, util_commit, hadoop_mapred_util, mapred_util_commit |
| 6 | remote, remote_start, start, terminator_shutdown, shutdown_remote_daemon, terminator_shutdown_remote, remote_terminator_shutdown | slf4j.Slf4jLogger: Slf4jLogger started | logger, start, logger_start |

**Figure 5.8:** Predictions NMF on Spark - ORIGIN+MESSAGE

The application on SPLIT is particular since it does not provide any similar topic. This is due to the lower starting value for the number of topics. Since the lower number of words, in this case the model is able to find good coherences without exploiting the whole space of possible clusters. This is a drawback coming from the combination of the algorithm with this modified version of the dataset. These reasonings are underlined by the predictions with poor associations and a general lower quality of the topic clusters.

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 4 | broadcast, memory, store, read_broadcast, torrent, read_broadcast_variable, broadcast_variable, read, variable, start | broadcast.TorrentBroadcast: Reading broadcast variable 9 took 73 ms | broadcast, torrent, broadcast, read, broadcast, variable, take, read_broadcast, broadcast_variable, variable_take, read_broadcast_variable, broadcast_variable_take |
| 4 | broadcast, memory, store, read_broadcast, torrent, read_broadcast_variable, broadcast_variable, read, variable, start | broadcast.TorrentBroadcast: Started reading broadcast variable 9 | broadcast, torrent, broadcast, start, read, broadcast, variable, start_read, read_broadcast, broadcast_variable, start_read_broadcast, read_broadcast_variable |
| 3 | shutdown, backend, coarse, grain, hook, util, driver, shutdown_hook_call, shutdown_hook, call | spark.CacheManager: Partition rdd_2_6 not found, computing it | cache, partition, not, find, compute, partition_not, not_find, find_compute, partition_not_find, not_find_compute |
| 0 | store, memory, memory_store, start, block_store, memory_store_start, capacity, start_capacity, store_start, store_start_capacity | storage.MemoryStore: Block broadcast_9_piece0 stored as bytes in memory (estimated size 5.2 KB, free 5.2 KB) | memory, store, store, byte, memory, block_store, store_byte, byte_memory, block_store_byte, store_byte_memory |
| 4 | broadcast, memory, store, read_broadcast, torrent, read_broadcast_variable, broadcast_variable, read, variable, start | storage.MemoryStore: Block broadcast_9 stored as values in memory (estimated size 8.8 KB, free 13.9 KB) | memory, store, store, value, memory, block_store, store_value, value_memory, block_store_value, store_value_memory |
| 4 | broadcast, memory, store, read_broadcast, torrent, read_broadcast_variable, broadcast_variable, read, variable, start | storage.MemoryStore: MemoryStore cleared | memory, store, memory, store, clear, memory_store, store_clear, memory_store_clear |
| 13 | deprecate, mapred, deprecate_use_mapreduce, use, mapreduce, deprecate_use, use_mapreduce, configuration, task, mapred_deprecate_use | mapred.SparkHadoopMapRedUtil: attempt_201706091116_0204_m_000033_8190: Committed | mapred, hadoop, mapred, util, commit |
| 4 | broadcast, memory, store, read_broadcast, torrent, read_broadcast_variable, broadcast_variable, read, variable, start | slf4j.Slf4jLogger: Slf4jLogger started | logger, start, logger_start |

**Figure 5.9:** Predictions NMF on Spark - SPLIT

| Results - Spark - MESSAGE | | | |
|---|---|---|---|
| Result Type | Hyperparameters | Coherence | To drop |
| Best Coherence model | K=39, kappa=0.01 | 0.7019±0.0231 | 23 |
| Best Model Analysis 1 | K=16, kappa=0.01 | 0.6256±0.0176 | 0 |
| Best Model Analysis 2 | K=16, kappa=0.01 | 0.6256±0.0176 | 0 |
| Best Model Analysis 3 | K=16, kappa= / | 0.5713±0.0183 | 0 |

**Table 5.9:** NMF Results on MESSAGE Spark dataset

For what concern MESSAGE dataset, the initial number of topics is high but linked to an high number of similar topics. However, the coherence for the best model is higher with respect to the one obtained with LDA. Differently from SPLIT, in this case the application of the different analysis can be done, however, as shown in the table, they do not provide better results.

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 34 | start, variable, broadcast_variable, read, read_broadcast_variable, broadcast, read_broadcast, start_read, take, broadcast_variable_take | broadcast.TorrentBroadcast: Reading broadcast variable 9 took 73 ms | read, broadcast, variable, take, read_broadcast, broadcast_variable, variable_take, read_broadcast_variable, broadcast_variable_take |
| 34 | start, variable, broadcast_variable, read, read_broadcast_variable, broadcast, read_broadcast, start_read, take, broadcast_variable_take | broadcast.TorrentBroadcast: Started reading broadcast variable 9 | start, read, broadcast, variable, start_read, read_broadcast, broadcast_variable, start_read_broadcast, read_broadcast_variable |
| 14 | finish, block, task, block_manager, partition, boot_init, byte, total, register_block, not_find | spark.CacheManager: Partition rdd_2_6 not found, computing it | partition, not, find, compute, partition_not, not_find, find_compute, partition_not_find, not_find_compute |
| 8 | block, store, memory, block_store, manager, block_manager, value, store_value, value_memory, block_store_value | storage.MemoryStore: Block broadcast_9_piece0 stored as bytes in memory (estimated size 5.2 KB, free 5.2 KB) | block, store, byte, memory, block_store, store_byte, byte_memory, block_store_byte, store_byte_memory |
| 8 | block, store, memory, block_store, manager, block_manager, value, store_value, value_memory, block_store_value | storage.MemoryStore: Block broadcast_9 stored as values in memory (estimated size 8.8 KB, free 13.9 KB) | block, store, value, memory, block_store, store_value, value_memory, block_store_value, store_value_memory |
| 35 | remote, start, memory_store, daemon, remote_daemon, remote_start, shutdown, user, shutdown_proceed_flush, proceed_flush_remote | storage.MemoryStore: MemoryStore cleared | memory, store, clear, memory_store, store_clear, memory_store_clear |
| 30 | task, mapreduce_task, use_mapreduce_task, mapred_task, use_mapreduce, mapreduce, deprecate, mapred, deprecate_use, deprecate_use_mapreduce | mapred.SparkHadoopMapRedUtil: attempt_201706091116_ 0204_m_000033_8190: Committed | attempt, commit, attempt_commit |
| 34 | start, variable, broadcast_variable, read, read_broadcast_variable, broadcast, read_broadcast, start_read, take, broadcast_variable_take | slf4j.Slf4jLogger: Slf4jLogger started | logger, start, logger_start |

**Figure 5.10:** Predictions NMF on Spark - MESSAGE

From Picture 5.10 we can see how the results are indeed better than in the analogous case of LDA. Regarding the issues, they are the same as in ORIGIN+MESSAGE; The models create fuzzy cluster in some cases, while in the other, mainly link with short logs, it is not able to find a group.

### HDFS

The results change significantly with HDFS. Thanks to the higher quantity of content located in the input data, NMF is able to find reasonable models, althought not optimal, in all the three versions of the dataset.

| Results - HDFS - ORIGIN+MESSAGE | | | |
|---|---|---|---|
| Result Type | Hyperparameters | Coherence | To drop |
| Best Coherence model | K=63, kappa=0.05 | 0.7805±0.0176 | 15 |
| Best Model Analysis 1 | K=48, kappa=1 | 0.7486±0.0076 | 0 |
| Best Model Analysis 2 | K=30, kappa=1 | 0.7799±0.0236 | 0 |
| Best Model Analysis 3 | K=48, kappa=/ | 0.7269±0.0114 | 0 |

**Table 5.10:** NMF Results on ORIGIN+MESSAGE HDFS dataset

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 52 | java, eof, io_eof_exception, java_io_eof, io_eof, eof_exception, file, exception, java_io, queue | Caused by java.io.EOFException | cause, java, io, eof, exception, cause_java, java_io, io_eof, eof_exception, cause_java_io, java_io_eof, io_eof_exception |
| 48 | io, io_exception, host, java_io, peer, reset_peer, exception_connection, connection_reset, connection, connection_reset_peer | Caused by java.io.IOException: Connection reset by peer | cause, java, io, io, exception, connection, reset, peer, cause_java, java_io, io_exception, exception_connection, connection_reset, reset_peer, cause_java_io, io_exception_connection, exception_connection_reset, connection_reset_peer |
| 52 | java, eof, io_eof_exception, java_io_eof, io_eof, eof_exception, file, exception, java_io, queue | java.io.EOFException | java, io, eof, exception, java_io, io_eof, eof_exception, java_io_eof, io_eof_exception |
| 60 | fsdataset_impl, fsdataset, impl, datanode_fsdataset, datanode_fsdataset_impl, block_pool, add, pool, fsdataset_impl_total, total_time | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Total time to add all replicas to map: 0ms | fsdataset, impl, fsdataset, impl, total, time, add, replica, map, datanode_fsdataset, fsdataset_impl, fsdataset_impl, fsdataset_impl, impl_total, total_time, time_add, add_replica, replica_map, datanode_fsdataset_impl, fsdataset_impl_total, impl_total_time, total_time_add, time_add_replica, add_replica_map |
| 38 | fsdataset_impl, fsdataset, impl, datanode_fsdataset, datanode_fsdataset_impl, scan, state, disk, time_scan, total_time_scan | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Total time to scan all replicas for block pool BP-108841162-10.10.34.11-1440074360971: 8ms | fsdataset, impl, fsdataset, impl, total, time, scan, replica, pool, datanode_fsdataset, fsdataset_impl, fsdataset_impl, fsdataset_impl, impl_total, total_time, time_scan, scan_replica, replica_block, block_pool, datanode_fsdataset_impl, fsdataset_impl_total, impl_total_time, total_time_scan, time_scan_replica, scan_replica_block, replica_block_pool |
| 32 | start, datanode_start, lock_memory, max_lock_memory, datanode_max_lock, datanode_max, max, max_lock, start_datanode_max, memory | org.mortbay.log: jetty-6.1.26 | mortbay |
| 24 | filter, http, static, user, call, user_filter_context, add_filter_static, http_add_filter, filter_static_user, user_filter | org.mortbay.log: Logging to org.slf4j.impl.Log4jLoggerAdapter (org.mortbay.log) via org.mortbay.log.Slf4jLog | mortbay, log, mortbay_log |

**Figure 5.11:** Predictions NMF on HDFS - ORIGIN+MESSAGE

In ORIGIN+MESSAGE the best results is obtained with 63 topics, where 15 need to be dropped. By searching a model without similar clusters, in all cases the

results are lower but comparable to the best score. This high number of clusters affects the prediction in a good way, making the model to commit an error when it finds few rare words in the input sentences. Differently from LDA, in this situation I obtain good labels even with the interference caused by the ORIGIN.

| Results - HDFS - SPLIT | | | |
|---|---|---|---|
| Result Type | Hyperparameters | Coherence | To drop |
| Best Coherence model | K=30, kappa=0.1 | 0.8172±0.0344 | 2 |
| Best Model Analysis 1 | K=28, kappa=0.5 | 0.7795±0.0257 | 0 |
| Best Model Analysis 2 | K=23, kappa=1 | 0.8145±0.0214 | 0 |
| Best Model Analysis 3 | K=28, kappa=/ | 0.7777±0.0228 | 0 |

**Table 5.11:** NMF Results on SPLIT HDFS dataset

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 17 | eof, exception, java, io_eof, io_eof_exception, eof_exception, java_io_eof, java_io, see, detail_see | Caused by java.io.EOFException | cause, java, io, eof, exception, java_io, io_eof, eof_exception, java_io_eof, io_eof_exception |
| 21 | io, exception, host, io_exception, reset, connection_reset, peer, exception_connection, connection_reset_peer, connection | Caused by java.io.IOException: Connection reset by peer | cause, java, io, io, exception, connection, reset, peer, java_io, io_exception, exception_connection, connection_reset, reset_peer, io_exception_connection, exception_connection_reset, connection_reset_peer |
| 17 | eof, exception, java, io_eof, io_eof_exception, eof_exception, java_io_eof, java_io, see, detail_see | java.io.EOFException | java, io, eof, exception, java_io, io_eof, eof_exception, java_io_eof, io_eof_exception |
| 10 | impl, fsdataset, add, replica, add_replica, replica_map, add_replica_map, map, pool_volume, block_pool_volume | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDataset Impl: Total time to add all replicas to map: 0ms | fsdataset, impl, fsdataset, impl, total, time, add, replica, map, total_time, time_add, add_replica, replica_map, total_time_add, time_add_replica, add_replica_map |
| 10 | impl, fsdataset, add, replica, add_replica, replica_map, add_replica_map, map, pool_volume, block_pool_volume | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Total time to scan all replicas for block pool BP-108841162-10.10.34.11-1440074360971: 8ms | fsdataset, impl, fsdataset, impl, total, time, scan, replica, pool, total_time, time_scan, scan_replica, replica_block, block_pool, total_time_scan, time_scan_replica, scan_replica_block, replica_block_pool |
| 23 | start, service, offer, offer_service, map, add_replica_map, replica_map, add_replica, lock, replica | org.mortbay.log: jetty-6.1.26 | mortbay |
| 2 | filter, http, static, add, user, filter_static_user, filter_context, static_user, static_user_filter, filter_static | org.mortbay.log: Logging to org.slf4j.impl.Log4jLoggerAdapter (org.mortbay.log) via org.mortbay.log.Slf4jLog | mortbay, log, mortbay_log |

**Figure 5.12:** Predictions NMF on HDFS - SPLIT

The models on SPLIT reflect exactly the same reasoning just done on ORIGIN+MESSAGE, with the only difference in a lower number of topics, as usual for this case. All the searches to avoid similar topics do not provide best results. Only

the best model with no similar topics has a coherence score close to the best one, however, in this case the value of $K$ is to low to be taken into consideration.

| Results - HDFS - MESSAGE | | | |
|---|---|---|---|
| Result Type | Hyperparameters | Coherence | To drop |
| Best Coherence model | K=49, kappa=0.05 | 0.7699±0.0216 | 47 |
| Best Model Analysis 1 | K=40, kappa=0.1 | 0.7408±0.0192 | 0 |
| Best Model Analysis 2 | K=30, kappa=1 | 0.7603±0.0198 | 0 |
| Best Model Analysis 3 | K=40, kappa=/ | 0.7476±0.0212 | 0 |

**Table 5.12:** NMF Results on MESSAGE HDFS dataset

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 47 | exception, java, io, java_io, eof_exception, io_eof, eof, io_eof_exception, java_io_eof, host_destination | java.io.EOFException | java, io, eof, exception, java_io, io_eof, eof_exception, java_io_eof, io_eof_exception |
| 6 | io, io exception, connection reset peer, reset peer, exception connection, connection reset, exception connection reset, connection, peer, reset | java.io.IOException: Connection reset by peer | java, io, io, exception, connection, reset, peer, java_io, io_exception, exception_connection, connection_reset, reset_peer, io_exception_connection, exception_connection_reset, connection_reset_peer |
| 27 | replica, time, add_replica_map, replica_map, add_replica, map, add, time_add, time_add_replica, total | Total time to add all replicas to map: 0ms | total, time, add, replica, map, total_time, time_add, add_replica, replica_map, total_time_add, time_add_replica, add_replica_map |
| 27 | replica, time, add_replica_map, replica_map, add_replica, map, add, time_add, time_add_replica, total | Total time to scan all replicas for block pool BP-108841162-10.10.34. 11- 1440074360971: 8ms | total, time, scan, replica, pool, total_time, time_scan, scan_replica, replica_block, block_pool, total_time_scan, time_scan_replica, scan_replica_block, replica_block_pool |
| 28 | retry, interval, retry_policy, interval_cachereport, use, find, time, retry_connect_server, server_try, retry_maximum_count | jetty-6.1.26 | jetty |
| 31 | filter, static, user, user_filter, static_user, filter_context, add_filter_static, filter_static_user, static_user_filter, user_filter_context | Logging to org.slf4j.impl.Log4jLoggerAdapter (org.mortbay.log) via org.mortbay.log.Slf4jLog | log |
| 46 | writer_os, cache_take, os, cache, slow_manage_writer, writer, os_cache_take, manage_writer_os, manage_writer, writer_os_cache | org.apache.hadoop.hdfs.server. datanode.DataNode: Slow manageWriterOsCache took 5299ms (threshold=300ms) | slow, manage, writer, os, cache, take, slow_manage, manage_writer, writer_os, os_cache, cache_take, slow_manage_writer, manage_writer_os, writer_os_cache, os_cache_take |

**Figure 5.13:** Predictions NMF on HDFS - MESSAGE

As in LDA, the MESSAGE dataset provides good result with HDFS. Other than good values of coherence, althougth obtained with a number of topic than in my judgement is too low, the predictions depict this goodness. Indeed, in many situations where the results are light green or in red in the LDA, now better results

are associated. Worse results are still present for the most difficult cases, however topic clusters are way better and more coherent.

Other than LDA, also NMF is a suitable algorithm for this task, since it provides both reasonable number of topics and predictions. These experiments show how a simpler algorithms like NMF can provide good results in situation where the data in input are not incumbent [68].

### 5.3.3 LSI

LSI is considered one of the first technique for topic modeling. As described in the corresponding theoretical section of Chapter 4, its core operation is based on the decomposition of the input analysis by mean of their singular values.

Among the possible drawbacks, it has been mentioned that the probabilistic model of LSI is not able to match the observed data. This behaviour is confirmed by my experiments, where it is directly visible from the output produced. For this reason I decide to consider this model as a baseline and to keep it among my proposals since it represents a first step in the field of topic modeling.

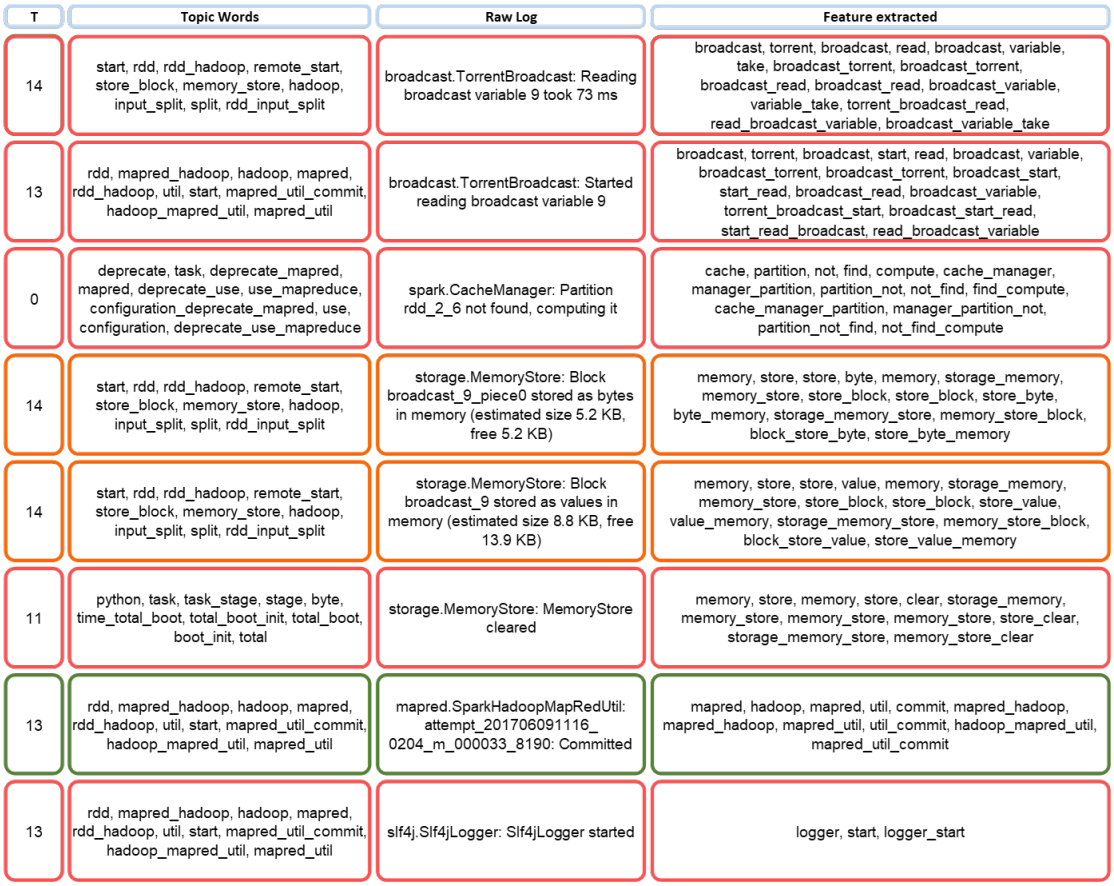
All the models have been trained by finetuning the number of topics with the same ranges explained before in this chapter. The input parameters are also the same since the implementation of other techniques, like explained in NMF, to represent the dataset rather than Bag-of-Word did not lead to better results.

#### Spark

| Results - Spark - ORIGIN+MESSAGE | | | |
|---|---|---|---|
| Result Type | Hyperparameters | Coherence | To drop |
| Best Coherence model | K=16 | 0.7197±0.0460 | 1 |
| Best Model Analysis 1 | K=15 | 0.6996±0.0399 | 0 |
| Best Model Analysis 2 | K=15 | 0.6996±0.0399 | 0 |
| Best Model Analysis 3 | K=14 | 0.6756±0.0477 | 0 |

**Table 5.13:** LSI Results on ORIGIN+MESSAGE Spark dataset

Looking at the results we can see how in ORIGIN+MESSAGE the coherence is in line with previous methods, however, the number of topic is too low. This is reflected in predictions that are not acceptable for most of the cases.



| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 14 | start, rdd, rdd_hadoop, remote_start, store_block, memory_store, hadoop, input_split, split, rdd_input_split | broadcast.TorrentBroadcast: Reading broadcast variable 9 took 73 ms | broadcast, torrent, broadcast, read, broadcast, variable, take, broadcast_torrent, broadcast_torrent, broadcast_read, broadcast_read, broadcast_variable, variable_take, torrent_broadcast_read, read_broadcast_variable, broadcast_variable_take |
| 13 | rdd, mapred_hadoop, hadoop, mapred, rdd_hadoop, util, start, mapred_util_commit, hadoop_mapred_util, mapred_util | broadcast.TorrentBroadcast: Started reading broadcast variable 9 | broadcast, torrent, broadcast, start, read, broadcast, variable, broadcast_torrent, broadcast_torrent, broadcast_start, start_read, broadcast_read, broadcast_variable, torrent_broadcast_start, broadcast_start_read, start_read_broadcast, read_broadcast_variable |
| 0 | deprecate, task, deprecate_mapred, mapred, deprecate_use, use_mapreduce, configuration_deprecate_mapred, use, configuration, deprecate_use_mapreduce | spark.CacheManager: Partition rdd_2_6 not found, computing it | cache, partition, not, find, compute, cache_manager, manager_partition, partition_not, not_find, find_compute, cache_manager_partition, manager_partition_not, partition_not_find, not_find_compute |
| 14 | start, rdd, rdd_hadoop, remote_start, store_block, memory_store, hadoop, input_split, split, rdd_input_split | storage.MemoryStore: Block broadcast_9_piece0 stored as bytes in memory (estimated size 5.2 KB, free 5.2 KB) | memory, store, store, byte, memory, storage_memory, memory_store, store_block, store_block, store_byte, byte_memory, storage_memory_store, memory_store_block, block_store_byte, store_byte_memory |
| 14 | start, rdd, rdd_hadoop, remote_start, store_block, memory_store, hadoop, input_split, split, rdd_input_split | storage.MemoryStore: Block broadcast_9 stored as values in memory (estimated size 8.8 KB, free 13.9 KB) | memory, store, store, value, memory, storage_memory, memory_store, store_block, store_block, store_value, value_memory, storage_memory_store, memory_store_block, block_store_value, store_value_memory |
| 11 | python, task, task_stage, stage, byte, time_total_boot, total_boot_init, total_boot, boot_init, total | storage.MemoryStore: MemoryStore cleared | memory, store, memory, store, clear, storage_memory, memory_store, memory_store, memory_store, store_clear, storage_memory_store, memory_store_clear |
| 13 | rdd, mapred_hadoop, hadoop, mapred, rdd_hadoop, util, start, mapred_util_commit, hadoop_mapred_util, mapred_util | mapred.SparkHadoopMapRedUtil: attempt_201706091116_ 0204_m_000033_8190: Committed | mapred, hadoop, mapred, util, commit, mapred_hadoop, mapred_hadoop, mapred_util, util_commit, hadoop_mapred_util, mapred_util_commit |
| 13 | rdd, mapred_hadoop, hadoop, mapred, rdd_hadoop, util, start, mapred_util_commit, hadoop_mapred_util, mapred_util | slf4j.Slf4jLogger: Slf4jLogger started | logger, start, logger_start |

**Figure 5.14:** Predictions LSI on Spark - ORIGIN+MESSAGE

| Results - Spark - SPLIT | | | |
|---|---|---|---|
| Result Type | Hyperparameters | Coherence | To drop |
| Best Coherence model | K=15 | 0.6705±0.0477 | 241 0 |
| Best Model Analysis 1 | K=15 | 0.6705±0.0477 | 0 |
| Best Model Analysis 2 | K=15 | 0.6705±0.0477 | 0 |
| Best Model Analysis 3 | K=15 | 0.6705±0.0477 | 0 |

**Table 5.14:** LSI Results on SPLIT Spark dataset

For SPLIT the result is similar to ORIGIN+MESSAGE, with a comparable coherence score and a low number of topic. In this case, the predictions seem to be worse. In detail, we can see a clear example of the discussed problems in the $6^{TH}$ row. Indeed, it is associated to a cluster which is strictly related to the first two rows. It shows how, even if the model is able to create meaningful clusters, it is

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 12 | start, shutdown, util, remote_start, service, listen_address, address, start_listen, start_listen_address, listen | broadcast.TorrentBroadcast: Reading broadcast variable 9 took 73 ms | broadcast, torrent, broadcast, read, broadcast, variable, take, read_broadcast, broadcast_variable, variable_take, read_broadcast_variable, broadcast_variable_take |
| 13 | register, block_manager, master, register_block, register_block_manager, try, try_register, try_register_block, start, remote_start | broadcast.TorrentBroadcast: Started reading broadcast variable 9 | broadcast, torrent, broadcast, start, read, broadcast, variable, start_read, read_broadcast, broadcast_variable, start_read_broadcast, read_broadcast_variable |
| 0 | deprecate, task, mapred, deprecate_use_mapreduce, use_mapreduce, deprecate_use, use, mapreduce, configuration, use_mapreduce_task | spark.CacheManager: Partition rdd_2_6 not found, computing it | cache, partition, not, find, compute, partition_not, not_find, find_compute, partition_not_find, not_find_compute |
| 12 | start, shutdown, util, remote_start, service, listen_address, address, start_listen, start_listen_address, listen | storage.MemoryStore: Block broadcast_9_piece0 stored as bytes in memory (estimated size 5.2 KB, free 5.2 KB) | memory, store, store, byte, memory, block_store, store_byte, byte_memory, block_store_byte, store_byte_memory |
| 12 | start, shutdown, util, remote_start, service, listen_address, address, start_listen, start_listen_address, listen | storage.MemoryStore: Block broadcast_9 stored as values in memory (estimated size 8.8 KB, free 13.9 KB) | memory, store, store, value, memory, block_store, store_value, value_memory, block_store_value, store_value_memory |
| 5 | broadcast, variable, broadcast_variable, torrent, read_broadcast, read, read_broadcast_variable, start, start_read_broadcast, start_read | storage.MemoryStore: MemoryStore cleared | memory, store, memory, store, clear, memory_store, store_clear, memory_store_clear |
| 9 | util, shutdown, hook, call, hook_call, shutdown_hook_call, shutdown_hook, python, directory, service | mapred.SparkHadoopMapRedUtil: attempt_201706091116_ 0204_m_000033_8190: Committed | mapred, hadoop, mapred, util, commit |
| 13 | register, block_manager, master, register_block, register_block_manager, try, try_register, try_register_block, start, remote_start | slf4j.Slf4jLogger: Slf4jLogger started | logger, start, logger_start |

**Figure 5.15:** Predictions LSI on Spark - SPLIT

not able to always do the proper associations. A peculiarity due to the low value of $K$ is the absence of similar topics, making unnecessary further analysis.

| Results - Spark - MESSAGE | | | |
|---|---|---|---|
| Result Type | Hyperparameters | Coherence | To drop |
| Best Coherence model | K=15 | 0.5401±0.0629 | 0 |
| Best Model Analysis 1 | K=15 | 0.5401±0.0629 | 0 |
| Best Model Analysis 2 | K=15 | 0.5401±0.0629 | 0 |
| Best Model Analysis 3 | K=15 | 0.5401±0.0629 | 0 |

**Table 5.15:** LSI Results on MESSAGE Spark dataset

The results obtained on MESSAGE are the lowest seen so far. The coherence suggests the need of further reasonings, however, by contrast, the predictions seem

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 6 | start, variable, broadcast, broadcast_variable, read, read_broadcast_variable, read_broadcast, start_read, start_read_broadcast, block | broadcast.TorrentBroadcast: Reading broadcast variable 9 took 73 ms | read, broadcast, variable, take, read_broadcast, broadcast_variable, variable_take, read_broadcast_variable, broadcast_variable_take |
| 6 | start, variable, broadcast, broadcast_variable, read, read_broadcast_variable, read_broadcast, start_read, start_read_broadcast, block | broadcast.TorrentBroadcast: Started reading broadcast variable 9 | start, read, broadcast, variable, start_read, read_broadcast, broadcast_variable, start_read_broadcast, read_broadcast_variable |
| 13 | driver, shutdown, command_shutdown, driver_command, driver_command_shutdown, command, disassociate, disassociate_shutdown | spark.CacheManager: Partition rdd_2_6 not found, computing it | partition, not, find, compute, partition_not, not_find, find_compute, partition_not_find, not_find_compute |
| 3 | driver, byte, stage, task_stage, finish, task, finish_task_stage, send, stage_byte, finish_task | storage.MemoryStore: Block broadcast_9_piece0 stored as bytes in memory (estimated size 5.2 KB, free 5.2 KB) | block, store, byte, memory, block_store, store_byte, byte_memory, block_store_byte, store_byte_memory |
| 3 | driver, byte, stage, task_stage, finish, task, finish_task_stage, send, stage_byte, finish_task | storage.MemoryStore: Block broadcast_9 stored as values in memory (estimated size 8.8 KB, free 13.9 KB) | block, store, value, memory, block_store, store_value, value_memory, block_store_value, store_value_memory |
| 3 | driver, byte, stage, task_stage, finish, task, finish_task_stage, send, stage_byte, finish_task | storage.MemoryStore: MemoryStore cleared | memory, store, clear, memory_store, store_clear, memory_store_clear |
| 12 | task_ismap, ismap, mapred_deprecate, mapred_deprecate_use, partition, task, driver, mapred_task_ismap, mapreduce_task_ismap, ismap_deprecate_use | mapred.SparkHadoopMapRedUtil: attempt_201706091116_0204_m_000033_8190: Committed | attempt, commit, attempt_commit |
| 11 | start, remote_start, listen_address, start_listen_address, address, start_listen, remote_start_listen, listen, remote, broadcast_variable | slf4j.Slf4jLogger: Slf4jLogger started | logger, start, logger_start |

**Figure 5.16:** Predictions LSI on Spark - MESSAGE

to be a little more reasonable with respect to the previous two cases. Also in this version the similarity analysis are not needed.

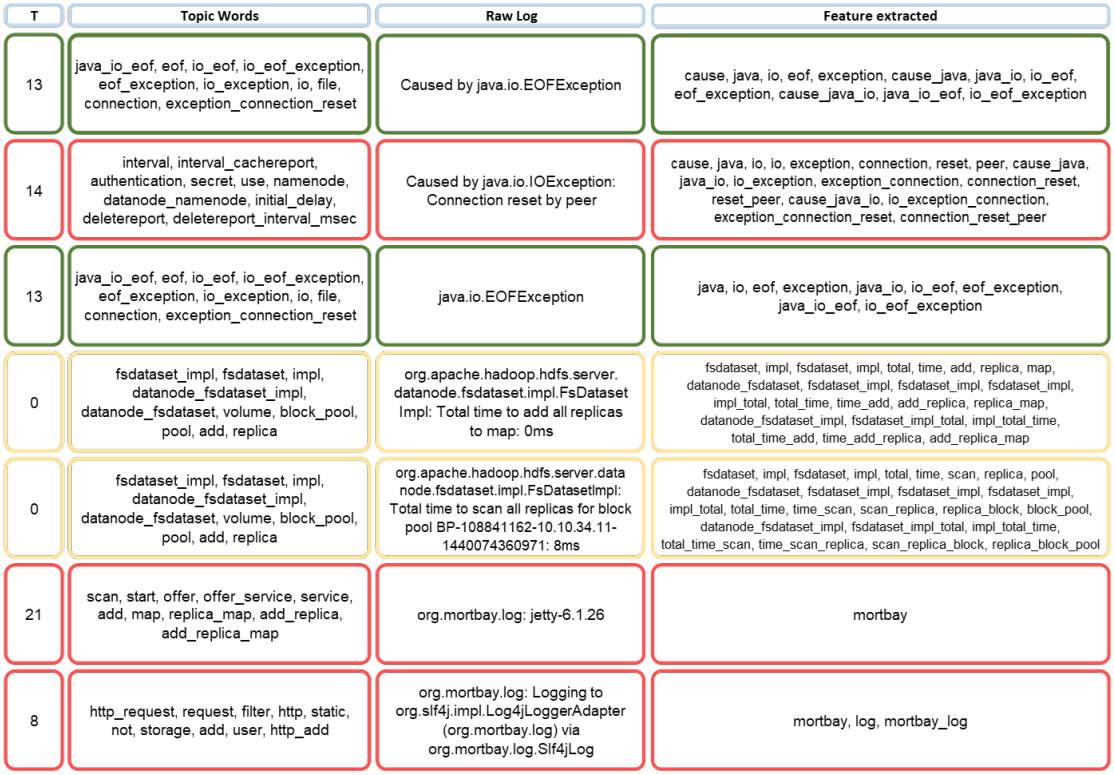
### HDFS

The behaviour expressed with the HDFS datasets follows the same pattern seen in Spark. In all cases the number of topics is too low with respect to a possible human judgement. However, the predictions are more acceptable, thanks to the higher amount of input data that helps the models to better perform.

The coherence score for the best model on ORIGIN+MESSAGE is obtained with 22 topics. It causes the production of too generic topics, with the recurrent issue related to the higher influence of the ORIGIN. The poorer quality of the algorithm is reflected in the second row, where both LDA and NMF are able to cluster that sentence properly. Due to the lower value of $K$, the number of similar topics is small, letting the first and the second analysis to obtain the same score,

| Results - HDFS - ORIGIN+MESSAGE | | | |
|---|---|---|---|
| Result Type | Hyperparameters | Coherence | To drop |
| Best Coherence model | K=22 | 0.7590±0.0412 | 3 |
| Best Model Analysis 1 | K=17 | 0.7143±0.0735 | 0 |
| Best Model Analysis 2 | K=17 | 0.7143±0.0735 | 0 |
| Best Model Analysis 3 | K=19 | 0.5433±0.0573 | 0 |

**Table 5.16:** LSI Results on ORIGIN+MESSAGE HDFS dataset



| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 13 | java_io_eof, eof, io_eof, io_eof_exception, eof_exception, io_exception, io, file, connection, exception_connection_reset | Caused by java.io.EOFException | cause, java, io, eof, exception, cause_java, java_io, io_eof, eof_exception, cause_java_io, java_io_eof, io_eof_exception |
| 14 | interval, interval_cachereport, authentication, secret, use, namenode, datanode_namenode, initial_delay, deletereport, deletereport_interval_msec | Caused by java.io.IOException: Connection reset by peer | cause, java, io, io, exception, connection, reset, peer, cause_java, java_io, io_exception, exception_connection, connection_reset, reset_peer, cause_java_io, io_exception_connection, exception_connection_reset, connection_reset_peer |
| 13 | java_io_eof, eof, io_eof, io_eof_exception, eof_exception, io_exception, io, file, connection, exception_connection_reset | java.io.EOFException | java, io, eof, exception, java_io, io_eof, eof_exception, java_io_eof, io_eof_exception |
| 0 | fsdataset_impl, fsdataset, impl, datanode_fsdataset_impl, datanode_fsdataset, volume, block_pool, pool, add, replica | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Total time to add all replicas to map: 0ms | fsdataset, impl, fsdataset, impl, total, time, add, replica, map, datanode_fsdataset, fsdataset_impl, fsdataset_impl, fsdataset_impl, impl_total, total_time, time_add, add_replica, replica_map, datanode_fsdataset_impl, fsdataset_impl_total, impl_total_time, total_time_add, time_add_replica, add_replica_map |
| 0 | fsdataset_impl, fsdataset, impl, datanode_fsdataset_impl, datanode_fsdataset, volume, block_pool, pool, add, replica | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Total time to scan all replicas for block pool BP-108841162-10.10.34.11-1440074360971: 8ms | fsdataset, impl, fsdataset, impl, total, time, scan, replica, pool, datanode_fsdataset, fsdataset_impl, fsdataset_impl, fsdataset_impl, impl_total, total_time, time_scan, scan_replica, replica_block, block_pool, datanode_fsdataset_impl, fsdataset_impl_total, impl_total_time, total_time_scan, time_scan_replica, scan_replica_block, replica_block_pool |
| 21 | scan, start, offer, offer_service, service, add, map, replica_map, add_replica, add_replica_map | org.mortbay.log: jetty-6.1.26 | mortbay |
| 8 | http_request, request, filter, http, static, not, storage, add, user, http_add | org.mortbay.log: Logging to org.slf4j.impl.Log4jLoggerAdapter (org.mortbay.log) via org.mortbay.log.Slf4jLog | mortbay, log, mortbay_log |

**Figure 5.17:** Predictions LSI on HDFS - ORIGIN+MESSAGE

while for the third one decreases drastically.

| Results - HDFS - SPLIT | | | |
|---|---|---|---|
| Result Type | Hyperparameters | Coherence | To drop |
| Best Coherence model | K=17 | 0.6934±0.0197 | 0 |
| Best Model Analysis 1 | K=17 | 0.6934±0.0197 | 0 |
| Best Model Analysis 2 | K=17 | 0.6934±0.0197 | 0 |
| Best Model Analysis 3 | K=17 | 0.6934±0.0197 | 0 |

**Table 5.17:** LSI Results on SPLIT HDFS dataset

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 15 | eof, java_io_eof, io_eof, io_eof_exception, eof_exception, io, io_exception, exception_connection, io_exception_connection, reset_peer | Caused by java.io.EOFException | cause, java, io, eof, exception, java_io, io_eof, eof_exception, java_io_eof, io_eof_exception |
| 0 | exception, java, socket, timeout, channel, channel_socket, io, timeout_exception, socket_timeout, socket_timeout_exception | Caused by java.io.IOException: Connection reset by peer | cause, java, io, io, exception, connection, reset, peer, java_io, io_exception, exception_connection, connection_reset, reset_peer, io_exception_connection, exception_connection_reset, connection_reset_peer |
| 15 | eof, java_io_eof, io_eof, io_eof_exception, eof_exception, io, io_exception, exception_connection, io_exception_connection, reset_peer | java.io.EOFException | java, io, eof, exception, java_io, io_eof, eof_exception, java_io_eof, io_eof_exception |
| 3 | scanner, volume, fsdataset, impl, volume_scanner, rescan, suspect_block, suspect, not, volume_scanner_not | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Total time to add all replicas to map: 0ms | fsdataset, impl, fsdataset, impl, total, time, add, replica, map, total_time, time_add, add_replica, replica_map, total_time_add, time_add_replica, add_replica_map |
| 3 | scanner, volume, fsdataset, impl, volume_scanner, rescan, suspect_block, suspect, not, volume_scanner_not | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Total time to scan all replicas for block pool BP-108841162-10.10.34.11-1440074360971: 8ms | fsdataset, impl, fsdataset, impl, total, time, scan, replica, pool, total_time, time_scan, scan_replica, replica_block, block_pool, total_time_scan, time_scan_replica, scan_replica_block, replica_block_pool |
| 14 | metric, system, start, impl, fsdataset, datanode_metric_system, system_start, datanode_metric, metric_system_start, metric_system | org.mortbay.log: jetty-6.1.26 | mortbay |
| 4 | http, filter, static, add, user, filter_context, filter_static_user, context, add_filter_static, add_filter | org.mortbay.log: Logging to org.slf4j.impl.Log4jLoggerAdapter (org.mortbay.log) via org.mortbay.log.Slf4jLog | mortbay, log, mortbay_log |

**Figure 5.18:** Predictions LSI on HDFS - SPLIT

In SPLIT the situation related to the predictions improves. Indeed, in this case the second row is correctly labeled, however, the lower cohesion of the words inside the topics persists.

| Results - HDFS - MESSAGE | | | |
|---|---|---|---|
| Result Type | Hyperparameters | Coherence | To drop |
| Best Coherence model | K=22 | 0.6365±0.0163 | 0 |
| Best Model Analysis 1 | K=22 | 0.6365±0.0163 | 0 |
| Best Model Analysis 2 | K=22 | 0.6365±0.0163 | 0 |
| Best Model Analysis 3 | K=22 | 0.6365±0.0163 | 0 |

**Table 5.18:** LSI Results on MESSAGE HDFS dataset

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 0.0 | channel, socket, timeout, java, exception, channel_socket, socket_timeout, timeout_exception, socket_timeout_exception, wait | java.io.EOFException | java, io, eof, exception, java_io, io_eof, eof_exception, java_io_eof, io_eof_exception |
| 0.0 | channel, socket, timeout, java, exception, channel_socket, socket_timeout, timeout_exception, socket_timeout_exception, wait | java.io.IOException: Connection reset by peer | java, io, io, exception, connection, reset, peer, java_io, io_exception, exception_connection, connection_reset, reset_peer, io_exception_connection, exception_connection_reset, connection_reset_peer |
| 9.0 | replica, add, add_replica_map, map, replica_map, add_replica, service, pool_volume, block_pool_volume, time | Total time to add all replicas to map: 0ms | total, time, add, replica, map, total_time, time_add, add_replica, replica_map, total_time_add, time_add_replica, add_replica_map |
| 2.0 | report, block_pool, pool, command, finalize_command, msec, sent, msec_generate, report_sent, total | Total time to scan all replicas for block pool BP-108841162-10.10.34. 11- 1440074360971: 8ms | total, time, scan, replica, pool, total_time, time_scan, scan_replica, replica_block, block_pool, total_time_scan, time_scan_replica, scan_replica_block, replica_block_pool |
| 20.0 | datanode, active, state, service, offer_service, offer, active_state, start, scan, namenode_block_pool | jetty-6.1.26 | jetty |
| 17.0 | directory, scan, format, service, storage, storage_directory, length, namenode, directory_not, not_format | Logging to org.slf4j.impl.Log4jLoggerAdapter (org.mortbay.log) via org.mortbay.log.Slf4jLog | log |
| 17.0 | directory, scan, format, service, storage, storage_directory, length, namenode, directory_not, not_format | org.apache.hadoop.hdfs.server. datanode.DataNode: Slow manageWriterOsCache took 5299ms (threshold=300ms) | slow, manage, writer, os, cache, take, slow_manage, manage_writer, writer_os, os_cache, cache_take, slow_manage_writer, manage_writer_os, writer_os_cache, os_cache_take |

**Figure 5.19:** Predictions LSI on HDFS - MESSAGE

In the application on MESSAGE the quality, instead, drops drastically. Both coherence and number of topics are low, causing poor topic clusters and, as consequence, wrong labeling. As in SPLIT, this version do not provide similar topics so the similarity analysis is not needed.

As explained in the beginning of this subsection, LSI is not able to provide good topics, making it not usable for a real case scenario.

In general, we have seen how classical algorithms are good in many common cases, but they start to have some issues when the number of words are lower. This behaviour is perfectly represented by the two dataset sources. Although both provide datasets with few word for each line, the higher number of rows in HDFS helps the algorithms to produces better models, connected to acceptable value of $K$ and predictions.

### 5.3.4 ProdLDA

ProdLDA is one of the first proposal to use Neural Network in Topic Modelling. As described in Chapter 4, it tries to overcome some issues related to the classical formulation of LDA, concerning the component collapsing and the Dirichlet prior not being a location scale family. For my implementation, the core algorithm comes from OCTIS and it has been fine-tuned in order to be adapted to my situation. The model, as indicated by the authors, uses ADAM as optimizer, while the choice for the activation functions has fallen on softplus. All the other parameters have been fine-tuned to obtained the best model. In detail, the number of hidden layers is chosen between 1 and 2, since higher values could have increase to much the complexity. Following the advice inside the paper, since utilization of Batch Normalization layers, the value for the momentum is chosen between $[0.8, 0.9, 0.99]$, where its recommended a value greater than 0.8. For the same reason, the learning rate has been tested in the range $[0.001, 0.005, 0.01, 0.05, 0.1, 0.5]$, considered the a recommended range between 0.001 and 0.1. For what concerns the dropout, it has been chosen between $[0.1, 0.2, 0.3]$, while the number of neurons and the number of epochs between 100 and 200. The last parameters is, as usual, the number of topics, here implemented in the same range as for the other algorithms.

Regarding the results, as anticipated in the introduction of this section, they describe the best combination that provides the highest coherence score and, at the same time, with a level of similarity among clusters lower than 85%.

In the following tables the hyper-parameters are represented as: the number of topics $K$, the number of layer $\eta$, the number of neurons $\gamma$, the dropout $\pi$, the number of epoch $\rho$, the momentum $\omega$ and the learning rate $\nu$.

#### Spark

Beginning on Spark, on ORIGIN+MESSAGE the coherence score is higher respect to LDA. This is reflected also in the predictions, which are good but, in some cases, they are not correct or partially correct. Indeed, in some situations the label is assigned considering only the ORIGIN part, in other cases, instead, the sentence is assigned to a cluster which clearly represent a mixture of themes, making it not acceptable.

| Results - Spark - All Versions | | |
|---|---|---|
| Dataset | Hyperparameters | Coherence |
| O+M | K=17,$\eta = 1, \gamma = 100, \pi = 0.2, \rho = 100, \omega = 0.9, \nu = 0.1$ | 0.7131±0.0194 |
| S | K=33,$\eta = 2, \gamma = 100, \pi = 0.1, \rho = 100, \omega = 0.9, \nu = 0.001$ | 0.7391±0.0215 |
| M | K=23,$\eta = 2, \gamma = 200, \pi = 0.3, \rho = 100, \omega = 0.9, \nu = 0.001$ | 0.7485±0.0261 |

**Table 5.19:** ProdLDA Results on all Spark datasets

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 6 | shutdown, shutdown_hook_call, call, host, driver, stage_byte_result, find_block, not_find_compute, store_byte, driver_command_shutdown, start, broadcast_variable_take | broadcast.TorrentBroadcast: Reading broadcast variable 9 took 73 ms | broadcast, torrent, broadcast, read, broadcast, variable, take, read_broadcast, broadcast_variable, variable_take, read_broadcast_variable, broadcast_variable_take |
| 14 | manager_stop, partition_not_find, compute, version, signal_handler, disable_acls, stop, directory, register_block, connect, run_task_stage, view_acls_yarn | broadcast.TorrentBroadcast: Started reading broadcast variable 9 | broadcast, torrent, broadcast, start, read, broadcast, variable, start_read, read_broadcast, broadcast_variable, start_read_broadcast, read_broadcast_variable |
| 0 | torrent, time, master, partition_deprecate, runner, store_byte_memory, authentication_disable_acls, partition_not_find, server, time_total, file_output_committer, not_find_compute | spark.CacheManager: Partition rdd_2_6 not found, computing it | cache, partition, not, find, compute, partition_not, not_find, find_compute, partition_not_find, not_find_compute |
| 0 | torrent, time, master, partition_deprecate, runner, store_byte_memory, authentication_disable_acls, partition_not_find, server, time_total, file_output_committer, not_find_compute | storage.MemoryStore: Block broadcast_9_piece0 stored as bytes in memory (estimated size 5.2 KB, free 5.2 KB) | memory, store, store, byte, memory, block_store, store_byte, byte_memory, block_store_byte, store_byte_memory |
| 13 | task_deprecate, configuration, yarn, signal, init_finish, executor_host, value_memory, modify_acls, memory_store_start, start_service_port, ismap_deprecate_use, memory, stage_byte, security_manager | storage.MemoryStore: Block broadcast_9 stored as values in memory (estimated size 8.8 KB, free 13.9 KB) | memory, store, store, value, memory, block_store, store_value, value_memory, block_store_value, store_value_memory |
| 0 | torrent, time, master, partition_deprecate, runner, store_byte_memory, authentication_disable_acls, partition_not_find, server, time_total, file_output_committer, not_find_compute | storage.MemoryStore: MemoryStore cleared | memory, store, memory, store, clear, memory_store, store_clear, memory_store_clear |
| 31 | remote, register_signal_handler, register_driver, curi, signal, driver_command_shutdown, register_block_manager, acls_yarn_curi, stop, register, mapred_task_ismap | mapred.SparkHadoopMapRedUtil: attempt_201706091116_ 0204_m_000033_8190: Committed | mapred, hadoop, mapred, util, commit |
| 30 | provider, memory_store_start, shutdown_remote, master, mapred_task_deprecate, remote_daemon, hadoop, register_signal_handler, run_task, try, hook, file, coarse, file_output, handler | slf4j.Slf4jLogger: Slf4jLogger started | logger, start, logger_start |

**Figure 5.20:** Predictions ProdLDA on Spark - SPLIT

On the other two cases the situation is different, even if the metric scores are higher than in LDA, the consideration is flipped by looking at the predictions. It is clear that the aim of ProdLDA is achievable for common implementation, however, in situation with few data, it still has some troubles about finding good clusters and correctly assigning the labels. In SPLIT, this reasoning is depicted in the first four rows. The first two are wrongly assigned, while, from the following two, we can

see that the topic cluster would have been more reasonable for the first two cases. This poor generalization is perfectly represented also in the MESSAGE version. Moreover, this condition is underlined by the coherence scores on the top 10 and 15 words. In both two cases they are lower with respect to the on 20 words, meaning that, in general, the topic clusters are a mixture that fits well all sentences, but no-one is perfectly suitable.

### HDFS

| Results - HDFS - All Versions | | |
|---|---|---|
| Dataset | Hyperparameters | Coherence |
| O+M | K=39,$\eta = 1, \gamma = 100, \pi = 0.2, \rho = 100, \omega = 0.9, \nu = 0.005$ | 0.7329±0.0321 |
| S | K=23,$\eta = 2, \gamma = 200, \pi = 0.2, \rho = 100, \omega = 0.8, \nu = 0.005$ | 0.7251±0.0087 |
| M | K=23,$\eta = 2, \gamma = 100, \pi = 0.1, \rho = 100, \omega = 0.9, \nu = 0.01$ | 0.7256±0.0164 |

**Table 5.20:** ProdLDA Results on ORIGIN+MESSAGE HDFS dataset

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 9 | io_eof_exception, file, java_io_eof, eof, eof_exception, eof_exception_end, end, see, exception, io_eof, exception_end_file, exception_local_host, java_io, end_file_exception, host_java_io, exception_end, detail_see… | Caused by java.io.EOFException | cause, java, io, eof, exception, cause_java, java_io, io_eof, eof_exception, cause_java_io, java_io_eof, io_eof_exception |
| 34 | cause_java_io, io_exception, exception_connection, peer, reset, connection_reset_peer, connection, exception_connection_reset, io, secret, connection_reset, io_exception_connection… | Caused by java.io.IOException: Connection reset by peer | cause, java, io, io, exception, connection, reset, peer, cause_java, java_io, io_exception, exception_connection, connection_reset, reset_peer, cause_java_io, io_exception_connection, exception_connection_reset, connection_reset_peer |
| 21 | io_eof_exception, io_eof, eof, java_io_eof, eof_exception, end, eof_exception_end, see, end_file_exception, detail_see, exception_local_host, exception_end_file, end_file, java_io, file_exception… | java.io.EOFException | java, io, eof, exception, java_io, io_eof, eof_exception, java_io_eof, io_eof_exception |
| 16 | fsdataset, fsdataset_async, datanode_fsdataset, fsdataset_impl, fsdataset_async_disk, disk_service, datanode_fsdataset_impl, async_disk, async_disk_service, delete, impl, impl_fsdataset_async, fsdataset_impl_total… | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Total time to add all replicas to map: 0ms | fsdataset, impl, fsdataset, impl, total, time, add, replica, map, datanode_fsdataset, fsdataset_impl, fsdataset_impl, fsdataset_impl, impl_total, total_time, time_add, add_replica, replica_map, datanode_fsdataset_impl, fsdataset_impl_total, impl_total_time, total_time_add, time_add_replica, add_replica_map |
| 16 | fsdataset, fsdataset_async, datanode_fsdataset, fsdataset_impl, fsdataset_async_disk, disk_service, datanode_fsdataset_impl, async_disk, async_disk_service, delete, impl, impl_fsdataset_async, fsdataset_impl_total… | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Total time to scan all replicas for block pool BP-108841162-10.10.34.11-1440074360971: 8ms | fsdataset, impl, fsdataset, impl, total, time, scan, replica, pool, datanode_fsdataset, fsdataset_impl, fsdataset_impl, fsdataset_impl, impl_total, total_time, time_scan, scan_replica, replica_block, block_pool, datanode_fsdataset_impl, fsdataset_impl_total, impl_total_time, total_time_scan, time_scan_replica, scan_replica_block, replica_block_pool |
| 13 | packet, receiver_write_packet, take, mirror, block_receiver, write_packet, slow_block, slow, packet_mirror_take, receiver, receiver_write, mirror_take, datanode_slow_block, write, write_packet_mirror, packet_mirror… | org.mortbay.log: jetty-6.1.26 | mortbay |
| 29 | http_listen, listen, traffic, receive_src, start_socket_reader, receive_src_dest, datanode_receive, datanode_generate_persist, start_socket, datanode_supergroup, http_traffic, receive, reader, persist_new_datanode… | org.mortbay.log: Logging to org.slf4j.impl.Log4jLoggerAdapter (org.mortbay.log) via org.mortbay.log.Slf4jLog | mortbay, log, mortbay_log |

**Figure 5.21:** Predictions ProdLDA on HDFS - ORIGIN+MESSAGE

The quality in HDFS, instead, is comparable between LDA and ProdLDA, both in terms of scores and predictions. The main difference with respect to the implementation with Spark is the better clusters made in SPLIT and MESSAGE datasets. Due to the higher quantity of data the coherences with respect to the different top words follow the correct growing trend. However, by looking at the number of topics found, the conclusion is that some drawbacks linked to this proposal still exist. Indeed, as LDA, it finds the best model with a small value of $K$, in conflict with the human judgement. This affects the predictions, since many sentences are not correctly labeled because the model was not able to find all the necessary topic clusters.

### 5.3.5 BERTopic

In Chapter 4 BERTopic is proposed as an evolution of Top2Vec. Indeed, rather that relying on a Doc2Vec model to transform words, it uses BERT to embed sentences into a numerical representation. This approach is more practical than the one proposed in Top2Vec. Indeed, the training of a complete and meaningful Doc2Vec model is an expensive process, requiring time, computational power and huge quantity of data. Regarding this last one, the situation in my project makes its implementation unfeasible, as observed in different trials. For this reason the focus is placed exclusively on BERTopic. The core of the BERTopic algorithm is made by 3 steps. In the first one the pre-trained model to embed the sentences is chosen, in my case the best performing is *all-MiniLM-L6-v2*.
As second step, the dimensionality reduction is applied on the features obtained. UMAP is a powerful tool but it requires many parameters to be tuned properly. For this project both the number of neighbors and the number of components have been fine-tuned. The first one controls how UMAP balances local versus global structure in the data. It is done by constraining the size of the local neighborhood UMAP will look when attempting to lean the manifold structure of the data. In this way UMAP concentrates on very local structure. The second one, instead, is used to determine the dimensionality of the reduced space where data are embedded. Other parameters have been used with fixed values. The metric distance used is the cosine, while the minimum distance, which allows how tightly UMAP can pack points together, is equal to 0. For my cases, the number of neighbors is used a range from 5 to 20, avoiding large value since otherwise too many data are taken into account. Regarding the number of components, instead, I choose among 5, 10 and 15, since with higher dimensions the computation cost starts to be too expensive Once data are reduced, HDBSCAN is applied on them. It is implemented using the euclidean distance and with a fixed value for the minimum cluster size equal to 2. About this last one, some considerations are necessary. Due to the nature of HDBSCAN, cluster with unitary size are not allowed. This is a huge problem

for my situation, since many line logs have to be taken independently. With the classical implementation, many of these logs are labelled as outliers, making the whole algorithm not suitable. To overcome this issues I propose two solutions regarding the logs considered as outliers. In the first solution I take each outlier as a cluster with unitary size, in this way those elements are kept rather than being discarded. However, with this solutions, it may happens, when the model makes an error, to consider independent clusters when an existing topic suitable for those logs exists. Directly from this consideration, I present the second solution. For those logs considered as outliers, before assigning a new label, I check if a match exists with the topics found. This control is done by looking at the similarity, by using the same approach already proposed in other algorithms. With this solution I am able to cluster even better those logs, furtherly improving coherence score with respect to the initial one. The proposed pictures regarding the predictions are obtained with the second solution proposed.

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 27 | coarse grain backend driver command shutdown driver_command command_shutdown driver_command_shutdown | executor.CoarseGrainedExecutor Backend: Driver commanded a shutdown | coarse, grain, backend, driver, command, shutdown, driver_command, command_shutdown, driver_command_shutdown |
| 13 | driver_disassociate_shutdown driver_disassociate disassociate_shutdown disassociate driver shutdown grain coarse backend | | |
| 29 | configuration, deprecate, mapred, task, partition, use, task, mapreduce, mapred_task, task_partition, partition_deprecate, deprecate_use, use_mapreduce, mapreduce_task, partition, mapreduce_task_partition, mapred_task_partition, task_partition_deprecate, partition_deprecate_use, deprecate_use_mapreduce, use_mapreduce_task | Configuration.deprecation: mapred.task.partition is deprecated. Instead use mapreduce.task.partition | configuration, deprecate, mapred, task, partition, deprecate, use, mapreduce, task, partition, mapred_task, task_partition, partition_deprecate, deprecate_use, use_mapreduce, mapreduce_task, task_partition, mapred_task_partition, task_partition_deprecate, partition_deprecate_use, deprecate_use_mapreduce, use_mapreduce_task, mapreduce_task_partition |
| 1 | deprecate, mapreduce, use_mapreduce, use, configuration, deprecate_use, deprecate_use_mapreduce, mapred, task, use_mapreduce_task, mapreduce_task, task_ismap, ismap, mapred_deprecate, mapred_deprecate_use, mapred_task, mapred_task_deprecate, task_deprecate, task_deprecate_use, task_attempt | | |

**Figure 5.22:** Example of sentences assigned to existing topics, rather than creating new ones.

### Spark

In table 5.21 we can appreciate the big improvements in the results compared to the algorithms seen so far. On ORIGIN+MESSAGE BERTopic finds 19 topics as starting point, however, by applying the first proposed solution, I obtain a coherence of 0.89 with 29 topics, a good value for Spark. With the second solution the coherence score decreases, but this is expected since I force the model to chose a similar existing topic rather than creating a new one with a perfect match, as shown in Figure 5.22.

| Results - Spark - ORIGIN+MESSAGE | | | |
|---|---|---|---|
| Result Type | Hyperparameters | Coherence | To drop |
| Best Coherence Model | K=19, NC=10, NN=15 | 0.8437±0.0258 | / |
| Best Model w Outliers | K=29, NC=10, NN=15 | 0.8903±0.0176 | 1 |
| Best Model w Similarity | K=28, NC=10, NN=15 | 0.8876±0.0113 | 0 |

**Table 5.21:** BERTopic Results on ORIGIN+MESSAGE Spark dataset

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 10 | broadcast, broadcast_torrent, broadcast_read, variable, torrent, broadcast_variable, read_broadcast_variable, read, start broadcast_variable_take, take, variable_take, broadcast_start, broadcast_start_read… | broadcast.TorrentBroadcast: Reading broadcast variable 9 took 73 ms | broadcast, torrent, broadcast, read, broadcast, variable, take, broadcast_torrent, broadcast_torrent, broadcast_read, broadcast_read, broadcast_variable, variable_take, torrent_broadcast_read, read_broadcast_variable, broadcast_variable_take |
| 10 | broadcast, broadcast_torrent, broadcast_read, variable, torrent, broadcast_variable, read_broadcast_variable, read, start broadcast_variable_take, take, variable_take, broadcast_start, broadcast_start_read… | broadcast.TorrentBroadcast: Started reading broadcast variable 9 | broadcast, torrent, broadcast, start, read, broadcast, variable, broadcast_torrent, broadcast_torrent, broadcast_start, start_read, broadcast_read, broadcast_variable, torrent_broadcast_start, broadcast_start_read, start_read_broadcast, read_broadcast_variable |
| 28 | cache, partition, not, find, compute, cache_manager, manager_partition, partition_not, not_find, find_compute, cache_manager_partition, manager_partition_not, partition_not_find, not_find_compute | spark.CacheManager: Partition rdd_2_6 not found, computing it | cache, partition, not, find, compute, cache_manager, manager_partition, partition_not, not_find, find_compute, cache_manager_partition, manager_partition_not, partition_not_find, not_find_compute |
| 3 | store_block, store, memory, byte, memory_store_block, storage_memory_store, storage_memory, block_store_byte, executor_finish_task, byte_result, byte_memory | storage.MemoryStore: Block broadcast_9_piece0 stored as bytes in memory (estimated size 5.2 KB, free 5.2 KB) | memory, store, store, byte, memory, storage_memory, memory_store, store_block, store_block, store_byte, byte_memory, storage_memory_store, memory_store_block, block_store_byte, store_byte_memory |
| 3 | store_block, store, memory, byte, memory_store_block, storage_memory_store, storage_memory, block_store_byte, executor_finish_task, byte_result, byte_memory | storage.MemoryStore: Block broadcast_9 stored as values in memory (estimated size 8.8 KB, free 13.9 KB) | memory, store, store, value, memory, storage_memory, memory_store, store_block, store_block, store_value, value_memory, storage_memory_store, memory_store_block, block_store_value, store_value_memory |
| 14 | mapred_hadoop, mapred, util_commit, mapred_util_commit, mapred_util, hadoop_mapred_util, commit, hadoop, util | mapred.SparkHadoopMapRedUtil: attempt_201706091116_0204_m_000033_8190: Committed | mapred, hadoop, mapred, util, commit |
| 12 | logger_start, logger, start | slf4j.Slf4jLogger: Slf4jLogger started | logger, start, logger_start |
| 0 | backend, coarse_grain, executor_coarse, executor_coarse_grain, grain, executor_backend, grain_executor_backend, coarse_grain_executor, grain_executor, executor_backend_register, backend_register, driver, register, handler, coarse | executor.CoarseGrainedExecutorBackend: Connecting to driver: spark://CoarseGrainedScheduler @10.10.34.11:59027 | coarse, grain, backend, connect, driver, executor_coarse, coarse_grain, grain_executor, executor_backend, backend_connect, connect_driver, executor_coarse_grain, coarse_grain_executor, grain_executor_backend, executor_backend_connect, backend_connect_driver |
| 0 | backend, coarse_grain, executor_coarse, executor_coarse_grain, grain, executor_backend, grain_executor_backend, coarse_grain_executor, grain_executor, executor_backend_register, backend_register, driver, register, handler, coarse | executor.CoarseGrainedExecutorBackend: Registered signal handlers for [TERM HUP INT] | coarse, grain, backend, register, signal, handler, executor_coarse, coarse_grain, grain_executor, executor_backend, backend_register, register_signal, signal_handler, executor_coarse_grain, coarse_grain_executor, grain_executor_backend, executor_backend_register, backend_register_signal, register_signal_handler |

**Figure 5.23:** Predictions BERTopic on Spark - ORIGIN+MESSAGE

At the end the best model achieves a similarity of 0.8876 with 28 topics and the high quality is reflected in the predictions. Indeed in these samples, all of them

have an acceptable quality, with topics linked to similar logs. All the input data are associated to meaningful topics, where the only issues are related to topic clusters that cover more than one possible theme. This is depicted in the last two row of Figure 5.23, where we can see how the topic is firstly made by common words belonging to the ORIGIN parts, secondly made by all the words related to the messages of different logs.

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 10 | broadcast, variable, torrent, read_broadcast_variable, read_broadcast, read, broadcast_variable, start_read_broadcast, start_read, take, variable_take, broadcast_variable_take, start | broadcast.TorrentBroadcast: Reading broadcast variable 9 took 73 ms | broadcast, torrent, broadcast, read, broadcast, variable, take, read_broadcast, broadcast_variable, variable_take, read_broadcast_variable, broadcast_variable_take |
| 10 | broadcast, variable, torrent, read_broadcast_variable, read_broadcast, read, broadcast_variable, start_read_broadcast, start_read, take, variable_take, broadcast_variable_take, start | broadcast.TorrentBroadcast: Started reading broadcast variable 9 | broadcast, torrent, broadcast, start, read, broadcast, variable, start_read, read_broadcast, broadcast_variable, start_read_broadcast, read_broadcast_variable |
| 6 | find, partition_not, partition_not_find, not_find_compute, not_find, not, find_compute, find_block, compute, cache, partition | spark.CacheManager: Partition rdd_2_6 not found, computing it | cache, partition, not, find, compute, partition_not, not_find, find_compute, partition_not_find, not_find_compute |
| 7 | store, memory, block_store, value, store_value_memory, store_value, value_memory, block_store_value, store_byte_memory, store_byte, byte_memory, block_store_byte, byte | storage.MemoryStore: Block broadcast_9_piece0 stored as bytes in memory (estimated size 5.2 KB, free 5.2 KB) | memory, store, store, byte, memory, block_store, store_byte, byte_memory, block_store_byte, store_byte_memory |
| 7 | store, memory, block_store, value, store_value_memory, store_value, value_memory, block_store_value, store_byte_memory, store_byte, byte_memory, block_store_byte, byte | storage.MemoryStore: Block broadcast_9 stored as values in memory (estimated size 8.8 KB, free 13.9 KB) | memory, store, store, value, memory, block_store, store_value, value_memory, block_store_value, store_value_memory |
| 26 | mapred, hadoop, mapred, util, commit | mapred.SparkHadoopMapRedUtil: attempt_201706091116_0204_ m_000033_8190: Committed | mapred, hadoop, mapred, util, commit |
| 21 | logger, start, logger_start | slf4j.Slf4jLogger: Slf4jLogger started | logger, start, logger_start |
| 0 | grain, backend, coarse, driver, register, connect, connect_driver, handler, driver_disconnect, register_signal_handler, register_driver, register_signal, unknown_driver_disconnect, signal, signal_handler, unknown, disconnect, command_shutdown, command | executor.CoarseGrainedExecutor Backend: Connecting to driver: spark://CoarseGrainedScheduler @10.10.34.11:59027 | coarse, grain, backend, connect, driver, connect_driver |
| 0 | grain, backend, coarse, driver, register, connect, connect_driver, handler, driver_disconnect, register_signal_handler, register_driver, register_signal, unknown_driver_disconnect, signal, signal_handler, unknown, disconnect, command_shutdown, command | executor.CoarseGrainedExecutor Backend: Registered signal handlers for [TERM HUP INT] | coarse, grain, backend, register, signal, handler, register_signal, signal_handler, register_signal_handler |

**Figure 5.24:** Predictions BERTopic on Spark - SPLIT

In SPLIT the behaviour is close to the one explained in ORIGIN+MESSAGE. In the proposed solutions, the final model, after aggregating outliers with existing clusters, provides a score similar to ORIGIN+MESSAGE. Regarding the predictions, the rebalancing of the ORIGIN does not provide any improvements for the last two rows, so the model is in line to the good model in ORIGIN+MESSAGE

| Results - Spark - SPLIT | | | |
|---|---|---|---|
| Result Type | Hyperparameters | Coherence | To drop |
| Best Coherence Model | K=18, NC=10, NN=15 | 0.8327±0.0062 | / |
| Best Model w Outliers | K=30, NC=10, NN=15 | 0.8893±0.0217 | 3 |
| Best Model w Similarity | K=28, NC=10, NN=15 | 0.8849±0.0301 | 0 |

**Table 5.22:** BERTopic Results on SPLIT Spark dataset

| Results - Spark - MESSAGE | | | |
|---|---|---|---|
| Result Type | Hyperparameters | Coherence | To drop |
| Best Coherence Model | K=31, NC=10, NN=15 | 0.9618±0.0226 | / |
| Best Model w Outliers | K=41, NC=10, NN=15 | 0.9670±0.0189 | 3 |
| Best Model w Similarity | K=38, NC=10, NN=15 | 0.9660±0.0213 | 0 |

**Table 5.23:** BERTopic Results on MESSAGE Spark dataset

The almost optimal situation is presented also in the MESSAGE dataset, with a really high coherence, linked to a final number of topic equal to 38. About this high value of coherence is needed a note; in many cases an high $CV$ score indicates a perfect match, i.e that the topic model has created a cluster for each input sentence. This usually should be avoided, however, some considerations are needed depending on the situation. Indeed, only with a human check the real quality of the associations done can be inferred. For example, in the context of different messages, an high score is a sign that the models has been able to find many appropriate clusters, rather that making poor generalizations. In terms of coherence score, this is represented by the fact that the bi-grams and tri-grams are usually linked to one or few sentences, so if they are matched between clusters and sentences, the final score increases. The clusters made by this model have an high quality and in many cases they are able to generalize similar topics. However this i not always the case, as shown in the $4^{TH}$ and $5^{TH}$ rows. The models have made two different topics for two almost identical logs.
On this behaviour some reasonings are needed. Indeed, this higher level of discrimination is object of interest in many cases where a topic model provides good value of coherence. Several discussions are done in the research field, where many asserts that is better to provide more generic topic, avoiding over-fitting. At the same time it is important to take into account the final goal and what will be done on the outputs produced. Depending on it, an higher level of precision can be useful, taking always into account to avoid making perfect matches between topic and sentences. For this project I decide to follow this last reasoning, since I want really

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 3 | variable, read_broadcast_variable, read_broadcast, read, broadcast_variable, broadcast take variable_take broadcast_variable_take start_read_broadcast start_read start | broadcast.TorrentBroadcast: Reading broadcast variable 9 took 73 ms | read, broadcast, variable, take, read_broadcast, broadcast_variable, variable_take, read_broadcast_variable, broadcast_variable_take |
| 3 | variable, read_broadcast_variable, read_broadcast, read, broadcast_variable, broadcast take variable_take broadcast_variable_take start_read_broadcast start_read start | broadcast.TorrentBroadcast: Started reading broadcast variable 9 | start, read, broadcast, variable, start_read, read_broadcast, broadcast_variable, start_read_broadcast, read_broadcast_variable |
| 8 | partition_not, partition_not_find, not_find_compute, not_find, not, find_compute, compute, find, partition | spark.CacheManager: Partition rdd_2_6 not found, computing it | partition, not, find, compute, partition_not, not_find, find_compute, partition_not_find, not_find_compute |
| 25 | store_byte, byte_memory, block_store_byte, store_byte_memory, byte, block_store, store, memory, block | storage.MemoryStore: Block broadcast_9_piece0 stored as bytes in memory (estimated size 5.2 KB, free 5.2 KB) | block, store, byte, memory, block_store, store_byte, byte_memory, block_store_byte, store_byte_memory |
| 26 | value, store_value_memory, store_value, block_store_value, value_memory, block_store, store, memory, block | storage.MemoryStore: Block broadcast_9 stored as values in memory (estimated size 8.8 KB, free 13.9 KB) | block, store, value, memory, block_store, store_value, value_memory, block_store_value, store_value_memory |
| 32 | attempt, commit, attempt_commit | mapred.SparkHadoopMapRedUtil: attempt_201706091116_0204_m_000033_8190: Committed | attempt, commit, attempt_commit |
| 35 | logger, start, logger_start | slf4j.Slf4jLogger: Slf4jLogger started | logger, start, logger_start |
| 38 | connect, driver, connect_driver | executor.CoarseGrainedExecutor Backend: Connecting to driver: spark://CoarseGrainedScheduler @10.10.34.11:59027 | connect, driver, connect_driver |
| 6 | signal, register_signal_handler, register_signal, signal_handler, handler, register | executor.CoarseGrainedExecutor Backend: Registered signal handlers for [TERM HUP INT] | register, signal, handler, register_signal, signal_handler, register_signal_handler |

**Figure 5.25:** Predictions BERTopic on Spark - MESSAGE

precise topics when needed due to the future application in the anomaly detection field.

### HDFS

The results obtained in HDFS follows the same trend of Spark. BERTopic also in this case is able to provide good models that creates clusters which cover almost all logs properly.

On ORIGIN+MESSAGE, after applying the second solution, the final number of topic is equal to 72, a good measure for HDSF. Also the topics have a better quality and it is underlined by the high coherence score. This is translated into

| Results - HDFS - ORIGIN+MESSAGE | | | |
|---|---|---|---|
| Result Type | Hyperparameters | Coherence | To drop |
| Best Coherence Model | K=48, NC=10, NN=9 | 0.8414±0.0142 | / |
| Best Model w Outliers | K=77, NC=10, NN=9 | 0.8904±0.0257 | 5 |
| Best Model w Similarity | K=72, NC=10, NN=9 | 0.8856±0.0098 | 0 |

**Table 5.24:** BERTopic Results on ORIGIN+MESSAGE HDFS dataset

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 20 | io_eof_exception, io_eof, java_io_eof, eof_exception, eof, java_io, cause_java_io, io, cause_java, cause, java, exception | Caused by java.io.EOFException | cause, java, io, eof, exception, cause_java, java_io, io_eof, eof_exception, cause_java_io, java_io_eof, io_eof_exception |
| 19 | io, io_exception, reset_peer,java_io exception_connection_reset, peer, reset, exception_connection, io_exception_connection, connection_reset_peer, connection_reset, connection, host, exception, cause_java_io… | Caused by java.io.IOException: Connection reset by peer | cause, java, io, io, exception, connection, reset, peer, cause_java, java_io, io_exception, exception_connection, connection_reset, reset_peer, cause_java_io, io_exception_connection, exception_connection_reset, connection_reset_peer |
| 20 | io_eof_exception, io_eof, java_io_eof, eof_exception, eof, java_io, cause_java_io, io, cause_java, cause, java, exception | java.io.EOFException | java, io, eof, exception, java_io, io_eof, eof_exception, java_io_eof, io_eof_exception |
| 72 | fsdataset, impl, total, time, add, replica, map, datanode_fsdataset, fsdataset_impl, impl_total, total_time, time_add, add_replica, replica_map, datanode_fsdataset_impl, fsdataset_impl_total, total_time_add, time_add_replica… | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Total time to add all replicas to map: 0ms | fsdataset, impl, fsdataset, impl, total, time, add, replica, map, datanode_fsdataset, fsdataset_impl, fsdataset_impl, fsdataset_impl, impl_total, total_time, time_add, add_replica, replica_map, datanode_fsdataset_impl, fsdataset_impl_total, impl_total_time, total_time_add, time_add_replica, add_replica_map |
| 71 | fsdataset, impl, total, time, scan, replica, pool, datanode_fsdataset, fsdataset_impl, impl_total, total_time, time_scan, scan_replica, replica_block, block_pool, datanode_fsdataset_impl, fsdataset_impl_total, impl_total_time… | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Total time to scan all replicas for block pool BP-108841162-10.10.34.11-1440074360971: 8ms | fsdataset, impl, fsdataset, impl, total, time, scan, replica, pool, datanode_fsdataset, fsdataset_impl, fsdataset_impl, fsdataset_impl, impl_total, total_time, time_scan, scan_replica, replica_block, block_pool, datanode_fsdataset_impl, fsdataset_impl_total, impl_total_time, total_time_scan, time_scan_replica, scan_replica_block, replica_block_pool |
| 3 | mortbay, mortbay_start, mortbay_log, log, start | org.mortbay.log: jetty-6.1.26 | mortbay |
| 61 | mortbay, log, mortbay_log | org.mortbay.log: Logging to org.slf4j.impl.Log4jLoggerAdapter (org.mortbay.log) via org.mortbay.log.Slf4jLog | mortbay, log, mortbay_log |
| 0 | fsdataset_impl, pool_volume, block_pool_volume, fsdataset, impl, fsdataset_impl_time, impl_time, replica_map, add_replica, map, add_replica_map, map_block_pool, replica_map_block, map_block, datanode_fsdataset_impl, datanode_fsdataset, replica, time, time_add_replica, time_add | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Time to add replicas to map for block pool BP-108841162-10.10.34.11-1440074360971 on volume /tmp/hadoop-hdfs/dfs/data/current: 0ms | fsdataset, impl, fsdataset, impl, time, add, replica, map, pool, volume, datanode_fsdataset, fsdataset_impl, fsdataset_impl, fsdataset_impl, impl_time, time_add, add_replica, replica_map, map_block, block_pool, pool_volume, datanode_fsdataset_impl, fsdataset_impl_time, impl_time_add, time_add_replica, add_replica_map, replica_map_block, map_block_pool, block_pool_volume |
| 0 | fsdataset_impl, pool_volume, block_pool_volume, fsdataset, impl, fsdataset_impl_time, impl_time, replica_map, add_replica, map, add_replica_map, map_block_pool, replica_map_block, map_block, datanode_fsdataset_impl, datanode_fsdataset, replica, time, time_add_replica, time_add | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Time taken to scan block pool BP-108841162-10.10.34.11-1440074360971 on /tmp/hadoop-hdfs/dfs/data/current: 7ms | fsdataset, impl, fsdataset, impl, time, take, scan, pool, datanode_fsdataset, fsdataset_impl, fsdataset_impl, fsdataset_impl, impl_time, time_take, take_scan, scan_block, block_pool, datanode_fsdataset_impl, fsdataset_impl_time, impl_time_take, time_take_scan, take_scan_block, scan_block_pool |

**Figure 5.26:** Predictions BERTopic on HDFS - ORIGIN+MESSAGE

predictions that are excellent for many situation, even the hardest ones. However, this is not always the case, like depicted in the last two rows of Figure 5.26. Indeed the first one of the two is correctly associated, while the second one is associated

to the same topic mainly for the common origin, making it not acceptable.

| Results - HDFS - SPLIT | | | |
|---|---|---|---|
| Result Type | Hyperparameters | Coherence | To drop |
| Best Coherence Model | K=50, NC=10, NN=14 | 0.8152±0.0196 | / |
| Best Model w Outliers | K=78, NC=10, NN=14 | 0.8769±0.0284 | 5 |
| Best Model w Similarity | K=73, NC=10, NN=14 | 0.8649±0.0201 | 0 |

**Table 5.25:** BERTopic Results on SPLIT HDFS dataset

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 15 | io_eof_exception, io_eof, java_io_eof, eof_exception, eof, java_io, cause, io, java, exception | Caused by java.io.EOFException | cause, java, io, eof, exception, java_io, io_eof, eof_exception, java_io_eof, io_eof_exception |
| 14 | io, io_exception_connection, connection, connection_reset, connection_reset_peer, reset, reset_peer, peer, exception_connection, exception_connection_reset, host, io_exception, local, java_io, exception, host_destination, javal | Caused by java.io.IOException: Connection reset by peer | cause, java, io, io, exception, connection, reset, peer, java_io, io_exception, exception_connection, connection_reset, reset_peer, io_exception_connection, exception_connection_reset, connection_reset_peer |
| 15 | io_eof_exception, io_eof, java_io_eof, eof_exception, eof, java_io, cause, io, java, exception | java.io.EOFException | java, io, eof, exception, java_io, io_eof, eof_exception, java_io_eof, io_eof_exception |
| 1 | pool_volume, block_pool_volume, fsdataset, replica_map_block, map_block_pool, map_block, impl, add_replica_map, replica_map, map, add_replica, replica, add, pool, block_pool, add_block_pool, add_block, time_add… | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Total time to add all replicas to map: 0ms | fsdataset, impl, fsdataset, impl, total, time, add, replica, map, total_time, time_add, add_replica, replica_map, total_time_add, time_add_replica, add_replica_map |
| 33 | time_scan_replica, time_scan, scan_replica_block, scan_replica, replica_block_pool, replica_block, total_time_scan, total_time, total, replica, time, scan, fsdataset, impl, pool, block_pool | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Total time to scan all replicas for block pool BP-108841162-10.10.34.11-1440074360971: 8ms | fsdataset, impl, fsdataset, impl, total, time, scan, replica, pool, total_time, time_scan, scan_replica, replica_block, block_pool, total_time_scan, time_scan_replica, scan_replica_block, replica_block_pool |
| 9 | mortbay, mortbay_start, mortbay_log, log, start | org.mortbay.log: jetty-6.1.26 | mortbay |
| 73 | mortbay, log, mortbay_log | org.mortbay.log: Logging to org.slf4j.impl.Log4jLoggerAdapter (org.mortbay.log) via org.mortbay.log.Slf4jLog | mortbay, log, mortbay_log |
| 1 | pool_volume, block_pool_volume, fsdataset, replica_map_block, map_block_pool, map_block, impl, add_replica_map, replica_map, map, add_replica, replica, add, pool, block_pool, add_block_pool, add_block, time_add… | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Time to add replicas to map for block pool BP-108841162-10.10.34.11-1440074360971 on volume /tmp/hadoop-hdfs/dfs/data/current: 0ms | fsdataset, impl, fsdataset, impl, time, add, replica, map, pool, volume, time_add, add_replica, replica_map, map_block, block_pool, pool_volume, time_add_replica, add_replica_map, replica_map_block, map_block_pool, block_pool_volume |
| 25 | time_take, take_scan_block, take_scan, time_take_scan, scan_block_pool, scan_block, take, time, scan, fsdataset, impl, pool, block_pool | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Time taken to scan block pool BP-108841162-10.10.34.11-1440074360971 on /tmp/hadoop-hdfs/dfs/data/current: 7ms | fsdataset, impl, fsdataset, impl, time, take, scan, pool, time_take, take_scan, scan_block, block_pool, time_take_scan, take_scan_block, scan_block_pool |

**Figure 5.27:** Predictions BERTopic on HDFS - SPLIT

In this situation, instead, the modifications done on SPLIT provide better results for what concern the predictions. Indeed, although the coherence is lower with respect to the one of ORIGIN+MESSAGE, the lower impact of the ORIGIN part in this case helps for the last two rows, here associated to topics which represent properly both themes.

| Results - HDFS - MESSAGE | | | |
|---|---|---|---|
| Result Type | Hyperparameters | Coherence | To drop |
| Best Coherence Model | K=44, NC=10, NN=9 | 0.8153±0.0121 | / |
| Best Model w Outliers | K=69, NC=10, NN=9 | 0.8669±0.0087 | 5 |
| Best Model w Similarity | K=64, NC=10, NN=9 | 0.8601±0.0129 | 0 |

**Table 5.26:** BERTopic Results on MESSAGE HDFS dataset

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 14 | io, host, connection, connection_reset_peer, reset, peer, connection_reset, io_exception_connection, exception_connection, io_exception, local, java_ioexception_connection_reset, reset_peer, | Caused by java.io.IOException: Connection reset by peer | java, io, eof, exception, java_io, io_eof, eof_exception, java_io_eof, io_eof_exception |
| 25 | io_eof_exception, io_eof, java_io_eof, eof_exception, eof, java_io, io, java, exception | java.io.EOFException | java, io, io, exception, connection, reset, peer, java_io, io_exception, exception_connection, connection_reset, reset_peer, io_exception_connection, exception_connection_reset, connection_reset_peer |
| 3 | block_miss, miss, total, miss_metadata, total_time, replica, time, total_time_add, pool_total_block, pool_total, miss_block_mismatch, metadata_miss_block, mismatch, metadata, block_pool_total, block_miss_metadata… | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Total time to add all replicas to map: 0ms | total, time, add, replica, map, total_time, time_add, add_replica, replica_map, total_time_add, time_add_replica, add_replica_map |
| 3 | block_miss, miss, total, miss_metadata, total_time, replica, time, total_time_add, pool_total_block, pool_total, miss_block_mismatch, metadata_miss_block, mismatch, metadata, block_pool_total, block_miss_metadata… | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Total time to scan all replicas for block pool BP-108841162-10.10.34.11-1440074360971: 8ms | total, time, scan, replica, pool, total_time, time_scan, scan_replica, replica_block, block_pool, total_time_scan, time_scan_replica, scan_replica_block, replica_block_pool |
| 28 | jetty | org.mortbay.log: jetty-6.1.26 | jetty |
| 52 | log | org.mortbay.log: Logging to org.slf4j.impl.Log4jLoggerAdapter (org.mortbay.log) via org.mortbay.log.Slf4jLog | log |
| 7 | block_pool_volume, pool_volume, replica_map_block, map_block_pool, map_block, map, add_replica_map, add_replica, replica_map, replica, volume, pool, block_pool, scan_block, scan_block_pool, add, time_add, time_add_replica, scan, time | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Time to add replicas to map for block pool BP-108841162-10.10.34.11-1440074360971 on volume /tmp/hadoop-hdfs/dfs/data/current: 0ms | time, add, replica, map, pool, volume, time_add, add_replica, replica_map, map_block, block_pool, pool_volume, time_add_replica, add_replica_map, replica_map_block, map_block_pool, block_pool_volume |
| 58 | time, take, scan, pool, time_take, take_scan, scan_block, block_pool, time_take_scan, take_scan_block, scan_block_pool | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Time taken to scan block pool BP-108841162-10.10.34.11-1440074360971 on /tmp/hadoop-hdfs/dfs/data/current: 7ms | time, take, scan, pool, time_take, take_scan, scan_block, block_pool, time_take_scan, take_scan_block, scan_block_pool |

**Figure 5.28:** Predictions BERTopic on HDFS - MESSAGE

Regarding MESSAGE version, instead the drawbacks are the same expressed in ORIGIN+MESSAGE. Both score and value of $K$ are good, but on the third and fourth rows we can see how the models still generalize some cases.

Overall, the application of BERTopic in situations represented in project is not straightforward. The solution proposed by me help to overcome a big problem related to HDBSCAN, making BERTopic eligible for real case utilization.

### 5.3.6 GEAC

My custom model has been developed in three principal parts. The first one concerns the transformation of textual data into numerical vectors. This operation is done by the concatenation of three optional components. The first one are the output probabilities coming from NMF or LDA. Differently from what implemented in their section, they are not trained exploiting all the possible variable, since otherwise the training phase would have been incumbent. At the same time, I decide to not use the best parameters found since my goal is to obtain a fast model from scratch, which is able to generalize the output. The second and third components regard the extraction of useful information by means of Word Embedding. The chosen algorithms are the ones already presented, BERT, where the pre-trained model with the best performance is *all-mpnet-base-v2*, and Word2Vec, extended from the Google pre-trained model. The goal is to combine these three components in order to improve each of them.
Once data are vectorized, the final dimensionality is in the order of thousands. To avoid possible issues related to the curse of dimensionality, I use an AutoEncoder for this second part. It is trained on 0-1 scaled data with a number of epochs equal to 100, by means of the ADAM optimizer and the binary cross-entropy loss. The architecture is composed by the stack of the encoder and of the decoder, where the latent dimension is being searched in the range $[32, 64, 96, 128]$.
In the last part the clustering algorithm is implemented. After standardizing the data, an algorithm between K-Means and K-Medoids is chosen. About the first, the model is taken from scikit-learn [61] and it is initialized with *k-means++*, the second one, implemented from [62]scikit-learn-extra, uses the *k-medoids++* initialization combined with the *pam* method, to obtain more accurate predictions.

#### Spark

On table 5.27 the results coming from different combinations are represented. The workflow is simple: I start with the simpler model of Words Embedding, then,

for each of them, I do different combinations with both other Words Embedding models and Classical algorithms. The coherence values are in general higher than BERTopic and, as consequence, the best seen so far. In detail, the best model on ORIGIN+MESSAGE has 33 topics, in line with my expectations, and it is made with BERT, LDA and Word2Vec. It is obtained with K-Medoids, while a dimensionality of 64 is chosen for the AutoEncoder.
A particularity can be noticed on K-Medoids results. Indeed, the application of LDA, rather than NMF, produces higher scores, showing the different contribution provided. The high quality is depicted in the predictions: improvements are consistent, however, minor issues could still happen. Some clusters skewed towards the origin, indeed, the drawback is the higher relevance on the words coming from the origin than a bad quality of topics, since the relevant words are always found.

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 7 | broadcast, broadcast_torrent, broadcast_read, torrent, read, variable, broadcast_variable, read_broadcast_variable, take, variable_take, torrent_broadcast_read, broadcast_variable_take, start, broadcast_start, start_read… | broadcast.TorrentBroadcast: Reading broadcast variable 9 took 73 ms | broadcast, torrent, broadcast, read, broadcast, variable, take, broadcast_torrent, broadcast_torrent, broadcast_read, broadcast_read, broadcast_variable, variable_take, torrent_broadcast_read, read_broadcast_variable, broadcast_variable_take |
| 7 | broadcast, broadcast_torrent, broadcast_read, torrent, read, variable, broadcast_variable, read_broadcast_variable, take, variable_take, torrent_broadcast_read, broadcast_variable_take, start, broadcast_start, start_read… | broadcast.TorrentBroadcast: Started reading broadcast variable 9 | broadcast, torrent, broadcast, start, read, broadcast, variable, broadcast_torrent, broadcast_torrent, broadcast_start, start_read, broadcast_read, broadcast_variable, torrent_broadcast_start, broadcast_start_read, start_read_broadcast, read_broadcast_variable |
| 20 | cache, partition, not, find, compute, cache_manager, manager_partition, partition_not, not_find, find_compute, cache_manager_partition, manager_partition_not, partition_not_find, not_find_compute | spark.CacheManager: Partition rdd_2_6 not found computing it | cache, partition, not, find, compute, cache_manager, manager_partition, partition_not, not_find, find_compute, cache_manager_partition, manager_partition_not, partition_not_find, not_find_compute |
| 0 | memory, store, store_block, byte, storage_memory, memory_store, store_byte, byte_memory, storage_memory_store, memory_store_block, block_store_byte, store_byte_memory | storage.MemoryStore: Block broadcast_9 stored as values in memory (estimated size 8.8 KB free 13.9 KB) | memory, store, store, value, memory, storage_memory, memory_store, store_block, store_block, store_value, value_memory, storage_memory_store, memory_store_block, block_store_value, store_value_memory |
| 29 | memory, store, memory_store, storage_memory, store_block, storage_memory_store, value, store_value, value_memory, memory_store_block, block_store_value, store_value_memory, start, capacity, store_start, start_capacity… | storage.MemoryStore: Block broadcast_9_piece0 stored as bytes in memory (estimated size 5.2 KB free 5.2 KB) | memory, store, store, byte, memory, storage_memory, memory_store, store_block, store_block, store_byte, byte_memory, storage_memory_store, memory_store_block, block_store_byte, store_byte_memory |
| 24 | mapred, mapred_hadoop, hadoop, util, commit, mapred_util, util_commit, hadoop_mapred_util, mapred_util_commit | mapred.SparkHadoopMapRedUtil: attempt_201706091116_0204 _m_000033_8190: Committed | mapred, hadoop, mapred, util, commit, mapred_hadoop, mapred_hadoop, mapred_util, util_commit, hadoop_mapred_util, mapred_util_commit |
| 10 | logger, start, logger_start | slf4j.Slf4jLogger: Slf4jLogger started | logger, start, logger_start |
| 31 | coarse, grain, backend, driver, executor_coarse, coarse_grain, grain_executor, executor_backend, executor_coarse_grain, coarse_grain_executor, grain_executor_backend, connect, backend_connect, connect_driver… | executor.CoarseGrainedExecutor Backend: Connecting to driver: spark://CoarseGrainedScheduler @10.10.34.11:59027 | coarse, grain, backend, connect, driver, executor_coarse, coarse_grain, grain_executor, executor_backend, backend_connect, connect_driver, executor_coarse_grain, coarse_grain_executor, grain_executor_backend, executor_backend_connect, backend_connect_driver |
| 2 | coarse, grain, backend, register, signal, handler, executor_coarse, coarse_grain, grain_executor, executor_backend, backend_register, register_signal, signal_handler, executor_coarse_grain, coarse_grain_executor… | executor.CoarseGrainedExecutor Backend: Registered signal handlers for [TERM HUP INT] | coarse, grain, backend, register, signal, handler, executor_coarse, coarse_grain, grain_executor, executor_backend, backend_register, signal_handler, executor_coarse_grain, coarse_grain_executor, grain_executor_backend, executor_backend_register, backend_register_signal, register_signal_handler,register_signal |

**Figure 5.29:** Predictions GEAC on Spark - ORIGIN+MESSAGE

| Results - Spark - ORIGIN+MESSAGE | | | | |
|---|---|---|---|---|
| Model | Cluster | K | AE dim | Coherence |
| BERT+LDA+W2V | K-Means | 31 | 96 | 0.9359±0.0153 |
| BERT+NMF+W2V | K-Means | 32 | 64 | 0.9263±0.0205 |
| BERT+W2V | K-Means | 29 | 128 | 0.9328±0.0238 |
| BERT+LDA | K-Means | 31 | 64 | 0.9342±0.0174 |
| W2V+LDA | K-Means | 30 | 128 | 0.9149±0.0169 |
| BERT+NMF | K-Means | 28 | 96 | 0.9116±0.0302 |
| W2V+NMF | K-Means | 29 | 64 | 0.9091±0.0271 |
| BERT | K-Means | 30 | 96 | 0.9228±0.0214 |
| W2V | K-Means | 29 | 96 | 0.9218±0.0092 |
| **BERT+LDA+W2V** | **K-Medoids** | **33** | **64** | **0.9554±0.0149** |
| BERT+NMF+W2V | K-Medoids | 28 | 64 | 0.9092±0.0317 |
| BERT+W2V | K-Medoids | 31 | 128 | 0.9461±0.0301 |
| BERT+LDA | K-Medoids | 31 | 64 | 0.9472±0.0154 |
| W2V+LDA | K-Medoids | 23 | 128 | 0.8084±0.0221 |
| BERT+NMF | K-Medoids | 25 | 64 | 0.8954±0.0294 |
| W2V+NMF | K-Medoids | 24 | 32 | 0.8399±0.0266 |
| BERT | K-Medoids | 28 | 64 | 0.9214±0.0301 |
| W2V | K-Medoids | 28 | 96 | 0.9258±0.0144 |

**Table 5.27:** GEAC Results on ORIGIN+MESSAGE Spark dataset

On the SPLIT dataset the best result is comparable with ORIGIN+MESSAGE. They both share the same value of $K$ and K-Medoids built on BERT, NMF and Word2Vec combination of elements, while the dimension for the AutoEncoder is fixed to 128. Nevertheless, the coherence score is lower than before, but it does not have repercussions on the predictions: the model seems to provide more meaningful topics. Regarding this difference in values, my consideration is that for very high scores, the predictions are in general very good and they are different only for few details. In this context, the data processing makes the difference. Indeed, the less weight, given to the origin, in this case can overcome the problems highlighted in the previous analysis.

| Results - Spark - SPLIT | | | | |
|---|---|---|---|---|
| Model | Cluster | K | AE dim | Coherence |
| BERT+LDA+W2V | K-Means | 33 | 128 | 0.9224±0.0136 |
| BERT+NMF+W2V | K-Means | 32 | 128 | 0.9239±0.0210 |
| BERT+W2V | K-Means | 33 | 64 | 0.9101±0.0197 |
| BERT+LDA | K-Means | 29 | 96 | 0.9172±0.0154 |
| W2V+LDA | K-Means | 28 | 96 | 0.8067±0.0299 |
| BERT+NMF | K-Means | 32 | 32 | 0.9167±0.0291 |
| W2V+NMF | K-Means | 32 | 128 | 0.9187±0.0183 |
| BERT | K-Means | 33 | 32 | 0.9173±0.0174 |
| W2V | K-Means | 31 | 128 | 0.9094±0.0166 |
| BERT+LDA+W2V | K-Medoids | 33 | 96 | 0.9219±0.0195 |
| **BERT+NMF+W2V** | **K-Medoids** | **32** | **128** | **0.9242±0.0134** |
| BERT+W2V | K-Medoids | 30 | 128 | 0.8948±0.0092 |
| BERT+LDA | K-Medoids | 28 | 68 | 0.9016±0.0162 |
| W2V+LDA | K-Medoids | 28 | 128 | 0.8380±0.0312 |
| BERT+NMF | K-Medoids | 32 | 128 | 0.9123±0.0244 |
| W2V+NMF | K-Medoids | 32 | 96 | 0.9166±0.0213 |
| BERT | K-Medoids | 32 | 128 | 0.9123±0.0199 |
| W2V | K-Medoids | 32 | 128 | 0.9150±0.0204 |

**Table 5.28:** GEAC Results on SPLIT Spark dataset

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 4 | broadcast, torrent, read, variable, read_broadcast, broadcast_variable, read_broadcast_variable, take, variable_take, broadcast_variable_take, start start_read, start_read_broadcast | broadcast.TorrentBroadcast: Reading broadcast variable 9 took 73 ms | broadcast, torrent, broadcast, read, broadcast, variable, take, read_broadcast, broadcast_variable, variable_take, read_broadcast_variable, broadcast_variable_take |
| 4 | broadcast, torrent, read, variable, read_broadcast, broadcast_variable, read_broadcast_variable, take, variable_take, broadcast_variable_take, start start_read, start_read_broadcast | broadcast.TorrentBroadcast: Started reading broadcast variable 9 | broadcast, torrent, broadcast, start, read, broadcast, variable, start_read, read_broadcast, broadcast_variable, start_read_broadcast, read_broadcast_variable |
| 15 | cache, partition, not, find, compute, partition_not, not_find, find_compute, partition_not_find, not_find_compute | spark.CacheManager: Partition rdd_2_6 not found, computing it | cache, partition, not, find, compute, partition_not, not_find, find_compute, partition_not_find, not_find_compute |
| 26 | memory, store, block_store, byte, store_byte, byte_memory, block_store_byte, store_byte_memory, value, store_value, block_store_value, value_memory, store_value_memory | storage.MemoryStore: Block broadcast_9_piece0 stored as bytes in memory (estimated size 5.2 KB, free 5.2 KB) | memory, store, store, byte, memory, block_store, store_byte, byte_memory, block_store_byte, store_byte_memory |
| 26 | memory, store, block_store, byte, store_byte, byte_memory, block_store_byte, store_byte_memory, value, store_value, block_store_value, value_memory, store_value_memory | storage.MemoryStore: Block broadcast_9 stored as values in memory (estimated size 8.8 KB, free 13.9 KB) | memory, store, store, value, memory, block_store, store_value, value_memory, block_store_value, store_value_memory |
| 22 | mapred, hadoop, util, commit | mapred.SparkHadoopMapRedUtil: attempt_201706091116_0204_ m_000033_8190: Committed | mapred, hadoop, mapred, util, commit |
| 10 | logger, start, logger_start | slf4j.Slf4jLogger: Slf4jLogger started | logger, start, logger_start |
| 0 | coarse, grain, backend, driver, connect, connect_driver, register, register_driver | executor.CoarseGrainedExecutor Backend: Connecting to driver: spark://CoarseGrainedScheduler @10.10.34.11:59027 | coarse, grain, backend, connect, driver, connect_driver |
| 31 | coarse, grain, backend, register, signal, handler, register_signal, signal_handler, register_signal_handler | executor.CoarseGrainedExecutor Backend: Registered signal handlers for [TERM HUP INT] | coarse, grain, backend, register, signal, handler, register_signal, signal_handler, register_signal_handler |

**Figure 5.30:** Predictions GEAC on Spark - SPLIT

For what concerns the MESSAGE dataset, the best solution is obtained with 39 topics, K-Medoids and 96 dimensions for the AutoEncoder. As in the previous analysis, this time the best model is still made by BERT, Word2Vec and NMF. The predictions are good, but in some cases the model seems that it makes distinctions with a too fine granularity, resulting in distinct topics when it could be only one.

| Results - Spark - MESSAGE | | | | |
|---|---|---|---|---|
| Model | Cluster | K | AE dim | Coherence |
| BERT+LDA+W2V | K-Means | 37 | 32 | 0.9564±0.0114 |
| BERT+NMF+W2V | K-Means | 39 | 64 | 0.9665±0.0165 |
| BERT+W2V | K-Means | 39 | 64 | 0.9552±0.0165 |
| BERT+LDA | K-Means | 38 | 64 | 0.9489±0.0203 |
| W2V+LDA | K-Means | 37 | 32 | 0.9214±0.0109 |
| BERT+NMF | K-Means | 40 | 64 | 0.9605±0.0116 |
| W2V+NMF | K-Means | 40 | 64 | 0.9478±0.0226 |
| BERT | K-Means | 39 | 128 | 0.9552±0.0286 |
| W2V | K-Means | 37 | 96 | 0.9589±0.0341 |
| BERT+LDA+W2V | K-Medoids | 38 | 96 | 0.9631±0.0110 |
| **BERT+NMF+W2V** | **K-Medoids** | **39** | **96** | **0.9715±0.0118** |
| BERT+W2V | K-Medoids | 38 | 128 | 0.9529±0.0273 |
| BERT+LDA | K-Medoids | 38 | 32 | 0.9598±0.0088 |
| W2V+LDA | K-Medoids | 37 | 128 | 0.9187±0.0158 |
| BERT+NMF | K-Medoids | 37 | 128 | 0.9600±0.0317 |
| W2V+NMF | K-Medoids | 37 | 64 | 0.9460±0.0302 |
| BERT | K-Medoids | 37 | 128 | 0.9516±0.0192 |
| W2V | K-Medoids | 37 | 64 | 0.9446±0.0185 |

**Table 5.29:** GEAC Results on MESSAGE Spark dataset

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 6 | read, broadcast, variable, read_broadcast, broadcast_variable, read_broadcast_variable, take, variable_take, broadcast_variable_take, start, start_read, start_read_broadcast | broadcast.TorrentBroadcast: Reading broadcast variable 9 took 73 ms | read, broadcast, variable, take, read_broadcast, broadcast_variable, variable_take, read_broadcast_variable, broadcast_variable_take |
| 6 | read, broadcast, variable, read_broadcast, broadcast_variable, read_broadcast_variable, take, variable_take, broadcast_variable_take, start, start_read, start_read_broadcast | broadcast.TorrentBroadcast: Started reading broadcast variable 9 | start, read, broadcast, variable, start_read, read_broadcast, broadcast_variable, start_read_broadcast, read_broadcast_variable |
| 18 | partition, not, find, compute, partition_not, not_find, find_compute, partition_not_find, not_find_compute | spark.CacheManager: Partition rdd_2_6 not found, computing it | partition, not, find, compute, partition_not, not_find, find_compute, partition_not_find, not_find_compute |
| 34 | block, store, byte, memory, block_store, store_byte, byte_memory, block_store_byte, store_byte_memory | storage.MemoryStore: Block broadcast_9_piece0 stored as bytes in memory (estimated size 5.2 KB, free 5.2 KB) | block, store, byte, memory, block_store, store_byte, byte_memory, block_store_byte, store_byte_memory |
| 2 | block, store, value, memory, block_store, store_value, value_memory, block_store_value, store_value_memory | storage.MemoryStore: Block broadcast_9 stored as values in memory (estimated size 8.8 KB, free 13.9 KB) | block, store, value, memory, block_store, store_value, value_memory, block_store_value, store_value_memory |
| 32 | attempt, commit, attempt_commit | mapred.SparkHadoopMapRedUtil: attempt_201706091116_0204_m_000033_8190: Committed | attempt, commit, attempt_commit |
| 16 | logger, start, logger_start | slf4j.Slf4jLogger: Slf4jLogger started | logger, start, logger_start |
| 38 | connect, driver, connect_driver | executor.CoarseGrainedExecutorBackend: Connecting to driver: spark://CoarseGrainedScheduler@10.10.34.11:59027 | connect, driver, connect_driver |
| 10 | register, signal, handler, register_signal, signal_handler, register_signal_handler | executor.CoarseGrainedExecutorBackend: Registered signal handlers for [TERM HUP INT] | register, signal, handler, register_signal, signal_handler, register_signal_handler |

**Figure 5.31:** Predictions GEAC on Spark - MESSAGE

The conclusion on Spark is that the K-Medoids clustering algorithm seems to provide better results, helping to overcome the low dimensionality of the input data.

### HDFS

The versions obtained on HDFS dataset re-affirm the goodness of the algorithm, already observed in Spark. The coherence values are in general better than those obtained with BERTopic and this is represented in the predictions. In fact, they are in general comparable or better, resulting as the best among the models analyzed in this thesis.

On ORIGIN+MESSAGE the best model is obtained with 75 topics on the combination made by BERT, NMF and Word2Vec, based on K-Means clustering. The predictions are comparable to the ones obtained on BERTopic. On the $4^{TH}$ and $5^{TH}$ rows GEAC assigns the logs to the same cluster, making this decision acceptable. Indeed, differently from BERTopic, this time the topic cluster is more coherent, with words coming from both sentences, and inside the same cluster the secondo to last log, since the theme is shared. On the last log of the table, the performance is better, with a topic more aligned with the theme expressed in the raw sentence, although it could have been included in the same cluster of the $5^{TH}$ due to the common words.

| Results - HDFS - ORIGIN+MESSAGE | | | | |
|---|---|---|---|---|
| Model | Cluster | K | AE dim | Coherence |
| BERT+LDA+W2V | KMeans | 73 | 64 | 0.9457±0.0184 |
| **BERT+NMF+W2V** | **KMeans** | **75** | **96** | **0.9509±0.0080** |
| BERT+W2V | KMeans | 71 | 64 | 0.9353±0.0313 |
| BERT+LDA | KMeans | 73 | 64 | 0.9478±0.0099 |
| W2V+LDA | KMeans | 65 | 128 | 0.8803±0.0174 |
| BERT+NMF | KMeans | 75 | 128 | 0.9427±0.0082 |
| W2V+NMF | KMeans | 67 | 32 | 0.9072±0.0310 |
| BERT | KMeans | 70 | 64 | 0.9259±0.0196 |
| W2V | KMeans | 73 | 96 | 0.9355±0.0269 |
| BERT+LDA+W2V | KMedoids | 77 | 96 | 0.9468±0.0264 |
| BERT+NMF+W2V | KMedoids | 72 | 96 | 0.9311±0.0149 |
| BERT+W2V | KMedoids | 71 | 128 | 0.9368±0.0119 |
| BERT+LDA | KMedoids | 76 | 96 | 0.9361±0.0232 |
| W2V+LDA | KMedoids | 72 | 64 | 0.9389±0.0080 |
| BERT+NMF | KMedoids | 72 | 64 | 0.9311±0.0219 |
| W2V+NMF | KMedoids | 74 | 64 | 0.9348±0.0268 |
| BERT | KMedoids | 63 | 32 | 0.7923±0.0234 |
| W2V | KMedoids | 70 | 128 | 0.9379±0.0286 |

**Table 5.30:** GEAC Results on ORIGIN+MESSAGE HDFS dataset

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 21 | java, io, eof, exception, java_io, io_eof, eof_exception, java_io_eof, io_eof_exception, cause, cause_java, cause_java_io | Caused by java.io.EOFException | cause, java, io, eof, exception, cause_java, java_io, io_eof, eof_exception, cause_java_io, java_io_eof, io_eof_exception |
| 52 | io, cause, java, exception, connection, reset, peer, cause_java, java_io, io_exception, exception_connection, connection_reset, reset_peer, cause_java_io, io_exception_connection… | Caused by java.io.IOException: Connection reset by peer | cause, java, io, io, exception, connection, reset, peer, cause_java, java_io, io_exception, exception_connection, connection_reset, reset_peer, cause_java_io, io_exception_connection, exception_connection_reset, connection_reset_peer |
| 21 | java, io, eof, exception, java_io, io_eof, eof_exception, java_io_eof, io_eof_exception, cause, cause_java, cause_java_io | java.io.EOFException | java, io, eof, exception, java_io, io_eof, eof_exception, java_io_eof, io_eof_exception |
| 43 | fsdataset_impl, fsdataset, impl, datanode_fsdataset, datanode_fsdataset_impl, replica, add, map, pool, add_replica, replica_map, block_pool, add_replica_map, time, volume, map_block, pool_volume, replica_map_block… | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Total time to add all replicas to map: 0ms | fsdataset, impl, fsdataset, impl, total, time, add, replica, map, datanode_fsdataset, fsdataset_impl, fsdataset_impl, fsdataset_impl, impl_total, total_time, time_add, add_replica, replica_map, datanode_fsdataset_impl, fsdataset_impl_total, impl_total_time, total_time_add, time_add_replica, add_replica_map |
| 43 | fsdataset_impl, fsdataset, impl, datanode_fsdataset, datanode_fsdataset_impl, replica, add, map, pool, add_replica, replica_map, block_pool, add_replica_map, time, volume, map_block, pool_volume, replica_map_block… | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Total time to scan all replicas for block pool BP-108841162-10.10.34.11-1440074360971: 8ms | fsdataset, impl, fsdataset, impl, total, time, scan, replica, pool, datanode_fsdataset, fsdataset_impl, fsdataset_impl, fsdataset_impl, impl_total, total_time, time_scan, scan_replica, replica_block, block_pool, datanode_fsdataset_impl, fsdataset_impl_total, impl_total_time, total_time_scan, time_scan_replica, scan_replica_block, replica_block_pool |
| 14 | mortbay | org.mortbay.log: jetty-6.1.26 | mortbay |
| 32 | mortbay, log, mortbay_log | org.mortbay.log: Logging to org.slf4j.impl.Log4jLoggerAdapter (org.mortbay.log) via org.mortbay.log.Slf4jLog | mortbay, log, mortbay_log |
| 43 | fsdataset_impl, fsdataset, impl, datanode_fsdataset, datanode_fsdataset_impl, replica, add, map, pool, add_replica, replica_map, block_pool, add_replica_map, time, volume, map_block, pool_volume, replica_map_block… | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Time to add replicas to map for block pool BP-108841162-10.10.34.11-1440074360971 on volume /tmp/hadoop-hdfs/dfs/data/current: 0ms | fsdataset, impl, fsdataset, impl, time, add, replica, map, pool, volume, datanode_fsdataset, fsdataset_impl, fsdataset_impl, fsdataset_impl, impl_time, time_add, add_replica, replica_map, map_block, block_pool, pool_volume, datanode_fsdataset_impl, fsdataset_impl_time, impl_time_add, time_add_replica, add_replica_map, replica_map_block, map_block_pool, block_pool_volume |
| 3 | fsdataset_impl, fsdataset, datanode_fsdataset, datanode_fsdataset_impl, pool, block_pool, add, volume, impl_add, fsdataset_impl_add, scan, scan_block, scan_block_pool, new, add_new, new_volume, impl_add_new, add_new_volume… | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Time taken to scan block pool BP-108841162-10.10.34.11-1440074360971 on /tmp/hadoop-hdfs/dfs/data/current: 7ms | fsdataset, impl, fsdataset, impl, time, take, scan, pool, datanode_fsdataset, fsdataset_impl, fsdataset_impl, fsdataset_impl, impl_time, time_take, take_scan, scan_block, block_pool, datanode_fsdataset_impl, fsdataset_impl_time, impl_time_take, time_take_scan, take_scan_block, scan_block_pool |

**Figure 5.32:** Predictions GEAC on HDFS - ORIGIN+MESSAGE

Regarding SPLIT, the chosen model is composed as in the previous case by BERT, NMF and Word2Vec and it is made by 79 topics and it is built with an AutoEncoder with 128 dimension and K-Means clustering. On K-means, a peculiarity can be observed in the combinations of W2V with classical models. In fact, in both cases the performance is lower than with W2V alone but, with the introduction of BERT, the values increase again. The results provided by K-Medoids are, instead, all very similar between them, the only difference is represented by BERT with a lower value, associated with a consequent lower number of topics. As far as predictions are concerned, in general the performance are good, with specific and correctly assigned topics, even similar thoughts to ORIGIN+MESSAGE could be done.

| Results - HDFS - SPLIT | | | | |
|---|---|---|---|---|
| Model | Cluster | K | AE dim | Coherence |
| BERT+LDA+W2V | KMeans | 78 | 128 | 0.9155±0.0216 |
| **BERT+NMF+W2V** | **KMeans** | **79** | **128** | **0.9410±0.0184** |
| BERT+W2V | KMeans | 75 | 64 | 0.9276±0.0306 |
| BERT+LDA | KMeans | 73 | 128 | 0.8929±0.0296 |
| W2V+LDA | KMeans | 60 | 64 | 0.8638±0.0210 |
| BERT+NMF | KMeans | 62 | 64 | 0.8762±0.0253 |
| W2V+NMF | KMeans | 68 | 96 | 0.9019±0.0251 |
| BERT | KMeans | 72 | 128 | 0.9115±0.0207 |
| W2V | KMeans | 67 | 64 | 0.9087±0.0187 |
| BERT+LDA+W2V | KMedoids | 78 | 128 | 0.9304±0.0172 |
| BERT+NMF+W2V | KMedoids | 75 | 128 | 0.9292±0.0202 |
| BERT+W2V | KMedoids | 75 | 64 | 0.9288±0.0293 |
| BERT+LDA | KMedoids | 73 | 32 | 0.9143±0.0269 |
| W2V+LDA | KMedoids | 73 | 64 | 0.9173±0.0338 |
| BERT+NMF | KMedoids | 73 | 32 | 0.9059±0.0302 |
| W2V+NMF | KMedoids | 74 | 64 | 0.9089±0.0160 |
| BERT | KMedoids | 75 | 96 | 0.9133±0.0201 |
| W2V | KMedoids | 70 | 96 | 0.9172±0.0341 |

**Table 5.31:** GEAC Results on SPLIT HDFS dataset

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 8 | java, io, eof, exception, java_io, io_eof, eof_exception, java_io_eof, io_eof_exception, cause | Caused by java.io.EOFException | cause, java, io, eof, exception, java_io, io_eof, eof_exception, java_io_eof, io_eof_exception |
| 43 | io, exception, java, host, connection, reset, peer, java_io, io_exception, exception_connection, connection_reset, reset_peer, io_exception_connection, connection_reset_peer, exception_connection_reset, | Caused by java.io.IOException: Connection reset by peer | cause, java, io, io, exception, connection, reset, peer, java_io, io_exception, exception_connection, connection_reset, reset_peer, io_exception_connection, exception_connection_reset, connection_reset_peer |
| 8 | java, io, eof, exception, java_io, io_eof, eof_exception, java_io_eof, io_eof_exception, cause | java.io.EOFException | java, io, eof, exception, java_io, io_eof, eof_exception, java_io_eof, io_eof_exception |
| 49 | fsdataset, impl, pool, block_pool, add, add_block, add_block_pool, total, time, scan, replica, total_time, time_scan, scan_replica, replica_block, total_time_scan, time_scan_replica, scan_replica_block, replica_block_pool | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Total time to add all replicas to map: 0ms | fsdataset, impl, fsdataset, impl, total, time, add, replica, map, total_time, time_add, add_replica, replica_map, total_time_add, time_add_replica, add_replica_map |
| 12 | fsdataset, impl, add, replica, map, add_replica, replica_map, add_replica_map, pool, volume, map_block, block_pool, pool_volume, replica_map_block, map_block_pool, block_pool_volume, time, time_add... | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Total time to scan all replicas for block pool BP-108841162-10.10.34.11-1440074360971: 8ms | fsdataset, impl, fsdataset, impl, total, time, scan, replica, pool, total_time, time_scan, scan_replica, replica_block, block_pool, total_time_scan, time_scan_replica, scan_replica_block, replica_block_pool |
| 9 | mortbay | org.mortbay.log: jetty-6.1.26 | mortbay |
| 59 | mortbay, log, mortbay_log | org.mortbay.log: Logging to org.slf4j.impl.Log4jLoggerAdapter (org.mortbay.log) via org.mortbay.log.Slf4jLog | mortbay, log, mortbay_log |
| 12 | fsdataset, impl, add, replica, map, add_replica, replica_map, add_replica_map, pool, volume, map_block, block_pool, pool_volume, replica_map_block, map_block_pool, block_pool_volume, time, time_add... | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Time to add replicas to map for block pool BP-108841162-10.10.34.11-1440074360971 on volume /tmp/hadoop-hdfs/dfs/data/current: 0ms | fsdataset, impl, fsdataset, impl, time, add, replica, map, pool, volume, time_add, add_replica, replica_map, map_block, block_pool, pool_volume, time_add_replica, add_replica_map, replica_map_block, map_block_pool, block_pool_volume |
| 0 | fsdataset, impl, scan, pool, scan_block, block_pool, scan_block_pool, volume, pool_volume, block_pool_volume, time, take, time_take, take_scan, time_take_scan, take_scan_block | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Time taken to scan block pool BP-108841162-10.10.34.11-1440074360971 on /tmp/hadoop-hdfs/dfs/data/current: 7ms | fsdataset, impl, fsdataset, impl, time, take, scan, pool, time_take, take_scan, scan_block, block_pool, time_take_scan, take_scan_block, scan_block_pool |

**Figure 5.33:** Predictions GEAC on HDFS - SPLIT

In last table 5.32 the results are still obtained with the same combination of the two cases above, but this time with 80 different topics. Some trends can be observed in this case. On K-Means, the best models are skewed towards NMF, indeed, on LDA the performance are overall worse. On K-Medoids the results are similar but with better performance on the models based on LDA and, in general, the addition of other techniques to BERT/Word2Vec increases always the score. Regarding the predictions, the associations are good, with clusters able to group together logs that are really similar.

| Results - HDFS - MESSAGE | | | | |
|---|---|---|---|---|
| Model | Cluster | K | AE dim | Coherence |
| BERT+LDA+W2V | KMeans | 77 | 64 | 0.9176±0.0160 |
| **BERT+NMF+W2V** | **KMeans** | **80** | **64** | **0.9581±0.0136** |
| BERT+W2V | KMeans | 75 | 128 | 0.9336±0.0282 |
| BERT+LDA | KMeans | 73 | 128 | 0.9121±0.0259 |
| W2V+LDA | KMeans | 70 | 96 | 0.8301±0.0186 |
| BERT+NMF | KMeans | 75 | 128 | 0.9355±0.0258 |
| W2V+NMF | KMeans | 70 | 128 | 0.9161±0.0259 |
| BERT | KMeans | 78 | 128 | 0.9321±0.0201 |
| W2V | KMeans | 81 | 64 | 0.9569±0.0126 |
| BERT+LDA+W2V | KMedoids | 77 | 96 | 0.9288±0.0212 |
| BERT+NMF+W2V | KMedoids | 82 | 96 | 0.9528±0.0212 |
| BERT+W2V | KMedoids | 81 | 32 | 0.9439±0.0290 |
| BERT+LDA | KMedoids | 77 | 128 | 0.9218±0.0172 |
| W2V+LDA | KMedoids | 69 | 128 | 0.8703±0.0311 |
| BERT+NMF | KMedoids | 77 | 64 | 0.9428±0.0308 |
| W2V+NMF | KMedoids | 78 | 64 | 0.9427±0.0223 |
| BERT | KMedoids | 75 | 32 | 0.9085±0.0270 |
| W2V | KMedoids | 75 | 96 | 0.9189±0.0113 |

**Table 5.32:** GEAC Results on MESSAGE HDFS dataset

| T | Topic Words | Raw Log | Feature extracted |
|---|---|---|---|
| 48 | io, exception, host, local, java, connection, reset, peer, java_io, io_exception, exception_connection, connection_reset, reset_peer, host_destination, io_exception_connection, exception_connection_reset... | Caused by java.io.IOException: Connection reset by peer | java, io, eof, exception, java_io, io_eof, eof_exception, java_io_eof, io_eof_exception |
| 79 | java, io, eof, exception, java_io, io_eof, eof_exception, java_io_eof, io_eof_exception | java.io.EOFException | java, io, io, exception, connection, reset, peer, java_io, io_exception, exception_connection, connection_reset, reset_peer, io_exception_connection, exception_connection_reset, connection_reset_peer |
| 9 | total, time, scan, replica, pool, total_time, time_scan, scan_replica, replica_block, block_pool, total_time_scan, time_scan_replica, scan_replica_block, replica_block_pool | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Total time to add all replicas to map: 0ms | total, time, add, replica, map, total_time, time_add, add_replica, replica_map, total_time_add, time_add_replica, add_replica_map |
| 38 | add, replica, map, add_replica, replica_map, add_replica_map, pool, volume, map_block, block_pool, pool_volume, replica_map_block, map_block_pool, block_pool_volume, time, time_add, time_add_replica, total, total_time... | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Total time to scan all replicas for block pool BP-108841162-10.10.34.11-1440074360971: 8ms | total, time, scan, replica, pool, total_time, time_scan, scan_replica, replica_block, block_pool, total_time_scan, time_scan_replica, scan_replica_block, replica_block_pool |
| 8 | jetty | org.mortbay.log: jetty-6.1.26 | jetty |
| 22 | log | org.mortbay.log: Logging to org.slf4j.impl.Log4jLoggerAdapter(org.mortbay.log) via org.mortbay.log.Slf4jLog | log |
| 9 | total, time, scan, replica, pool, total_time, time_scan, scan_replica, replica_block, block_pool, total_time_scan, time_scan_replica, scan_replica_block, replica_block_pool | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Time to add replicas to map for block pool BP-108841162-10.10.34.11-1440074360971 on volume /tmp/hadoop-hdfs/dfs/data/current: 0ms | time, add, replica, map, pool, volume, time_add, add_replica, replica_map, map_block, block_pool, pool_volume, time_add_replica, add_replica_map, replica_map_block, map_block_pool, block_pool_volume |
| 38 | add, replica, map, add_replica, replica_map, add_replica_map, pool, volume, map_block, block_pool, pool_volume, replica_map_block, map_block_pool, block_pool_volume, time, time_add, time_add_replica, total, total_time... | org.apache.hadoop.hdfs.server.datanode.fsdataset.impl.FsDatasetImpl: Time taken to scan block pool BP-108841162-10.10.34.11-1440074360971 on /tmp/hadoop-hdfs/dfs/data/current: 7ms | time, take, scan, pool, time_take, take_scan, scan_block, block_pool, time_take_scan, take_scan_block, scan_block_pool |

**Figure 5.34:** Predictions GEAC on HDFS - MESSAGE

Overall, on HDFS the best models are obtained all with the same combination and with K-Means. By comparing with Spark, K-Means seems to perform better or similar to K-Medoids in situations where the datasets is quite big. Conversely, K-Medoids seems to produce better topics on small datasets, due to the better exploration of the space with the cosine similarity.

The final conclusion on this custom model is that in all cases, for both Spark and HDFS, the best results are obtained with the combination of the 3 elements, BERT, Word2Vec and a classical algorithm. The other results do not follow a precise trend, since, in many cases, a combination with less elements provides better results than more complex ones. Finally, the scores are all improved with respect to all the other models, making this proposal as the best suited for this case study.

# Chapter 6

# Conclusion

This thesis represents a first step to tackle the complex tasks of Topic Modeling and data processing. The nature of the data makes the analysis challenging. It is important to understand how to manage an uncommon textual structure as log and how to implement topic models in uncomfortable context, where there are few words and sentences.

The proposed data processing framework is able to clean logs with a satisfying level. Indeed, extracting useful words from those data requires many efforts to understand which elements have to be considered as noise and which are instead the crucial parts to figure out the theme expressed in that log.

Then, the deep analysis on the different algorithms depicts how, over the year, there have been several improvements in this field. The performance are quite acceptable in classical proposals, even if their drawbacks make them not recommended for a real case scenario. Instead, the newer options available in the latest years highlight the possibility to use Topic Modeling in situation unfeasible before. The study on the different versions of the datasets underlines this aspect. Indeed, the MESSAGE ones are really hard to process correctly and we have seen how with BERTopic and GEAC are possible to manage.

About these last two models, the considerations on BERTopic extend its usage in situations where the input data are too low and all must be kept. Rather than discarding such an interesting model due to its formulation, I can use its shortcomings as an advantage with similarity analysis to produce really good level of quality for predictions. Instead, my custom model GEAC, coming directly from the other models techniques, shows how the combination of classical and modern approaches can provide high results, whit additional support for this task.

Nevertheless, today many issues are still present in Topic Modeling. In this context, the analysis of the topics depending on the similarity score represents an important consideration: it helps to focus on the real meaningful models only. Overall, the analysis about similarity is incredibly useful in all the models. It makes

easy the choice of the best model, avoiding situations that could compromise the future step of anomaly detection and letting the comprehension of the outputs more feasible.

## 6.1 Future Implementations

Although the analysis and the study have achieved top-level results, many improvements are still possible.

About the Data processing phase, today other solutions exist to manage the steps implemented in better ways. As discussed in Chapter 3, in this project it is essential to use algorithms human readable, due to the novelty of data used. However, techniques based on neural networks can be useful and provide additional help for the steps of POS tagging and Lemmization, making them less dependant on the supervisor.

For topic models, the focus has been placed on Neural networks, in particular on AutoEncoders and their variations. In the latest years, however, the research begins to propose other networks like GAN, as shown in the detailed analysis in [69]. This network can be helpful for many other situations of Topic Modeling, also outside the context of log analysis.

Regarding the score used, the $CV$ coherence is suitable for this task but, from the experience acquired, it seems clear the need of improvements. Indeed, the perfect matching or the presence of mixtures of words make this metric not reliable at a first glance. For this reason, different studies and proposals are possible to achieve better results, when those situations occur.

# Bibliography


[1] SumoLogic. *Log Analysis*. URL: https://www.sumologic.com/glossary/log-analysis/ (cit. on p. 4).

[2] Wikipedia. *Natural language processing*. URL: https://en.wikipedia.org/wiki/Natural_language_processing (cit. on p. 4).

[3] Wikipedia. *Topic model*. URL: https://en.wikipedia.org/wiki/Topic_model (cit. on p. 6).

[4] Shilin He, Jieming Zhu, Pinjia He, and Michael R. Lyu. «Loghub: A Large Collection of System Log Datasets towards Automated Log Analytics». In: *CoRR* abs/2008.06448 (2020). arXiv: 2008.06448. URL: https://arxiv.org/abs/2008.06448 (cit. on p. 6).

[5] K. Shvachko, Hairong Kuang, S. Radia, and R. Chansler. «The Hadoop Distributed File System». In: *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. May 2010, pp. 1–10. DOI: 10.1109/MSST.2010.5496972. URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5496972&contentType=Conference+Publications&queryText%3Dthe+hadoop+distributed+file+system (cit. on p. 7).

[6] Matei Zaharia et al. «Apache Spark: A Unified Engine for Big Data Processing». In: *Commun. ACM* 59.11 (Oct. 2016), pp. 56–65. ISSN: 0001-0782. DOI: 10.1145/2934664. URL: https://doi.org/10.1145/2934664 (cit. on p. 7).

[7] Wikipedia. *Apache Spark*. URL: https://en.wikipedia.org/wiki/Apache_Spark (cit. on p. 7).

[8] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. «Deeplog: Anomaly detection and diagnosis from system logs through deep learning». In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017, pp. 1285–1298 (cit. on p. 10).

[9] Weibin Meng et al. «LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs». In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 4739–4745. DOI: 10.24963/ijcai.2019/658. URL: https://doi.org/10.24963/ijcai.2019/658 (cit. on p. 10).

[10] Towards Data Science. *NLP: Building Text Cleanup and PreProcessing Pipeline*. URL: https://towardsdatascience.com/nlp-building-text-cleanup-and-preprocessing-pipeline-eba4095245a0 (cit. on p. 16).

[11] P.V. Kooten. *Contractions*. 2017. URL: https://github.com/kootenpv/contractions (cit. on p. 16).

[12] Neptune.AI. *Tokenization in NLP*. URL: https://neptune.ai/blog/tokenization-in-nlp (cit. on p. 17).

[13] Edward Loper and Steven Bird. «NLTK: The Natural Language Toolkit». In: *CoRR* cs.CL/0205028 (2002). URL: http://dblp.uni-trier.de/db/journals/corr/corr0205.html#cs-CL-0205028 (cit. on pp. 17, 20).

[14] Towards Data Science. *Part Of Speech Tagging for Beginners*. URL: https://towardsdatascience.com/part-of-speech-tagging-for-beginners-3a0754b2ebba (cit. on p. 17).

[15] DataCamp. *Stemming and Lemmatization*. URL: https://www.datacamp.com/community/tutorials/stemming-lemmatization-python (cit. on p. 19).

[16] Medium. *NLP- Text Preprocessing Techniques*. URL: https://medium.com/swlh/nlp-text-preprocessing-techniques-ea34d3f84de4 (cit. on p. 20).

[17] Christiane Fellbaum, ed. *WordNet: An Electronic Lexical Database*. Language, Speech, and Communication. Cambridge, MA: MIT Press, 1998. ISBN: 978-0-262-06197-1 (cit. on pp. 20, 21).

[18] Antoine Blanchard. «Understanding and customizing stopword lists for enhanced patent mapping». In: *World Patent Information* 29.4 (Dec. 2007), p. 308. DOI: 10.1016/j.wpi.2007.02.002. URL: https://hal.archives-ouvertes.fr/hal-01247971 (cit. on p. 22).

[19] W. John Wilbur and Karl Sirotkin. «The automatic identification of stop words». In: *Journal of Information Science* (1992), pp. 45–55 (cit. on p. 22).

[20] Serhad Sarica and Jianxi Luo. «Stopwords in Technical Language Processing». In: *CoRR* abs/2006.02633 (2020). arXiv: 2006.02633. URL: https://arxiv.org/abs/2006.02633 (cit. on p. 22).

[21] Wikipedia. *Latent Dirichlet Allocation*. URL: https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation (cit. on p. 28).

[22] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. «Latent Dirichlet Allocation». In: *J. Mach. Learn. Res.* 3.null (Mar. 2003), pp. 993–1022. ISSN: 1532-4435 (cit. on pp. 28, 49).

[23] Wikipedia. *Non-negative matrix factorization*. URL: https://en.wikipedia.org/wiki/Non-negative_matrix_factorization (cit. on p. 31).

[24] Thomas Hofmann. «Probabilistic Latent Semantic Analysis». In: *Proc. of Uncertainty in Artificial Intelligence, UAI'99*. Stockholm, 1999. URL: http://citeseer.ist.psu.edu/hofmann99probabilistic.html (cit. on p. 32).

[25] Renbo Zhao and Vincent Y. F. Tan. «Online Nonnegative Matrix Factorization With Outliers». In: *IEEE Transactions on Signal Processing* 65.3 (Feb. 2017), pp. 555–570. ISSN: 1941-0476. DOI: 10.1109/tsp.2016.2620967. URL: http://dx.doi.org/10.1109/TSP.2016.2620967 (cit. on p. 33).

[26] Julien Mairal. «Stochastic Majorization-Minimization Algorithms for Large-Scale Optimization». In: *Advances in Neural Information Processing Systems*. Ed. by C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger. Vol. 26. Curran Associates, Inc., 2013. URL: https://proceedings.neurips.cc/paper/2013/file/4da04049a062f5adfe81b67dd755cecc-Paper.pdf (cit. on p. 33).

[27] Wikipedia. *Latent Semantic Analysis*. URL: https://en.wikipedia.org/wiki/Latent_semantic_analysis (cit. on p. 35).

[28] Wikipedia. *Artificial Neural Network*. URL: https://en.wikipedia.org/wiki/Artificial_neural_network (cit. on p. 39).

[29] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. «Learning internal representations by error propagation». In: 1986 (cit. on p. 40).

[30] Wikipedia. *AutoEncoder*. URL: https://en.wikipedia.org/wiki/Autoencoder (cit. on p. 40).

[31] Diederik P. Kingma and Max Welling. «Auto-Encoding Variational Bayes». In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. 2014. arXiv: http://arxiv.org/abs/1312.6114v10 [stat.ML] (cit. on pp. 41, 50).

[32] Wikipedia. *Variational AutoEncoder*. URL: https://en.wikipedia.org/wiki/Variational_autoencoder (cit. on p. 41).

[33] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL]. URL: https://code.google.com/archive/p/word2vec/ (cit. on pp. 44, 64, 67).

[34] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding». In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805 (cit. on pp. 44, 45).

[35] Nils Reimers and Iryna Gurevych. «Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks». In: *CoRR* abs/1908.10084 (2019). arXiv: 1908.10084. URL: http://arxiv.org/abs/1908.10084 (cit. on pp. 44, 48, 67).

[36] Wikipedia. *Word2Vec*. URL: https://en.wikipedia.org/wiki/Word2vec (cit. on p. 44).

[37] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. «Deep contextualized word representations». In: *CoRR* abs/1802.05365 (2018). arXiv: 1802.05365. URL: http://arxiv.org/abs/1802.05365 (cit. on p. 44).

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. «Attention Is All You Need». In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762 (cit. on p. 45).

[39] Peter Bloem. *Transformers from scratch*. URL: http://peterbloem.nl/blog/transformers (cit. on p. 45).

[40] Jay Alammar. *The Illustrated Transformer*. URL: http://jalammar.github.io/illustrated-transformer/ (cit. on p. 45).

[41] Akash Srivastava and Charles Sutton. «Autoencoding variational inference for topic models». In: *arXiv preprint arXiv:1703.01488* (2017) (cit. on p. 49).

[42] Diederik P. Kingma and Jimmy Ba. «Adam: A Method for Stochastic Optimization». In: *CoRR* abs/1412.6980 (2015) (cit. on p. 51).

[43] Dimitar Angelov. «Top2Vec: Distributed Representations of Topics». In: *ArXiv* abs/2008.09470 (2020) (cit. on p. 52).

[44] Quoc V. Le and Tomas Mikolov. «Distributed Representations of Sentences and Documents». In: *ArXiv* abs/1405.4053 (2014) (cit. on p. 53).

[45] Ricardo J. G. B. Campello, Davoud Moulavi, and Joerg Sander. «Density-Based Clustering Based on Hierarchical Density Estimates». In: *Advances in Knowledge Discovery and Data Mining*. Ed. by Jian Pei, Vincent S. Tseng, Longbing Cao, Hiroshi Motoda, and Guandong Xu. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 160–172. ISBN: 978-3-642-37456-2 (cit. on p. 54).

[46] Leland McInnes and John Healy. «UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction». In: *ArXiv* abs/1802.03426 (2018) (cit. on p. 54).

[47] Maarten Grootendorst. *BERTopic: Leveraging BERT and c-TF-IDF to create easily interpretable topics.* Version v0.9.4. 2020. DOI: 10.5281/zenodo.4381785. URL: https://doi.org/10.5281/zenodo.4381785 (cit. on pp. 57, 67).

[48] Radim Rehurek and Petr Sojka. «Gensim–python framework for vector space modelling». In: *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic* 3.2 (2011) (cit. on pp. 58, 66, 67).

[49] James MacQueen et al. «Some methods for classification and analysis of multivariate observations». In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. 14. Oakland, CA, USA. 1967, pp. 281–297 (cit. on p. 59).

[50] «Partitioning Around Medoids (Program PAM)». In: *Finding Groups in Data*. John Wiley  Sons, Ltd, 1990. Chap. 2, pp. 68–125. ISBN: 9780470316801. DOI: https://doi.org/10.1002/9780470316801.ch2. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470316801.ch2. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470316801.ch2 (cit. on p. 59).

[51] Jonathan Chang, Sean Gerrish, Chong Wang, Jordan Boyd-graber, and David Blei. «Reading Tea Leaves: How Humans Interpret Topic Models». In: *Advances in Neural Information Processing Systems*. Ed. by Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta. Vol. 22. Curran Associates, Inc., 2009. URL: https://proceedings.neurips.cc/paper/2009/file/f92586a25bb3145facd64ab20fd554ff-Paper.pdf (cit. on p. 61).

[52] David Newman, Jey Han Lau, Karl Grieser, and Timothy Baldwin. «Automatic Evaluation of Topic Coherence». In: *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. HLT '10. Los Angeles, California: Association for Computational Linguistics, 2010, pp. 100–108. ISBN: 1932432655 (cit. on pp. 61, 62).

[53] Towards Data Science. *Evaluate Topic Models*. URL: https://towardsdatascience.com/evaluate-topic-model-in-python-latent-dirichlet-allocation-lda-7d57484bb5d0 (cit. on p. 61).

[54] Michael Röder, Andreas Both, and Alexander Hinneburg. «Exploring the Space of Topic Coherence Measures». In: *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*. WSDM '15. Shanghai, China: Association for Computing Machinery, 2015, pp. 399–408. ISBN:

9781450333177. DOI: 10.1145/2684822.2685324. URL: https://doi.org/10.1145/2684822.2685324 (cit. on p. 62).

[55] Branden Fitelson. «A Probabilistic Theory of Coherence». In: *Analysis* 63.3 (2003), pp. 194–199. DOI: 10.1111/1467-8284.00420 (cit. on p. 62).

[56] David Mimno, Hanna M. Wallach, Edmund Talley, Miriam Leenders, and Andrew McCallum. «Optimizing Semantic Coherence in Topic Models». In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. EMNLP '11. Edinburgh, United Kingdom: Association for Computational Linguistics, 2011, pp. 262–272. ISBN: 9781937284114 (cit. on p. 62).

[57] Gerlof Bouma. «Normalized (pointwise) mutual information in collocation extraction». In: 2009 (cit. on pp. 62, 63).

[58] Igor Douven and Wouter Meijs. «Measuring coherence». In: *Synthese* 156 (May 2007), pp. 405–425. DOI: 10.1007/s11229-006-9131-z (cit. on p. 62).

[59] Silvia Terragni, Elisabetta Fersini, Bruno Giovanni Galuzzi, Pietro Tropeano, and Antonio Candelieri. «OCTIS: Comparing and Optimizing Topic models is Simple!» In: *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*. Online: Association for Computational Linguistics, Apr. 2021, pp. 263–270. DOI: 10.18653/v1/2021.eacl-demos.31. URL: https://aclanthology.org/2021.eacl-demos.31 (cit. on p. 67).

[60] Francois Chollet et al. *Keras*. 2015. URL: https://github.com/fchollet/keras (cit. on p. 67).

[61] Lars Buitinck et al. «API design for machine learning software: experiences from the scikit-learn project». In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122 (cit. on pp. 67, 102).

[62] R. Yurchak. *Scikit-Learn-Extra*. https://github.com/scikit-learn-contrib/scikit-learn-extra/. 2020 (cit. on pp. 67, 102).

[63] Politecnico di Torino. *Smart Data PoliTO*. https://smartdata.polito.it/ (cit. on p. 67).

[64] Ekaba Bisong. «Google Colaboratory». In: *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*. Berkeley, CA: Apress, 2019, pp. 59–64. ISBN: 978-1-4842-4470-8. DOI: 10.1007/978-1-4842-4470-8_7. URL: https://doi.org/10.1007/978-1-4842-4470-8_7 (cit. on p. 67).

[65] Google. *Google Colaboratory*. URL: https://colab.research.google.com/ (cit. on p. 67).

[66] Liangjie Hong and Brian D. Davison. «Empirical Study of Topic Modeling in Twitter». In: *Proceedings of the First Workshop on Social Media Analytics*. SOMA '10. Washington D.C., District of Columbia: Association for Computing Machinery, 2010, pp. 80–88. ISBN: 9781450302173. DOI: 10.1145/1964858.1964870. URL: https://doi.org/10.1145/1964858.1964870 (cit. on p. 69).

[67] Rania Albalawi, Tet Hin Yeap, and Morad Benyoucef. «Using Topic Modeling Methods for Short-Text Data: A Comparative Analysis». In: *Frontiers in Artificial Intelligence* 3 (2020). ISSN: 2624-8212. DOI: 10.3389/frai.2020.00042. URL: https://www.frontiersin.org/article/10.3389/frai.2020.00042 (cit. on p. 69).

[68] Zhikui Chen, Shan Jin, Runze Liu, and Jianing Zhang. «A Deep Non-negative Matrix Factorization Model for Big Data Representation Learning». In: *Frontiers in Neurorobotics* 15 (2021). ISSN: 1662-5218. DOI: 10.3389/fnbot.2021.701194. URL: https://www.frontiersin.org/article/10.3389/fnbot.2021.701194 (cit. on p. 83).

[69] He Zhao, Dinh Phung, Viet Huynh, Yuan Jin, Lan Du, and Wray L. Buntine. «Topic Modelling Meets Deep Neural Networks: A Survey». In: *CoRR* abs/2103.00498 (2021). arXiv: 2103.00498. URL: https://arxiv.org/abs/2103.00498 (cit. on p. 116).