UNIVERSITÀ DI PISA

Master Degree in Artificial Intelligence and Data Engineering

Business and Project Management

Operations/Image processing: Automatic defect detection

Carmine Tranfa

Contents

1. Introduction

The water utility companies in Italy are the less technological ones if compared to the gas or electrical ones. Due to this gap, in the last decades Italian Governments tryed to promote the technical innovation grow in the sector through subsidized loans and law decrees.

These companies have different challenges waiting for them in the next future:

1. the climate changes;
2. the environement pubblic attenction;
3. the importance and the fragility of the drinking water like limited resource.

In this context this kind of companies are pushed to grow as much as possibile to close the gap. One of the opportunity they have is the boost that AI could bring to the companies in a lot of ways.

1.1 Reasearch question

We assumed the point of view of a water utility company that would like to understand if they could reduce the operation costs on the assets and infrastructures using the automatic defects detection. They are interseted to understand if this tool has been used in water company industry.

1.2 Goal of the project

The aim of our project is to analyze the reseach papers about the “image defect detection” in order to understand in which industries this tool is used most and if there is some experiences about their use in the water companies.

2. Process

2.1 Data scraping

First of all it was created the dateaset of abstract's papers related to a search on Google Scholar of these keywords:

```
search_keyword = ['image','defect','detection']
```

To create the complete dataset, the scraping work was divided in two times:

1. Scraping of titles and links of the papers related to the search from Google Scholar;
2. Scraping of the abstracts related to every single papers from the related websites.

To avoid the google blocks, it was preferred to scrape the data from the Google pages instead of using the Google API via requests library. To performe the scraping of the pages it was used the Selenium and BeautifulSoup libraries with the chrome driver. Trying to skip al blocks as possibile, we used these options related to the chrome driver agent:

```
### Chrome driver options
filename = 'chromedriver.exe'
chromedriver_path = os.path.join(drivepath, filename)

options = webdriver.ChromeOptions()
options.add_argument("start-maximized")
options.add_experimental_option("excludeSwitches", ["enable-automation"])
options.add_experimental_option('useAutomationExtension', False)

user_data_dir = "C:/Users/carmi/AppData/Local/Google/Chrome/User Data"
options.add_argument('--user-data-dir=' + user_data_dir)
options.add_argument('--profile-directory=profile_selenium')
```

The most important option used is the one related to the Chrome's user data. With this option it was possible to skip very well quite all the Google blocks.

```python
# this function for the getting inforamtion of the web page
def get_paperinfo_selenium(page_url, sleep_time= 15):
    try:
        driver = webdriver.Chrome(executable_path=chromedriver_path, options=options)
        driver.get(page_url)
        sleep(sleep_time)
        html = driver.page_source

        #parse using beautiful soup
        paper_doc = BeautifulSoup(html,'lxml')
    except Exception as e:
        print(e)
    finally:
        driver.close()
        driver.quit()

    return paper_doc
```

```python
### Link to the GS search
# get url for the each page
url_start = "https://scholar.google.com/scholar?start={}&q="
query_url = '+'.join(search_keyword)
url_end = "&hl=en&as_sdt=0,5"

url0 = url_start + query_url + url_end

for i in tqdm(range(id_paper,990,10)):

    id_paper = i
    url = url0.format(i)

    doc = get_paperinfo_selenium(url, sleep_time=15)
    if doc == None:
        break

    # function for the collecting tags
    link_tag, author_tag = get_tags(doc)

    # paper title and link from each page
    papername, link = get_papername_link_info(link_tag)

    # year , author , publication of the paper
    year, publication, author = get_author_year_publi_info(author_tag)

    print("ID:", i, "-", papername[0])

    # add in paper repo dict
    df_papers = add_in_paper_repo(papername,
                                  year,
                                  author,
                                  publication,
                                  link)
```

The others functions used in the script are utility functions created to take the data that there would like to scrape:

	Paper Title	Year	Author	Publication	Url of paper
0	Fabric defect detection and classification usi...	1995.0	YF Zhang, RR Bresee	journals.sagepub.com	https://journals.sagepub.com/doi/abs/10.1177/0...
1	Approaches for improvement of the X-ray image ...	2019.0	W Du, H Shen, J Fu, G Zhang, Q He	Elsevier	https://www.sciencedirect.com/science/article/...
2	The application of one-class classifier based ...	2017.0	M Zhang, J Wu, H Lin, P Yuan, Y Song	Elsevier	https://www.sciencedirect.com/science/article/...

After we compled the download of the 100 pages Google allow to navigate, it was possible to collect a 1'000 rows dataset of papers related to the keywords searched.

After the first step, it was possibile to try to download the abstract of every articles in the dataframe. Like before, it was preferred to download the data scraping the information using Selenium, BeautifulSoup and chrome driver:

```
def get_abstract_selenium(paper_url, sleep_time=15):

    driver = webdriver.Chrome(executable_path=chromedriver_path, options=options)
    try:
        driver.get(paper_url)
        sleep(10)
        html = driver.page_source
        #parse using beautiful soup
        doc = BeautifulSoup(html,'lxml')
        abstract = extract_abstract_from_source(doc)

    except:
        abstract = None

    finally:
    ### close chrome driver
        driver.close()
        driver.quit()

    return abstract
```

Not all the links getted from Google were still working and for some relative small scientific companies it was not possibile to scrape the abstract.

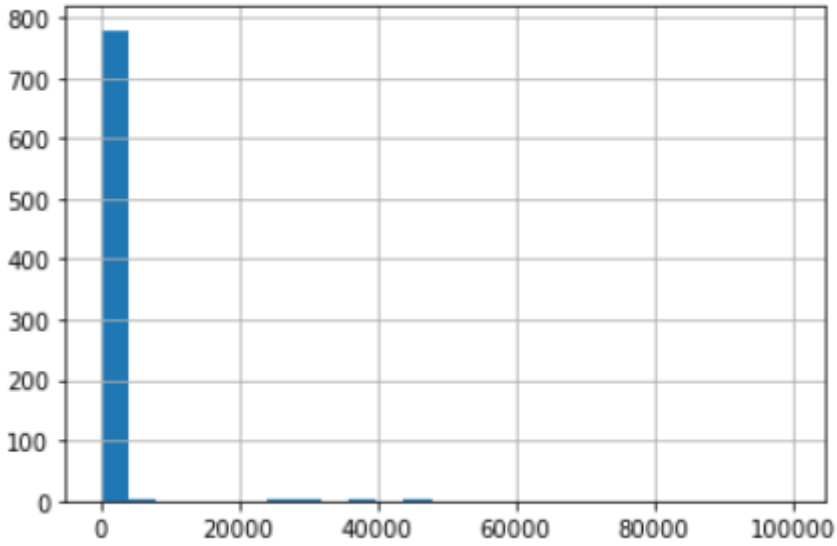
At this point it was possibile to start to work on the raw data downloaded to perform the desired analysis.

2.2 Data check

Using custom functions to scrape the data and due to the difference between the scientic websites, it was neccessary to quickly pre-analyse the data before starting the NLP's part.

```
ID: 469 - https://www.scientific.net/AMR.734-737.2898
ID: 489 - https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.638
ID: 497 - AbstractDetection of substation equipment can prom ...
ID: 498 -



Click on "Download PDF" for the PDF version o ...
ID: 500 - https://search.ieice.org/bin/summary.php?id=e90-c_11_21
ID: 522 - https://ascelibrary.org/doi/abs/10.1061/(ASCE)CP.1943-5
ID: 524 - https://www.matec-conferences.org/articles/matecconf/ab
0.html
ID: 526 - https://search.proquest.com/openview/0f9b53f5a0712b6f4c
ID: 527 - https://www.scientific.net/AMR.295-297.1274
ID: 534 - AbstractAutomatic defect detection is an important ...
ID: 543 - AbstractIn recent years, more and more scholars de ...
ID: 552 - https://search.proquest.com/openview/446c4242a1d7c4b471
```

It was made a quality check of the data analyzing the length of the abstracts and sampling the dataframe.

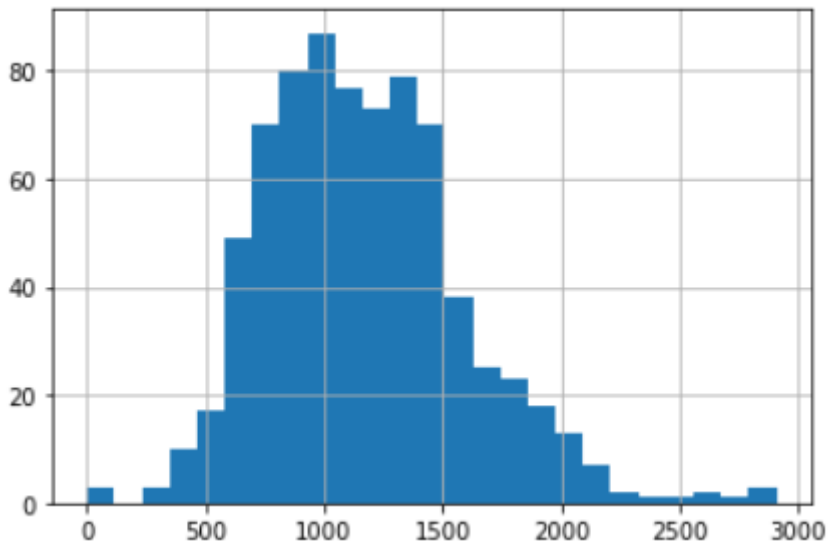


After these checks it was possibile to clean the data extracting from the texts only the abstract's part.

```python
def skip_text(text, text_to_find, pos = 'start'):
    if text==None:
        return None
    elif text.find(text_to_find) == -1:
        return text
    elif pos == 'start':
        return text[text.find(text_to_find)+len(text_to_find):]
    elif pos == 'end':
        return text[:text.find(text_to_find)]
```

```python
def remove_intial_char(text, char=' '):
    if text==None:
        return None
    char_len = len(char)
    if text[:char_len].lower()==char.lower():
        text = text[char_len:]
        text = remove_intial_char(text, char)

    return text

def remove_final_char(text, char=' '):
    if text==None:
        return None
    char_len = len(char)
    if text[-char_len:].lower()==char.lower():
        text = text[:-char_len]
        text = remove_final_char(text, char)

    return text
```

After this part, it was possible to obtain a Gaussian distribution of the abstracts length.



2.3 Topic modeling

Topic modeling is a powerful technique for unsupervised analysis of large document collections. Topic models conceive latent topics in text using hidden random variables, and discover that structure with posterior inference. Topic models have a wide range of applications like tag recommendation, text categorization, keyword extraction and similarity search in the broad fields of text mining, information retrieval, statistical language modeling. […]

(Paper Title: LDA based topic modeling of journal abstracts – Authors: P. Anupriya, S. Karpagavalli – Link: https://ieeexplore.ieee.org/document/7324058)

After a little search on the best method suggested to use on abstracts data, it was decided to use topic modeling technique to analyze abstracts data.

It was used the LDA (Latent Dirichlet Allocation) model cause it's one of the famousest and used model for the topic modeling problems:

LDA is a generative probabilistic model that assumes each topic is a mixture over an underlying set of words, and each document is a mixture of over a set of topic probabilities.

The main steps performed are:

1. Data pre-processing
2. Exploratory analysis
3. Preparing data for LDA analysis
4. Building LDA model
5. Analyzing LDA model results

2.3.1 Data pre-processing

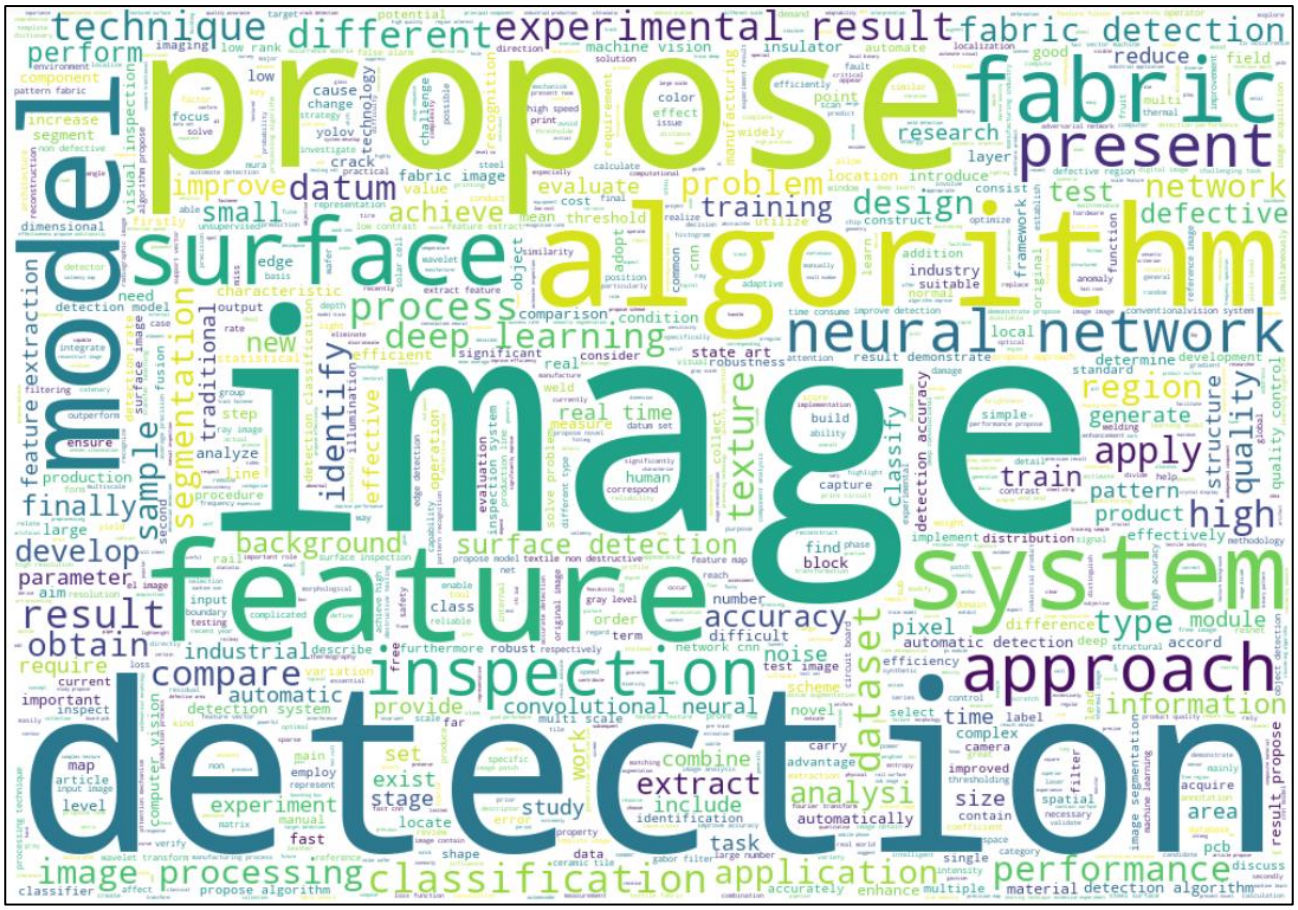
Since the goal of this analysis is to perform topic modeling on the abstracts, it was created a list conteining all the abstracts downloaded.

All the elements of the list are:

1. Transformed in lower case;
2. Tokenized and cleaned from punctuations (Gensim - simple_preprocess with deacc=True);
3. Cleaned from all the “StopWords” (SpaCy);
4. Lemmatized (SpaCy “en_core_web_lg” model).

2.3.2 Exploratory Analysis

To verify whether the preprocessing, it was made a word cloud using the “WordCloud” package to get a visual representation of most common words. It is important to understanding the data and if any more preprocessing is necessary before training the model.



[…]

There are increasing applications of natural language processing techniques for information retrieval, indexing, topic modelling and text classification in engineering contexts. A standard component of such tasks is the removal of stopwords, which are uninformative components of the data. While researchers use readily available stopwords lists that are derived from non-technical resources, the technical jargon of engineering fields contains their own highly frequent and uninformative words and there exists no standard stopwords list for technical language processing applications. Here we address this gap by rigorously identifying generic, insignificant, uninformative stopwords in engineering texts beyond the stopwords in general texts, based on the synthesis of alternative statistical measures such as term frequency, inverse document frequency, and entropy, and curating a stopwords dataset ready for technical language processing applications.

[...]

(Paper: Stopwords in technical language processing, link: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0254937)

For the reason above, it was created a list of “dummy words” frequently repeated on al the documents and that don’t give any addictional information to the analysis. After that it was applied a function to remove all this words.

2.3.3 Prepare data for LDA Analysis

To perform the Topic Model analysis it was used the “Gensim” python’s package:

Gensim is a Python library for topic modelling, document indexing *and* similarity retrieval *with large corpora. Target audience is the* natural language processing *(NLP) and* information retrieval *(IR) community.* (webpage: https://pypi.org/project/gensim/)

To perform the model it is necessary to:

1. Create a “Dictionary” of the words in the abstracts (gensim.corpora.Dictonary)
2. Count how many times each words occurs in the abstracts (doc2bow() function).

The last point it allow to define a “Corpus”.

Gensim creates a unique id for each word in the document. The produced corpus is a mapping of (word_id, word_frequency).

For example, (0, 1) means that word_id 0 occurs once.

Dictionary and corpus is used as the input by the LDA model.

2.3.4 Building LDA model

It was created the LDA model using the Corpus ("corpus") and the Dictionary ("id2word") created before. The other parameters used (update_every, chunksize, passes, alpha) were chosen different from the default value cause the number of abstracts used in this project is not so big (less than 1'000 abstracts vs a default chucksize of 2'000).

```
class gensim.models.ldamodel.LdaModel(corpus=None, num_topics=100, id2word=None,
distributed=False, chunksize=2000, passes=1, update_every=1, alpha='symmetric', eta=None,
decay=0.5, offset=1.0, eval_every=10, iterations=50, gamma_threshold=0.001,
minimum_probability=0.01, random_state=None, ns_conf=None, minimum_phi_value=0.01,
per_word_topics=False, callbacks=None, dtype=<class 'numpy.float32'>)
```

```
# number of topics
num_topics = 10

chunksize = 100
passes = 10
update_every = 1
alpha = 'auto'

# Build LDA model
lda_model = gensim.models.LdaModel(corpus=corpus,
                                   id2word=id2word,
                                   update_every=update_every,
                                   num_topics=num_topics,
                                   chunksize=chunksize,
                                   passes=passes,
                                   alpha=alpha,
                                   per_word_topics=True)

# Print the Keyword in the 10 topics
[ print(t) for t in lda_model.print_topics()]
```

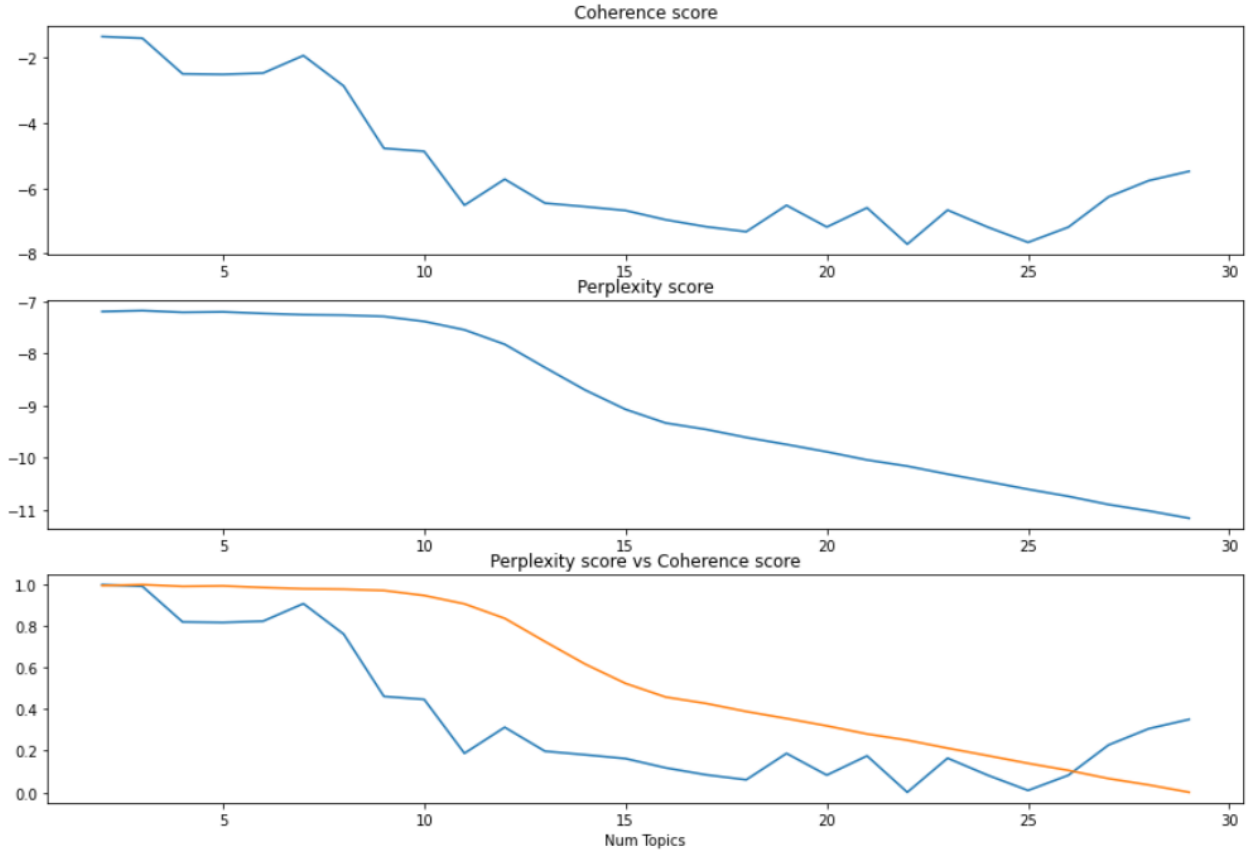
It was build a first model with 10 topics to check the right execution of the model, after that it was finded the optimal number of topics evaluating the coherence's and the perplexity's scores.

- Compute Perplexity & Coherence Score

```
# Compute Perplexity
print('\nPerplexity: ', lda_model.log_perplexity(corpus))

# Compute Coherence Score
coherence_model_lda = CoherenceModel(model=lda_model,
                                     texts=texts_words,
                                     dictionary=id2word,
                                     coherence='c_v')

coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score: ', coherence_lda)
```

```
Perplexity:  -7.446215397734431

Coherence Score:  0.38251700781086495
```

The best solution is located where the coherence curve start to flatten and the perplexity curve start to drop (link: https://www.researchgate.net/publication/342263210_Social-Child-Case_Document_Clustering_based_on_Topic_Modeling_using_Latent_Dirichlet_Allocation). With this approach, it was possible to obtain the optimum number of topics (11 topics) for the model.



2.3.5 Analyzing LDA model results

Now that it was finded the optimal number of topics for th model, it's possible to visualize the topics for interpretability. To do so, it was used a popular visualization package, pyLDAvis which:

[…]

is designed to help users interpret the topics in a topic model that has been fit to a corpus of text data. The package extracts information from a fitted LDA topic model to inform an interactive web-based visualization.

[…]

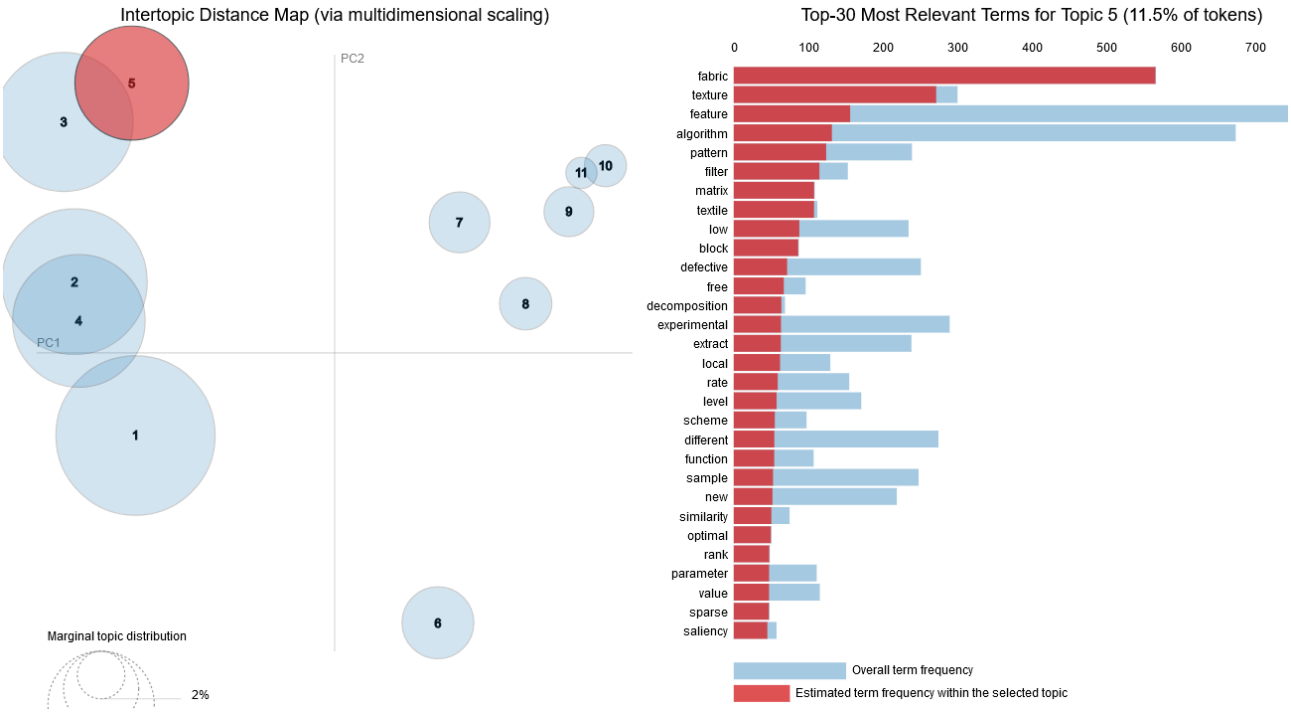
(pyLDAvis official documentation, link: https://pyldavis.readthedocs.io/en/latest/readme.html)

The graph returned from this package provide:

1. better understanding for individual topics;
2. better understanding about the relationships between the topics.

For 1. It is possible to manually select each topic to view its top most frequent and/or "relevant" terms. This can help when you're trying to assign a human interpretable name or "meaning" to each topic.

For 2. exploring the "*Intertopic Distance Plot*" can help to learn about how topics relate to each other, including potential higher-level structure between groups of topics.



For every topics defined by the model, it was ploted the WordCloud picture to better visualize the LDA results and to appreciate the difference between every cluster:

The different topics related to the defects detection generated from the alghoritm are:

0. Automatic processes and visual ispections applications;
1. Textures and matrices applications in fabrics;
2. Optical and thermal methods;
3. Infrastructure's conditions monitoring (sewers, pipes, tunnels, pavements, etc);
4. Deep learning improvements on surfaces;
5. Pixel analysis on surfaces;
6. Use of convolutional networks;
7. Manufacturing applications (bottles, glasses, tiles, ceramics, etc);

8. Printing systems applications;
9. Railways applications;
10. Use of Furier transform.

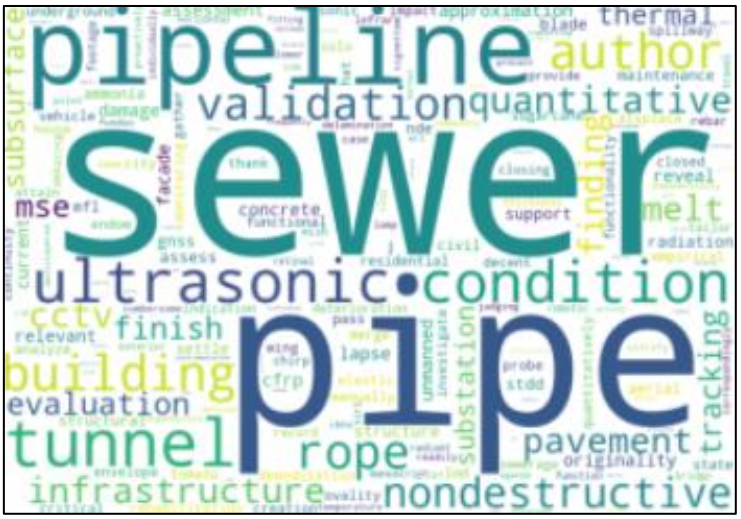
3. Results

3.1 Comments on the results

The analysis performed on the scientific abstracts highlighted different topics related to the “automatic defect detection” themes. The topics identified by the model are very various and are related to different industries and different types of analysis for defect detection from images.

The most of the topics extracted are related to industry sectors: printing, texture analysis, surface inspection etc. The ther big part of results are related to improvement methods to rise the detections check with the application of scientific innovations (Furier transform, convolutional networks, etc.)

Only two topics are related to the infrastructure world but one of these topics have importants words matching to specific terms related to the water utility industries and this is a real important result.



3.2 Usefulness of the results

The results showed by this analysis demonstrated that there are scientific applications of automatic defects detection in the water utility’s world. This result may allow the companies in the sector to launch experimental projects of these new methods.