THE SUMMARIZATION OF ARABIC NEWS TEXTS USING PROBABILISTIC TOPIC MODELING FOR L2 MICRO LEARNING TASKS

by

Elsayed Sabry Abdelaal Issa

AN HLT INTERNSHIP REPORT

SUBMITTED TO THE DEPARTMENT OF LINGUISTICS

THE UNIVERSITY OF ARIZONA

2020

This project was fully or partially funded under a grant from the Institute of International Education (IIE), acting as the administrative agent of the Defense Language and National Security Education Office (DLNSEO) for The Language Flagship (P.I.: Dr. Julio C. Rodriguez). One should not assume endorsement by the Federal Government.

TABLE OF CONTENTS

SECTION 1

INTRODUCTION

The field of Natural Language Processing (NLP) combines computer science, linguistic theory, and mathematics. Natural Language Processing applications aim at equipping computers with human linguistic knowledge. Applications such as Information Retrieval, Machine Translation, spelling checkers, as well as text summarization, are intriguing fields that exploit the techniques of NLP. Text summarization represents an important NLP task that simplifies various reading tasks. These NLP-based text summarization tasks can be utilized for the benefits of language acquisition.

1.1 Motivation

The Arabic Flagship is an overseas summer and capstone program in Morocco administered by the American Councils[1] for qualified American students currently enrolled in a domestic Flagship program. Since students travel to Morocco, they need to gain more exposure to the Moroccan dialect and culture. The Language Flagship Technology Innovation Center (Tech Center) at the University of Hawaii at Manoa is the leading entity in the field of education technology that provides the Flagship programs with the top-notch technologies nationwide to enhance the processes of learning foreign languages. Therefore, the Tech Center started a project that involved building a summarization tool for Arabic news articles from Moroccan newspapers for microlearning tasks for L2 Arabic learners.

[1] https://flagship.americancouncils.org

1.2 Objectives

Given the increasing number of U.S. students learning Arabic and the need for authentic pedagogical materials for language learning, as well as the use of new emerging technologies in language acquisition, the Tech Center's project stresses several objectives. First, it aims at building a database for Moroccan news articles in multiple genres, especially genres that leverage the linguistic and cultural competence of the students. Second, it uses this database to build summaries using Probabilistic Topic Modeling (PTM) that can be used by microlearning tasks. The main objective of these microlearning tasks is to explore the perspectives and the underlying assumptions surrounding several topics and address culturally appropriate ways to react to them. In brief, this project aims at scraping Moroccan websites and preparing the scraped data to provide the authentic and the most probable sentences and phrases that microlearning tasks use. Microlearning refers to relatively small learning units and short-term learning activities.

1.3 Outline of the Report

The report is organized as follows. The introduction (Section 1) describes the motivation behind this report and introduces the objectives. Section 2 lays out the concrete and necessary facts about topic modeling, text summarization, microlearning, and the Arabic language. Section 3 describes the implementation of the summarization task and evaluates the available Arabic NLP tools. Section 4 summarizes the results and future work.

SECTION 2

BACKGROUND

This section is divided into four subsections. It gives an overview of topic modeling, different algorithms of probabilistic topic modeling, text summarization methods, microlearning, and an overview of the Arabic language.

2.1 Probabilistic Topic Modeling

Topic Modeling is the process of extracting topics from a large number of texts and documents. Probabilistic Topic Modeling (PTM) is an unsupervised technique of machine learning. It is used to discover the underlying topics in a text document or across several documents. The basic assumption behind topic modeling is that a document can be represented by a set of latent topics, multinomial distributions over words, and assume that each document can be described as a mixture of these topics (Chang et al., 2009). Each document has then a set of topics and probability distributions associated with them. At the same time, each topic has a set of words and their probabilities of occurrence given that document and topic, i.e., topic models build bags for topics to extract information. In figure 2.1 on the next page, the document is represented by four topics. These topics are 'Arts', 'Budgets', 'Children', and 'Education'. Each topic, in turn, is a bag of words occurring in a document (Blei et al., 2003).

There are several methods and algorithms under topic modeling. In their paper, *A Survey of Topic Modeling in Text Mining*, Alghamdi and Alfalqi (2015) discuss two main categories under the field of topic modeling. The first category, or basic methods, includes Latent Semantic Analysis, Probabilistic Latent Semantic Analy-

"Arts"	"Budgets"	"Children"	"Education"
NEW	MILLION	CHILDREN	SCHOOL
FILM	TAX	WOMEN	STUDENTS
SHOW	PROGRAM	PEOPLE	SCHOOLS
MUSIC	BUDGET	CHILD	EDUCATION
MOVIE	BILLION	YEARS	TEACHERS
PLAY	FEDERAL	FAMILIES	HIGH
MUSICAL	YEAR	WORK	PUBLIC
BEST	SPENDING	PARENTS	TEACHER
ACTOR	NEW	SAYS	BENNETT
FIRST	STATE	FAMILY	MANIGAT
YORK	PLAN	WELFARE	NAMPHY
OPERA	MONEY	MEN	STATE
THEATER	PROGRAMS	PERCENT	PRESIDENT
ACTRESS	GOVERNMENT	CARE	ELEMENTARY
LOVE	CONGRESS	LIFE	HAITI

The William Randolph Hearst Foundation will give $1.25 million to Lincoln Center, Metropolitan Opera Co., New York Philharmonic and Juilliard School. "Our board felt that we had a real opportunity to make a mark on the future of the performing arts with these grants an act every bit as important as our traditional areas of support in health, medical research, education and the social services," Hearst Foundation President Randolph A. Hearst said Monday in announcing the grants. Lincoln Center's share will be $200,000 for its new building, which will house young artists and provide new public facilities. The Metropolitan Opera Co. and New York Philharmonic will receive $400,000 each. The Juilliard School, where music and the performing arts are taught, will get $250,000. The Hearst Foundation, a leading supporter of the Lincoln Center Consolidated Corporate Fund, will make its usual annual $100,000 donation, too.

Figure 2.1: An example article from the AP corpus

sis, Latent Dirichlet Allocation, Correlated Topic model (CTM). The second category, or Topic Evolution Models, includes Topic Over time (TOT), Dynamic Topic Models (DTM), Multiscale Topic Tomography (MTT), Dynamic Topic Correlation Detection (DTCD), Detecting Topic Evolution(DTE).

Our topic models are implemented using Python GenSim library (Řehůřek and Sojka, 2010), which was introduced in 2008 to model similar topics, and it stands for Generate Similar (GenSim)[1]. In GenSim, topic modeling can be done using Latent Semantic Allocation (LSA) or Latent Dirichlet Allocation (LDA). These two algorithms are used to perform classification on documents. On the one hand, Latent Semantic Allocation (LSA) is a mathematical and statistical technique that is used

[1]https://radimrehurek.com/gensim/about.html

for extracting relations of the contextual usage of words in texts. Landauer et al. (1998) identify three steps to employ Latent Semantic Allocation. First, the text is represented as a term-sentence matrix (**n by m matrix**) where each row stands for a unique word and each column stands for the text. Each entry a_{ij} weights the word i in the sentence j. Suppose we have $\mathbf{X}$ matrix, the $\mathbf{t}_i^\top$ (each row) is a vector representation of the words while the $\mathbf{d_j}$ (each column) is a vector representation of the texts. The dot product of the two vectors gives the correlation between the words over the set of texts.

$$\mathbf{X} = \begin{bmatrix} a_{11} & \dots & a_{1j} & \dots & a_{1n} \\ \vdots & \dots & \vdots & \dots & \vdots \\ a_{i1} & \dots & a_{ij} & \dots & a_{in} \\ \vdots & \dots & \vdots & \dots & \vdots \\ a_{m1} & \dots & a_{mj} & \dots & a_{mn} \end{bmatrix} = \mathbf{U} \left[\left[u_1 \right] \dots \left[u_k \right] \right] * \sum \begin{bmatrix} \sigma_1 & \dots & 0 \\ \vdots & \dots & \vdots \\ 0 & \dots & \sigma_k \end{bmatrix} * \mathbf{V}^\top \begin{bmatrix} v_1 \\ \vdots \\ v_k \end{bmatrix}$$

Second, each cell frequency is weighted by the function of (TF-IDF)[2] to express the word importance and the information it carries in the text. Third, LSA applies singular value decomposition (SVD) to the matrix, which is a form of factor analysis. The SVD is used to transform the matrix $\mathbf{X}$ into three matrices: $\mathbf{X} = \mathbf{U} \sum \mathbf{V}^\top$ as illustrated above. The $\mathbf{U}$ is a term-topic (n by m) matrix that contains the weights of words. The $\sum$ is a diagonal (m by m) matrix where each row i corresponds to the weights of a topic j. The $\mathbf{V}^\top$ is the topic sentence matrix. In linear algebra,

[2]TF-IDF stands for term frequency-inverse document frequency, and it is a weight often used in information retrieval and data mining. It is is a statistical measure used to evaluate how important a word is to a document in a corpus. Term Frequency for term (t) is computed as follows: TF(t) = (No. of times t appears in a document)/(total No. of terms in the document). Inverse Document Frequency (IDF) measures how important a term is, and it is measured by the following equation: $\mathbf{idf_t} = \mathbf{log} \frac{\mathbf{N}}{\mathbf{df_t}}$ where N is the total number of documents (Manning et al., 2008).

the SVD is the decomposition of the $\mathbf{X}$ such that $\mathbf{U}$ and $\mathbf{V}$ are orthogonal (word eigenvector) matrices and $\sum$ is a diagonal matrix. In $\mathbf{U}$ and $\mathbf{V}$, the u and v are called the left and right singular vectors where the σs are called the singular values. The resulting matrix $\mathbf{D} = \sum \mathbf{V}^\top$ describes how much a sentence represents a topic (Allahyari et al., 2017).

Latent Dirichlet Allocation (LDA), on the other hand, is a generative probabilistic model of a corpus. The idea is that "documents are represented as random mixtures over latent topics, where each topic is characterized by a distribution over words" (Blei et al., 2003). In our project, we opt for Latent Dirichlet Allocation (LDA) algorithm. It is considered as the simplest topic model (Blei, 2010). Blei et al. (2003) illustrate that LDA assumes the following generative process for each document $\mathbf{w}$ in a corpus D:

1. Choose $N \sim$ Poisson (ξ).
2. Choose $\theta \sim$ Dir(α).
3. For each of the N words w_n:
   (a) Choose a topic $z_n \sim$ Multinomial ($theta$)
   (b) Choose a word w_n from $p(w_n|z_n, \beta)$, a multinomial probability conditioned on the topic z_n.

This can be interpreted as follows: 1) Given a k number of topics, 2) distribute these k topics across document m following a Dirichlet distribution and assign each word a topic, 3) for each word w in document m, assign word w a topic. More formally, a k-dimensional Dirichlet random variable θ can take values in the (k-1)-simplex. This is a k-vector θ that lies in the (k-1) if $\theta_i \geq 0$, $\sum_{i=1}^{k} \theta_i = 1$. The probability density on this simplex is as follows:

$$\mathbf{p}(\theta|\alpha) = \frac{\Gamma(\sum_{i=1}^{k} \alpha_i)}{\Pi_{i=1}^{k} \Gamma(\alpha_i)} \theta_1^{\alpha_1 - 1} ... \theta_k^{\alpha_k - 1},$$

where the parameter α is a k-vector with components $\alpha_i > 0$, and where $\Gamma(x)$ is the Gamma function. Given the parameters α and β, the joint distribution of a topic mixture θ, a set of N topics $\mathbf{z}$, and a set of N words $\mathbf{w}$ is given by:

$$\mathbf{p}(\theta, \mathbf{z}, \mathbf{w}|\alpha, \beta) = p(\alpha|\beta)\Pi_{n=1}^{N} p(z_n|\theta)p(w_n|z_n, \beta),$$

where $p(z_n|\theta)$ is simply θ_i for the unique i such that $\mathbf{z_n^i} = 1$. Integrating over θ and summing over z, we obtain the marginal distribution of a document:

$$\mathbf{p}(\mathbf{w}|\alpha, \beta) = \int p(\theta|\alpha)(\Pi_{n=1}^{N} \sum_{z_n} p(z_n|\theta)p(w_n|z_n, \beta))d\theta.$$

Finally, the probability of the corpus is obtained by taking the product of the marginal probabilities of single documents:

$$\mathbf{p}(D|\alpha, \beta) = \Pi_{d=1}^{M} \int p(\theta_d|\alpha)(\Pi_{n=1}^{N_d} \sum_{z_{dn}} p(z_{dn}|\theta_d)p(w_{dn}|z_{dn}, \beta))d\theta_d.$$

The LDA model is represented as a probabilistic graphical model in the figure below.

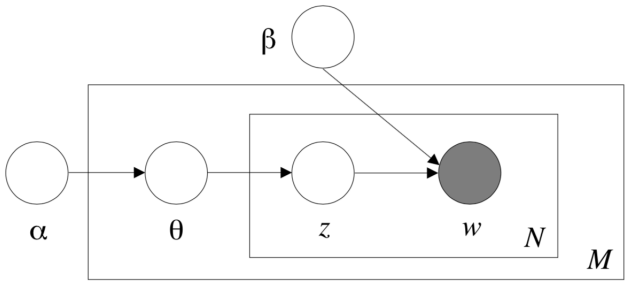

Figure 2.2: Graphical model representation of LDA

The three levels of LDA explained above are represented by this figure. The parameters *alpha* and β are corpus-level parameters. The variables θ_d are the

document-level variables. Finally, the variables z_{dn} and w_{dn} word-level variables (Blei et al., 2003)

2.2 Text Summarization

Text summarization is the process of creating a concise and coherent summary of a longer text while preserving the meaning and the important information in the text (Allahyari et al., 2017). Moreover, Das and Martins (2007) hold that the simple definition captures three essential features that characterize automatic summarization. First, summaries are extracted from a single document or multiple documents. Second, summaries should preserve the important key information. Finally, they should be short. The following figure [3] shows that text summarization can be divided into summaries that are based on the input type, the output type, and the purpose of the summary.

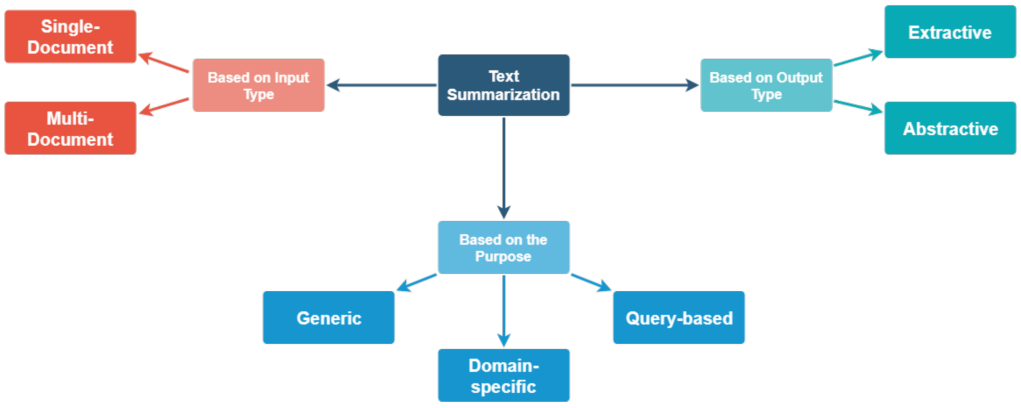

Figure 2.3: The types of text summarization

Based on the output of the summarization task, there are two approaches; extractive and abstractive summarization. The extractive method selects and extracts the more relevant pieces or sentences than others in a longer text.

[3] https://aitor-mir.github.io/summarization.html

Abstractive approaches, on the other side, attempt to generate new content based on the existing one (Das and Martins, 2007).

In this project, we use the extractive summarization approach. This approach work by 1) Constructing an intermediate representation of the input text, which expresses the most salient content. 2) Scoring the sentences based on the representation, which represents how well the sentence explains some of the most important topics of the text. 3) Selecting a summary comprising a number of the top sentences based on their scores (Allahyari et al., 2017).

2.3 Microlearning

Langreiter and Bolka (2006) define microlearning as a term that "reflects the emerging reality of the ever-increasing fragmentation of both information sources and information units used for learning." In other words, microlearning refers to short-term-focused activities on small units of learning content (Hug and Friesen, 2007). From a language acquisition perspective, we define microlearning as a learning activity on small pieces of texts (summaries) based on longer news articles. Summaries of news articles can be used by a microlearning approach to language learning. In 15 minutes a day, a student can gain some linguistic proficiency by reading a summary that contains words with the highest probability in a text. They can learn more effective vocabulary and gain the gist of the entire article. This means that microlearning is a way of teaching and delivering language content to learners in small, concise, and accessible formats. Text summarization and microlearning are motivated by some considerations. First, personal learning content leverages the linguistic proficiency of language learners. If there is an online summarization system for news articles in one language, learners of that language can advance their linguistic as well as cultural competence by reading these summaries on daily basis. Second, exploiting the ever growing online content as a learning resource that

enriches learners' linguistic and cultural competence. Third, summaries used in microlearning activities are heterogeneous and authentic content that can be used in reading comprehension drills.

2.4 The Arabic Language

Arabic is a Semitic language spoken widely in the Middle East and several African countries. Arabic has two standard forms, Classical Arabic which is not widely spoken today and Modern Standard Arabic that is derived from Classical Arabic and contains more modern vocabulary. Arabic also consists of a wide range of varieties, which are considered the spoken Arabic dialects that all native speakers learn as their mother tongue before they begin their formal education (Holes, 2004).

Arabic linguistic features differ in morphology, syntax, and writing systems from other languages. These linguistic features make it one of the complex languages that poses a challenge not only to second language learners but also to Natural Language Processing tasks. Arabic derivational or lexical morphology deals with how Arabic words are formed. Derivational morphology follows the 'root and pattern' principle where the root is a semantic abstraction consisting of three consonants from which words are derived following certain patterns or templates (Holes, 2004). Arabic inflectional morphology, on the other side, deals with how Arabic words interact with syntax where they inflect to express grammatical categories such as gender, number, person, case and tense. This non-linearity in Arabic derivational morphology, the sophisticated principles of derivation, and inflectional endings complicate the attempts to morphological analysis, stemming or lemmatization.

Arabic sentence structure posits various types of complexities. Most Arabic sentences are classified broadly into nominal, sentences that involve a subject and a predicate, or verbal sentences. Verbal sentences, in turn, contain a verb, a subject and/or an object. This entails that Arabic sentences follow either SVO or VSO word order. However, Arabic sentences exhibit preposing and postposing sentence

constituents, which results in VSO, OVS, SOV word orders. Furthermore, Arabic sentences exhibit an interesting phenomenon. They tend to connect using conjunctions such as *waa* (*and*), and *ʔaw* (*or*), as well as commas to form run-on sentences. Run-on sentences may constitute one paragraph or, sometimes, an entire page. This phenomenon creates several problems for natural language processing tasks, and it requires a highly efficient sentence splitter that targets sentence conjunctions to form short meaningful sentences.

SECTION 3

IMPLEMENTATION AND PROBLEMS

This section discusses the implementation of the summarization method and the problems incurred during the process of the design. It is divided into two subsections. The first subsection focuses on the implementation of the summarization method. The second subsection discusses and evaluates the different Arabic NLP tools used and tested during the implementation of the topic model.

3.1 Implementation of the Summarization Method

This subsection discusses the design of the Moroccan News Corpus using the Scrapy framework, and the implementation of both the LDA algorithm and the summarization algorithm. It concludes with an evaluation of the results of the summarized texts.

3.1.1 The Design of the Moroccan News Corpus

This part discusses the process of web scraping/crawling used in this project, and the design of the Moroccan News Corpus that involves using Scrapy's spiders and crawlers as well as the CSS and XPath selectors. Web scraping, on the one hand, refers to the process of data extraction from a website automatically using scraping modules. On the other hand, web crawling is the process of following the links we select. These definitions are fundamental to the case of scraping news articles for the Moroccan newspapers because we found that web crawlers are more effective in following news links than Scrapy spiders.

Scrapy is a free open source and collaborative framework for extracting data from websites using CSS and XPath selectors.[1] Scrapy has five spiders (`scrapy.spider`) that determine both the behavior and the aim of scraping. These are `scarpy.Spider` which is used to scrape data from websites while `CrawlSpider` that is used to follow links on the website. `XMLFeedSpider`, on the one hand, scrapes XML documents on the websites while `CSVFeedSpider` scrapes comma-separated value data on the other hand. Finally, `SitemapSpider` handles all the URLs in the sitemap. We used `scrapy.Spider` and `CrawlSpider` in our project to scrape/crawl the Moroccan newspapers websites.[2] Additionally, `Rule` and `LinkExtractor` objects are used to build rules for the crawlers using regular expressions to crawl the newspaper links and extract the relevant data.

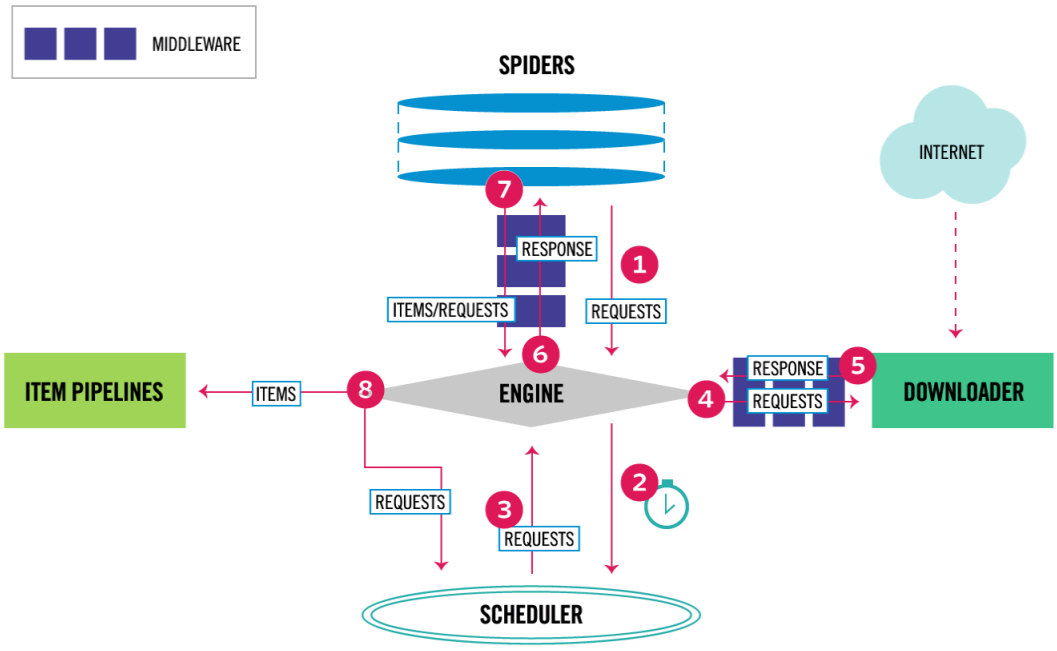

Figure 3.1: Scrapy architecture

[1] https://scrapy.org

[2] https://docs.scrapy.org/en/latest/topics/spiders.html

The above diagram[3] shows an overview of the scrapy architecture as well as the data flow inside it. The scrapy algorithm as well as data flow are explained by the following steps. First, the **engine** gets the initial **requests** to crawl from the **spider**. Second, the **engine** schedules the **requests** in the **scheduler** and asks for the next **requests**. Third, the **scheduler** returns the next **requests** to the **engine**. Fourth, the **engine** sends the **requests** to the **downloader** through the **downloader middleware**. Fifth, once the page finishes downloading, the **downloader** generates a **response** and sends it to the **engine** through the **downloader middleware**. Sixth, the **engine** receives the **response** from the **downloader** and sends it to the **spider** for processing through the **spider middleware**. Seventh, the **spider** processes the **response** and returns scraped items and new **requests** to the **engine** through the **spider middleware**. Eighth, the **engine** sends processed items to the **item pipelines**, then send processed **requests** to the **scheduler** and asks for other **requests** to crawl. Ninth, the process repeats until there are no more requests from the **scheduler**.

To create a project, we use the terminal to create a `CrawlSpider` using the following commands.

```
scrapy startproject <name of the project>
scrapy genspider <name of spider> <website domain> -t crawl
```

The first command creates a project folder with a name of our choice. After changing the directory to the project folder, the second command is run to create a spider of the kind `CrawlSpider` to crawl the links of the website. The following code shows the `CrawlSpider` object under which there are the name of the spider, the allowed domains, and the start URLs. The name of the spider should be unique and different from the name of the project folder. We use it later to call the spider/crawler for scraping/crawling.

[3]https://docs.scrapy.org/en/latest/topics/architecture.html

```
import scrapy
from scrapy.linkextractors import LinkExtractor
from scrapy.spiders import CrawlSpider, Rule
from ..items import AkhbaronaItem
from scrapy.loader import ItemLoader

class GoodCrawlerSpider(CrawlSpider):
    name = "akhbar_crawler"
    allowed_domains = ["www.akhbarona.com"]
    start_urls = ["https://www.akhbarona.com/economy/index.1.html",
                  "https://www.akhbarona.com/politic/index.1.html",
                  "https://www.akhbarona.com/national/index.1.html",
                  "https://www.akhbarona.com/sport/index.1.html",
                  "https://www.akhbarona.com/world/index.1.html",
                  "https://www.akhbarona.com/health/index.1.html",
                  "https://www.akhbarona.com/technology/index.1.html",
                  "https://www.akhbarona.com/culture/index.1.html",
                  "https://www.akhbarona.com/last/index.1.html"]
```

The scrapy `CrawlSpider` inherits from the `scrapy.Spider` class to scrape data from the website. Since news websites include hundreds of links to new articles as well as future articles, the `CrawlSpider` proves the best choice along with `Rule` and `LinkExtractor` objects. Regular expressions are used as well in building the rules for the spider. According to the following rule, the spider crawls all the above urls starting from the first page (index 1) to the last page for each category (i.e., economy, politic, ... etc.) in the news website.

```
rules = (Rule(LinkExtractor(allow=r"(economy|politic|national
        |sport|world|health|technology|culture|last).*"),
```

```
        callback="parse_item", follow=True),)
```

The `callback` function in the above code parses the response (web page) and returns an item in the `parse_item` method below.[4] Spiders are managed by Scrapy engine. This engine first makes requests from URLs specified in start urls and passes them to a downloader which is responsible for fetching web pages and feeding them to the engine. When downloading finishes, the callback specified in the request is called, which is the `parse_item` in the above code snippet. If the callback returns another request, the same thing is repeated. If the callback returns an *Item*, the item is passed to a pipeline to save the scraped data.

```
def parse_item(self, response):
            item = AkhbaronaItem()
            item["content"] = response.xpath('//*[@id="article_body"]
                              /p/text()').extract()
            with open("akhbar.txt","a") as f:
                f.write("content: {0}\n".format(item["content"]))
            yield item
```

Scrapy uses either XPath or CSS selectors to select specific parts of the HTML document to extract the relevant data. XPath, which stands for XML Path Language, is a language for selecting nodes in XML (Extensible Markup Language) documents, which can also be used with HTML (HyperText Markup Language). CSS, which stands for Cascading Style Sheet, is a language for applying styles to HTML documents. It defines selectors to associate those styles with specific HTML elements.[5]

One of the helpful tools in building the `response.xpath` or `response.css` is the `scrapy shell` from the command line, which opens a shell where

[4] http://doc.scrapy.org/en/latest/topics/spiders.html

[5] http://doc.scrapy.org/en/latest/topics/selectors.html

URLs can be fetched. The use of the scrapy shell and the Chrome developer tools help write CSS or XPath selectors. The following code `response.xpath('//*[@id="article_body"]/p/text()').extract()` extracts all news articles from the news website indicated in the crawler above puts them in items called "content," and stores the articles in json file after invoking the following code on the command line.

```
scrapy crawl <name of the spider> -o <name of the file>.json
```

Scrapy framework is more powerful and faster than beautiful soup and selenium, which are other frameworks or modules used in scraping Internet data. In the first week of the internship, we tried beautiful soup and selenium, especially to scrape heavy javascript news websites. However, they provided neither optimal nor quick results like scrapy. The use of these modules requires writing more extended code, while scrapy uses the minimum lines of code for optimal results. One of the advantages of scrapy is that it represents a compatible framework with other modules such as beautiful soup and selenium. In other words, the three modules can be used together to scape heavy javascript websites in particular. The Moroccan News Corpus consists of articles scarped from 20 newspapers. These articles handle several topics such as politics, economy, sports, culture, health, etc.[6]

3.1.2 The Latent Dirichlet Allocation Algorithm

Since topic modeling is the process to extract the hidden topics from large volumes of texts, there are several algorithms for topic modeling, as mentioned in section 2.1. in chapter 2. In the following discussion, we implement a Latent Dirichlet Allocation (LDA) model. The quality of the extracted clear topics depends on the quality of text preprocessing, as well as the optimal number of topics. The

[6]The Moroccan News Corpus and all the relevant scrapy crawlers are available on GITHUB via the following link: https://github.com/Elsayedissa/The-Moroccan-News-Corpus

following discussion illustrates the preprocessing of the raw text and the building of the model using the LDA algorithm.

This first part of the following code reads all json files, that contain the scraped data as illustrated in the previous subsection, and confirms that these files exist in the directory by printing a list of them. Every json file consists of several dictionaries where each dictionary represents a document (text or news article). In other words, the json dictionary consists of a key that is "content" and a value. The "content" is the item assigned to scrapy Items class in the scrapy code (see section 3.1.1). The dictionary value is the news article scarped from the website. After reading the json files, the second code snippet below appends all these dictionaries to `documents= [ ]` and outputs a list of dictionaries.

```
import os
import json
path_to_json = "~/path to json files"
json_files = [pos_json for pos_json in os.listdir(path_to_json)
              if pos_json.endswith(".json")]
print("Reading the following files: ", json_files)

documents=[]
# reading the json files in the specified directory
for index, js in enumerate(json_files):
    with open(os.path.join(path_to_json, js)) as json_file:
        json_data = json.load(json_file)
        documents.append(json_data)
```

The following code iterates through these news articles stored in `documents`, cleans the text from the non-Arabic text, digits, punctuation symbols, and other encoded Arabic words, and splits them into tokens. It also appends these tokens

to the list `tokens`.The code also cleans the data from empty lists that are created during the process of crawling. The final product is the `cleaned_data`, which is a list of lists of tokens.

```
tokens = []
for docs in documents:
    for doc in docs:
       for line in doc["content"]:
           text = re.sub(r"[\d+ a-zA-Z? & , \xd8 << >> . :"]', " ", line)
           tkns = text.split()
           tokenss = []
           for token in tkns:
               tokenss.append(token)
           tokens.append(tokenss)
cleaned_data = [item for item in tokens if item != []]
```

After cleaning the data, the following code removes stop words from the Arabic text. Arabic stop words in the NLTK corpus were not comprehensive, so that we added more stop words to the corpus to improve the preprocessing of the raw text. In addition to updating the NLTK corpus with the necessary Arabic stop words, we also used the `extend()` method to add stop words that we find during the first evaluations of the model results. Our approach here added stop words that affect the performance of the model. Most of these words are the names of the newspapers, countries and other proper names due to our inability to find a Named Entity Recognition (NER) tool for Arabic and integrate it in our code.

```
from nltk.corpus import stopwords
stop_words = stopwords.words("arabic")
stop_words.extend([])
# get rid of the stop words
```

```
no_stopwords_data = [[word for word in doc if word not in stop_words
                     if len(word)>3] for doc in cleaned_data]
```

The following code serves as the lemmatizer for the Arabic language. The NLTK ISRIS Stemmer is a light stemming system that does not use root dictionary.[7] This light stemmer removes diacritics, prefixes and suffixes. Therefore, it is a light stemmer that deals with linear morphology. Other stemmers deal with non-linear morphology where they render the root words or the stems into their consonantal roots such as the Khoja stemmer (Khoja and Garside, 1999).

The ISRIS stemmer performed poorly because we could not integrate other stemmers into our models because either they are unavailable or perform poorly as well. The difference between light stemmers and other stemmers will be highlighted in section 3.3., as well as an evaluation of Arabic natural language processing tools that we used in this project. The final product is the list `lemmatized_data`, which the lemmas from the texts.

```
from nltk.stem.isri import ISRIStemmer
stemmer = ISRIStemmer()
lemmatized_data = []
for items in bigram_data:
    lemmas = []
    for token in items:
        #remove the three-letter and two-letter prefixes
        token = stemmer.pre32(token)
        # removes the three-letter and two-letter suffixes
        token = stemmer.suf32(token)
        # removes diacritics
        token = stemmer.norm(token, num=1)
```

[7] https://www.nltk.org/_modules/nltk/stem/isri.html

```
        lemmas.append(token)
    lemmatized_data.append(lemmas)
print (lemmatized_data[0:2])
```

The lemmatized data is bigramed using gensim's built-in function `Phrases()` that concatenates words into bigrams. Before implementing the LDA model, the lemmatizer and the bigramer are used interchangeably, depending on the LDA's results. Therefore, we implemented two LDA models. The first model uses the lemmatizer first, and the bigramer benefits from the lemmatized data. The second model uses the bigramer first; then, the lemmatizer lemmatizes the bigrams. We found that the results of the LDA model that uses the first approach are better than the results of the LDA model that uses the second approach. Therefore, we adopt the first approach in this report as the following code shows.

```
# create the bigrams given a minimum count
bigrams = gensim.models.Phrases(lemmatized_data, min_count=5)
print (bigrams)
bigrams_model = gensim.models.phrases.Phraser(bigrams)
print (bigrams_model)
bigram_data = [bigrams_model[doc] for doc in lemmatized_data]
print (bigram_data[0:2])
```

The core concepts in the Gensim package are dictionaries, corpora, vector space model, bag of words vector and the topic model.[8] By performing these two steps (the lemmatization and the bigrams) in the previous two snippets of code, the data is ready. Gensim's `Dictionary` and `doc2bow` objects convert the data to unique IDs. In other words, they create the dictionary and the corpus (bag of words). The dictionary is a bag of words created from lists of sentences, and the `doc2bow` is used

[8]https://radimrehurek.com/gensim/intro.html

to give these words unique numerical IDs. The corpus is the input collection of texts.

```
# the dictionary
dictionary = corpora.Dictionary(bigram_data)
# the corpus
corpus = [dictionary.doc2bow(d) for d in bigram_data]
print (corpus[0:2])
```

Now it is time to design our LDA model. Most of the `LdaModel` parameters are set to the default values as indicated by Gensim's documentation.[9] However, the most critical parameter is the `num_topics` (the number of topics) parameter. Our topic model will provide the topic keywords for each topic and the percentage of contribution of topics in each document.

```
#the model
print ("Please Wait, Printing Topics ... ")
for k in [10, 20, 30, 40]:
    lda = gensim.models.ldamodel.LdaModel (corpus=corpus,
                                           id2word=dictionary,
                                           num_topics=k,
                                           random_state=100,
                                           update_every=1,
                                           chunksize=100,
                                           passes=10,
                                           alpha='auto',
                                           per_word_topics=True)
    print ("Number of topics %d" % k)
```

[9] https://radimrehurek.com/gensim/models/ldamodel.html

```
print ("perplexity: %d" % lda.log_perplexity(corpus))
coherence=gensim.models.CoherenceModel(model=lda, corpus=corpus,
                                       coherence="u_mass")
print ("coherence: %d" % coherence.get_coherence())
```

To choose the best number of topics or, rather optimize the number of topics, we used similarity measures such as perplexity and coherence score. Perplexity, on the one hand, is a measurement of the performance of a probability model in predicting a sample, or it refers to the log-averaged inverse probability on unseen data (Hofmann, 1999). It is the average branching factor in predicting the next word.

$$\mathbf{Per} = \sqrt[N]{\frac{1}{P(w_1 w_2 ... w_n)}}$$

where N is the number of words. The logarithmic version of perplexity is expressed as follows:

$$\mathbf{Per} = \mathbf{2^{-(1/N)} \sum log_2 p(w_i)}$$

where the exponent is the number of bits to encode each word. Since perplexity is a measurement of the good performance of a probability distribution in predicting a sample as defined above. So, the log perplexity of a probability distribution p can be defined as:

$$\mathbf{2^{H(p)} = 2^{-\sum_x p(x) log_2 p(x)}}$$

where the H(p) is the entropy[10] of the distribution and x ranges over events. We try to capture the degree of uncertainty of a model in assigning probabilities to a

[10]Entropy is a way of quantifying the complexity of a message that would be sent between a sender and a receiver over a noisy communication channel (Shannon, 1948, 1951). Therefore, lower entropy means lower perplexity.

text. If the model has higher probabilities, then it has lower perplexity. In GenSim, perplexity is calculated to return per-word likelihood $\mathbf{2}^{(-\mathbf{bound})}$ using a chunk of documents as evaluation corpus. On the other hand, coherence score measures a single topic by measuring the degrees of similarity between high scoring words in this single topic (Röder et al., 2015). The coherence module in GenSim library is an implementation of Röder et al. (2015) paper.

We select the model with the highest coherence score. The following figure shows the results of the topic model concerning the different number of topics. The following two figures show pyLDAis plots for 20 and 14 topics, respectively.

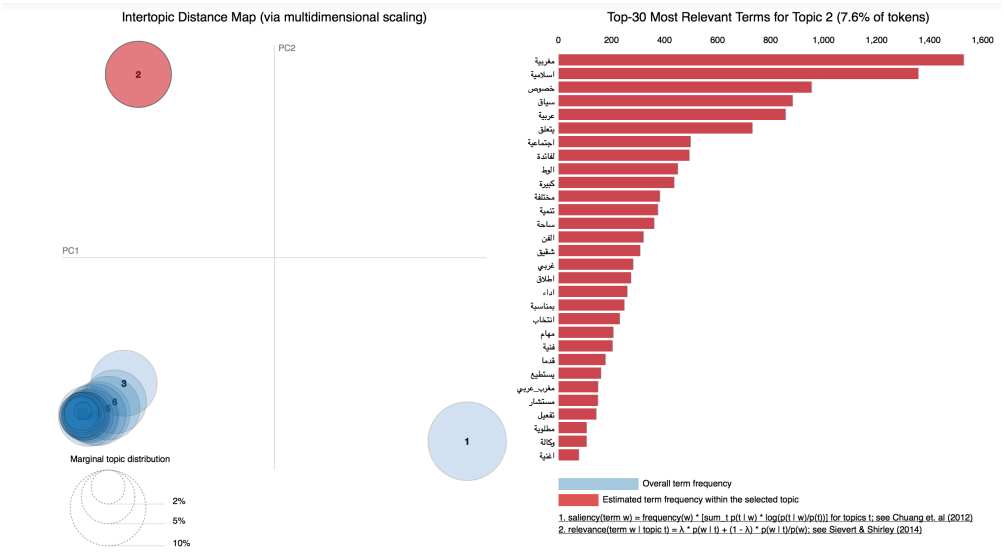

Figure 3.2: A pyLDAVis visualization of an LDA Model of 20 topics

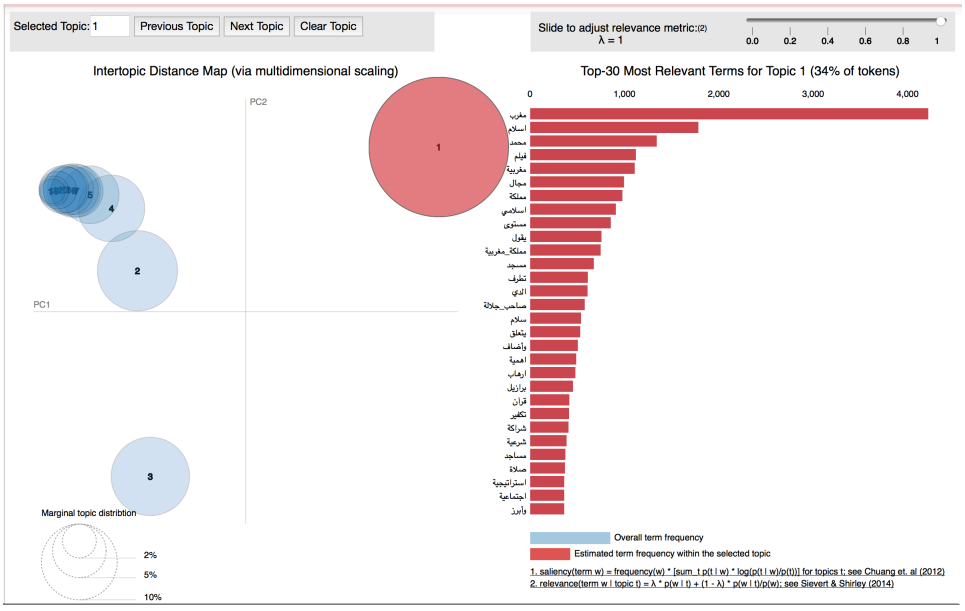

Figure 3.3: A pyLDAVis visualization of an LDA Model of 14 topics

The model with 14 topics performs better than the model with 20 topics. The number of topics parameter represented a critical problem for our topic model. Every time we run the model on new data, we change the number of topics to obtain the best results. We attribute this to several reasons. First, the bad performance of both the lemmatizer and the bigramer affects the choice of the number of topics. Second, the Arabic extended run-on sentences contribute to this choice because we use an extractive approach to text summarization. We tried several number of topics, and we reached the following conclusion. Since we use the extractive approach to text summarization and since we use these summaries for L2 micro-leaning tasks, we decided to get summaries per topic. This gives us flexibility in choosing the number of topics, that in turn provides us with the number of summaries. Our final decision involved feeding the topic model with no more than 2000 documents (i.e., news articles) and setting the number of topics to 100. The summarization algorithm results in 100 summaries, and this is what we mean by summaries per topic. On the other hand, summaries per document (i.e., news article) can involve more processing

of the Arabic sentence which is beyond the scope of this project. In short, summaries per topic provides us with the most prominent sentence that expresses the entire idea behind the news article without any further processing of the sentence, and this is optimal for L2 microlearning tasks.

3.1.3 The Summarization Algorithm

As discussed in section 2.2 in chapter 2, we adopt an extractive approach to text summarization[11]. The basic steps in the summarization pipeline involve: 1) constructing an intermediate representation of the input text, which expresses the most salient content. 2) Scoring the sentences based on the representation, which represents how well the sentence explains some of the most important topics of the text. 3) Selecting a summary comprising a number of the top sentences based on their scores.

However, our summarization algorithm involved using the intermediate representation (the result of the LDA) to reveal the most dominant topics. The following code[12] is a function that gets the most dominant topic in a topic model.

```
import pandas as pd
print ("Please Wait, Printing the Dominant Topics ... ")
def format_topics_sentences(ldamodel=None, corpus=corpus, texts=cleaned_data):
    sent_topics_df = pd.DataFrame()
    # Get main topic in each document
    for i, row_list in enumerate(ldamodel[corpus]):
        row = row_list[0] if ldamodel.per_word_topics else row_list
```

[11] All the codes for the LDA model, the summarization codes, and the evaluated summaries are available on GITHUB via the following link: https://github.com/Elsayedissa/Arabic-News-Summaries

[12] this code is extracted from https://www.machinelearningplus.com/nlp/topic-modeling-gensim-python/#18dominanttopicineachsentence

```
        print(row)
        row = sorted(row, key=lambda x: (x[1]), reverse=True)
        # Get the Dominant topic, Perc Contribution and
        Keywords for each document
        for j, (topic_num, prop_topic) in enumerate(row):
            if j == 0:  # => dominant topic
                wp = ldamodel.show_topic(topic_num)
                topic_keywords = ", ".join([word for word, prop in wp])
                sent_topics_df = sent_topics_df.append(pd.Series([int(topic_num),
                                                        round(prop_topic,4),
                                                        topic_keywords]),
                                                        ignore_index=True)
            else:
                break
    sent_topics_df.columns = ["Dominant_Topic",
                              "Perc_Contribution",
                              "Topic_Keywords"]
    # Add original text to the end of the output
    contents = pd.Series(texts)
    sent_topics_df = pd.concat([sent_topics_df, contents], axis=1)
    return(sent_topics_df)
df_topic_sents_keywords = format_topics_sentences(ldamodel=lda,
                                                  corpus=corpus,
                                                  texts=cleaned_data)
# Format the dominant topics and put them in a dataframe
df_dominant_topic = df_topic_sents_keywords.reset_index()
df_dominant_topic.columns = ["Document_No",
                             "Dominant_Topic",
```

```
                                "Topic_Perc_Contrib",
                                "Keywords",
                                "Text"]
#print (df_dominant_topic)
print ("A sample of the keywords \n", df_dominant_topic["Keywords"][:1])
print ("A sample of the text \n", df_dominant_topic["Text"][:1])

df_dominant_topic["Text"].to_csv("ss.csv", sep = "\t", header = True,
                                     index = False,encoding = "utf-8")
# save to file
#df_dominant_topic.columns["Text"].to_csv("dominant_topics_lda.csv",
                                              sep="\t",header = True,
                                              index = False,
                                              encoding = "utf-8")
```

The above function organizes the results in a data frame with five columns; document number, the dominant topic, the topic percentage contribution, the keywords, and text. The following figure shows the final results.

	Document_No	Dominant_Topic	...	Keywords	Text
0	0	1.0	...	[يونيو), جيدة, اولي, محاكمة, مغربي, يبلغ, تور	...حكومة, محمد, رئيس_حكومة, عدالة_تنمية, اصالة_مع...
1	1	5.0	...	[وحسب_مصادر, يومية_"الصباح", يونيو), جنس, احد	...فريق, مغربية, جديد, رباط, منتخب, مستوى, جديدة,...
2	2	5.0	...	[وانطلق, المدى, موقوق, ماضي_يونيو), "الصباح", ش	...فريق, مغربية, جديد, رباط, منتخب, مستوى, جديدة,...
3	3	2.0	...	[وأوردت, مصادر, قضائية, المدى, ليسقط, شرطة, جرأ	...رئيس, مجال, اولي, سابقة, قناة, ثانية, يتعلق, م...
4	4	5.0	...	[وواصل, المدى, عملياته, لإسقاط, اجانب, سالفة, ا	...فريق, مغربية, جديد, رباط, منتخب, مستوى, جديدة,...
5	5	4.0	...	[وتوصلت, عناصر_شرطة, قضائية, هوية, المدى, احالة	...مغربي, اللي, ماي), مجموعة, عملية, لاعب, قرار, ...
6	6	5.0	...	[وجرى, تقديم, المدى, ماضي_يونيو), وكيل, اودعه	... ,فريق, مغربية, جديد, رباط, منتخب, مستوى, جديدة,...
7	7	4.0	...	[يشارك, فيصل, لاعب, منتخب, اسود_اطلس, مباراة	... ,مغربي, اللي, ماي), مجموعة, عملية, لاعب, قرار, ...
8	8	4.0	...	[وحسب, مجلة, "فرانس, فوتبول", فرنسية, فريق, نا	...مغربي, اللي, ماي), مجموعة, عملية, لاعب, قرار, ...
9	9	7.0	...	[وأكدت, مجلة, ادارة, كناري, تريد, حصول, صاحب]	الله, وطنية, ماضي, فنانة, موقع_"كيفاش", ادارة,...
10	10	4.0	...	[ونشر_موقع, دبيبغ", فرنسي, تقريرا, ادارة, فريق	...مغربي, اللي, ماي), مجموعة, عملية, لاعب, قرار, ...
11	11	3.0	...	[وأكدت_مصادر, لموقع_"كيفاش", نادي, خيتافي, اسب	...الفن, مصدر, فرنسا, نادي, قامت, امريكية, اسماء,...
12	12	4.0	...	[وأشارت, مصادر, فريق, عاصمة, مدريدية, لأسباب	... ,مغربي, اللي, ماي), مجموعة, عملية, لاعب, قرار, ...
13	13	4.0	...	[ونفت, مصادر, توصل, فيصل, اتفاق, فريق, مشيرة	... ,مغربي, اللي, ماي), مجموعة, عملية, لاعب, قرار, ...
14	14	3.0	...	[ونقلت, مجلة, فرنسية, سابقا, تصريحا, لفجر, يؤك	...الفن, مصدر, فرنسا, نادي, قامت, امريكية, اسماء,...
15	15	5.0	...	[يذكر, فيصل, يملك, عقدا, فريق, فرنسي, بنتهم]	فريق, مغربية, جديد, رباط, منتخب, مستوى, جديدة,...
16	16	5.0	...	[قررت, مصرية, ترحيل, مشجع, جزائري, بلاده, لافت	...فريق, مغربية, جديد, رباط, منتخب, مستوى, جديدة,...
17	17	4.0	...	[مشجع, مذكور, صورة, رفقة, مشجع, يحمل, لافتة, م	...مغربي, اللي, ماي), مجموعة, عملية, لاعب, قرار, ...
18	18	5.0	...	[وانتقل, مشجع, مركز, شرطة, اعتقل, مطار_قاهرة	... ,فريق, مغربية, جديد, رباط, منتخب, مستوى, جديدة,...
19	19	7.0	...	[وأشارت, تقارير, امنية, اوقفت, مشجع, وتقرر, نا	...الله, وطنية, ماضي, فنانة, موقع_"كيفاش", ادارة,...

[20 rows x 5 columns]

	Topic_Num	...	Representative Text
0	0.0	...	[وستهب_رياح, قوية_نوعا, قوية, قطاع_شمالي, سواحل_سهول, وسطى, وكذا, اقاليم_جنوبية, ستهب_ضعيفة, معت] ...وستهب_رياح, قوية_نوعا, قوية, قطاع_شمالي, سواحل_سهول, وسطى, وكذا, اقاليم_جنوبية, ستهب_ضعيفة, معت
1	1.0	...	[الوداد_بيضاوي), وعصام, راقي, (الرجاء_بيضاوي), سعيد, فتاح, (الوداد_بيضاوي), وصفي, كادوم] ...اصباحي, (الوداد_بيضاوي), وعصام, راقي, (الرجاء_بيضاوي), سعيد, فتاح, (الوداد_بيضاوي), وصفي, كادوم
2	2.0	...	[كفاءة, مجال, تربية_تعليم, تضمن, شواهد, تكوين, فحسب, قلنا, سابق, يتعلق, بمياوم, بناء, فلاحة, اشغ] ...كفاءة, مجال, تربية_تعليم, تضمن, شواهد, تكوين, فحسب, قلنا, سابق, يتعلق, بمياوم, بناء, فلاحة, اشغ
3	3.0	...	[وظهر, مدرب, فوزي_بلزرتي, كواحد, اسماء, شغلت, رواد_موقع, "غوغل", تونس, ارتبط, اسمه, الدية, سابقا] ...وظهر, مدرب, فوزي_بلزرتي, كواحد, اسماء, شغلت, رواد_موقع, "غوغل", تونس, ارتبط, اسمه, الدية, سابقا
4	4.0	...	[وحسب_موقع, ذاته, مهاجر_مغربي, بالغ, يعيش, نواحي, برشلونة, وبعد, توصل, مركز, الأم, بشكاي, فتحت] ...وحسب_موقع, ذاته, مهاجر_مغربي, بالغ, يعيش, نواحي, برشلونة, وبعد, توصل, مركز, الأم, بشكاي, فتحت
5	5.0	...	[وواصل, المدى, عملياته, لإسقاط, اجانب, سالفة, اعتقادا, مصالح, توقفه, وأنه, بنفذ, جريمة, كاملة, يح] ...وواصل, المدى, عملياته, لإسقاط, اجانب, سالفة, اعتقادا, مصالح, توقفه, وأنه, بنفذ, جريمة, كاملة, يح
6	6.0	...	[وتفتح, مباراة_توظيف, بموجب, عقود, ستنظم, اكاديمي_جهوية, تربية, حاجيات, فعلية, مترشح_مترشح, موظف] ...وتفتح, مباراة_توظيف, بموجب, عقود, ستنظم, اكاديمي_جهوية, تربية, حاجيات, فعلية, مترشح_مترشح, موظف
7	7.0	...	[ماضي, قررت, تنسيقية_وطنية, اسانذة_متعاقد, اضراب, متتالية, وزارة_تربية, وطنية, لإدماج, ازبد, اسد] ...ماضي, قررت, تنسيقية_وطنية, اسانذة_متعاقد, اضراب, متتالية, وزارة_تربية, وطنية, لإدماج, ازبد, اسد
8	8.0	...	[وسنشأرجح_حرارة, العل_درجة, مرتفع, وسايس, وهضاب_فوسفاط, ووالماس, وشمال_منطقة, شرقية, ساحل_متوسطي] ...وسنشأرجح_حرارة, العل_درجة, مرتفع, وسايس, وهضاب_فوسفاط, ووالماس, وشمال_منطقة, شرقية, ساحل_متوسطي
9	9.0	...	[رئيس_حكومة, بتجاوز, صعوب, واجهت, مشاريع, تعثرت, اخيرا, قدمت, مشروع, ملكي, "منارة, متوسط", وأضا] ...رئيس_حكومة, بتجاوز, صعوب, واجهت, مشاريع, تعثرت, اخيرا, قدمت, مشروع, ملكي, "منارة, متوسط", وأضا

[10 rows x 4 columns]

Figure 3.4: The Results of the Most Dominant Topics

The following code retrieves the most representative sentence/s for each dominant topic model.

```
# most representative sentence for each topic
# Display setting to show more characters in column
pd.options.display.max_colwidth = 0
sent_topics_sorteddf_mallet = pd.DataFrame()
sent_topics_outdf_grpd = df_topic_sents_keywords.groupby
                                        ('Dominant_Topic')
for i, grp in sent_topics_outdf_grpd:
    sent_topics_sorteddf_mallet =
                                pd.concat([sent_topics_sorteddf_mallet,
                                grp.sort_values(['Perc_Contribution'],
                                ascending=False).head(1)], axis=0)
# Reset Index
sent_topics_sorteddf_mallet.reset_index(drop=True, inplace=True)
```

```
# Format
sent_topics_sorteddf_mallet.columns = ["Topic_Num",
                                       "Topic_Perc_Contrib",
                                       "Keywords",
                                       "Representative Text"]
# Show
print (sent_topics_sorteddf_mallet)
# save to file
sent_topics_sorteddf_mallet["Representative Text"].
                            to_csv("sentences1_lda.csv",
                            sep = "\t", header=True,
                            index = False, encoding = "utf-8")
```

3.1.4 Evaluation Method

The evaluation techniques of automatic summarization involve two methods. The extrinsic method, on the one hand, judges the quality of summaries based on how they are helpful for a given task. It involves a comparison between the source document and the summarized document. Then, it measures how many ideas of the source document are covered by the summary or a content comparison with an abstract written by a human. On the other hand, the intrinsic method is based on the analysis of the summary. This method can be further classified into content evaluation and text quality evaluation (Steinberger and Ježek, 2012). The following figure shows the taxonomy of these measures.

In this project, we use the text quality evaluation method that provides four criteria to analyze several linguistic aspects of the text. These four criteria are grammaticality, non-redundancy, reference, and coherence. Grammaticality means that the text should not include any non-textual items such as punctuation errors or incorrect words, while non-redundancy indicates that redundant data should not

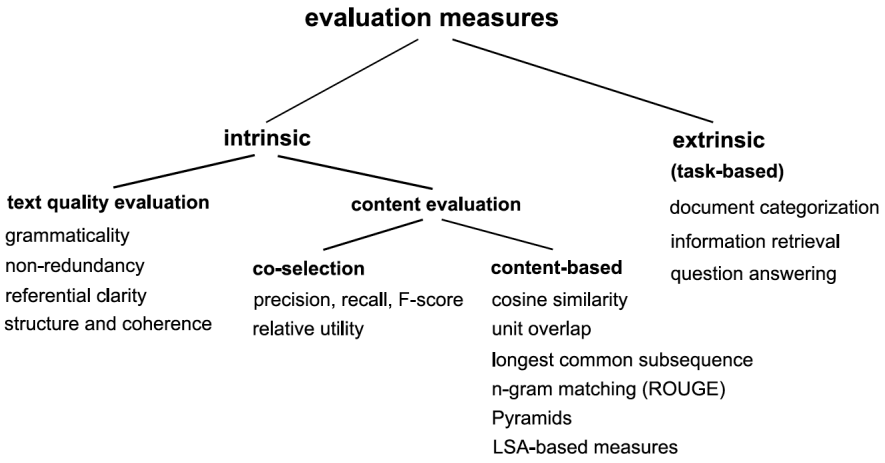

Figure 3.5: The taxonomy of summary evaluation measures

be included. Reference clarity shows that nouns and pronouns should be clearly referred to in the summary, and coherence refers to the coherent structure of the sentences (Steinberger and Ježek, 2012). The summaries are assigned a number on a scale from 0 to 4 (0 = very poor, 1 = poor, 2 = acceptable, 3 = good, 4 = very good).

Scale/Criteria	Grammaticality	Non-redundancy	Reference	Coherence
0	0	0	0	0
1	0	0	0	0
2	0	1	25	0
3	0	11	12	0
4	100	88	63	100

Table 3.1: The evaluation summary for 100 summaries

The above table summarizes the scores given to the 100 sentences that were scored by hand by a human evaluator. Since our summarization method is extractive, which means that we extract the most prominent sentence that summarizes the entire text, as well ones that are error free in grammaticality and coherence. Refer-

ence and anaphors represent a problem. The extracted sentences included anaphors that refer to other antecedents in previous sentences in the text. This concludes that nouns and pronouns are not clearly referred to when we use the extractive method in summarization. Some summaries included redundant data, such as some repeated proper names.

3.2 Arabic NLP Tools Evaluation

In this project, there were several attempts to use Arabic morphological analyzers, POS taggers, and stemmers to prepare the processed raw data in the preprocessing stage to get optimal results when implementing the topic model. This section evaluates some of the tools attempted during the design of the LDA model.

3.2.1 The Earlier Arabic Morphological Analyzers

Morphological Analyzers analyze texts and output the tokens with their morphological information. The Buckwalter Morphological Analyzer BAMA 1.0 (Buckwalter, 2002a) or BAMA/AraMorph 2.0 (Buckwalter (2002b) is one of the first morphological analyzers for Arabic. The engine is written in Perl that accepts input in Arabic Windows encoding (cp1256). BAMA's documentation outlines the running of the analyzer using input/output files through the command line. The documentation neither shows any details about the installation nor the integration of the BAMA into other systems. Three files build the analyzer's lexicon. The first file contains all Arabic prefixes, their combinations, and their English glosses. These prefixes are up to 4 characters long. The second file contains Arabic suffixes and their combinations. These suffixes are up to 6 characters long. The third file contains all Arabic stems that are not specified any length.

The algorithm is simple because most of the rules are already in the three files that constitute the lexicon. The algorithm involves six steps which are tokenization,

word segmentation, dictionary lookup, compatibility check, analysis report and second lookup for orthographic variations. The Arabic words are segmented into their affixes and stems according to these rules: 1) the prefix can be 0 to 4 characters long, 2) the stem does not have any length limit, and 3) the suffix can be 0 to 6 characters long. This segmentation step is done by regular expressions. Then, the dictionary is queried for the existence of the prefix, stem and suffix. If they are found, the algorithm checks if they are compatible with each other. If the prefix, stem and suffix are compatible with each other, the analysis report outputs the different solutions for a word. Finally, if no analysis is found, there will be a spelling check by looking up the orthographic variants.

Due to the problems we encountered in using the analyzer in Arabic Windows encoding (cp1256) and the inability to integrate it into our topic modeling preprocessing step, we looked at its implementation in python. Pyaramorph [13] is a python reimplementation of the Buckwalter morphological analyzer. It contains a broad set of tables to represent information about roots and affixes that contribute to morphological analysis of Arabic words. This morphological analyzer also supported UTF-8 encoding. Although it worked very well in the command line, we could not integrate it into our code. Moreover, the analyzer can handle only short phrases and sentences. It can be installed using (`pip install pyaramorph`) and called from the command line using (`pyaramorph)` as the following figure shows.

[13]https://pypi.org/project/pyaramorph/

```
dhcp-10-134-221-203:~ elsayedissa$ pyaramorph
loading dictPrefixes ... loaded 299 entries
loading dictStems ... loaded 38600 lemmas and 82158 entries
loading dictSuffixes ... loaded 618 entries
Unicode Arabic Morphological Analyzer (press ctrl-d to exit)
$ كتب محمد كتاب
analysis for: كتب ktb
    solution: (كَتَبَ kataba) [katab-u_1]
         pos: katab/VERB_PERFECT+a/PVSUFF_SUBJ:3MS
       gloss: ___ + write + he/it <verb>

    solution: (كُتِبَ kutiba) [katab-u_1]
         pos: kutib/VERB_PERFECT+a/PVSUFF_SUBJ:3MS
       gloss: ___ + be written;be fated;be destined + he/it <verb>

    solution: (كُتُب kutub) [kitAb_1]
         pos: kutub/NOUN
       gloss: ___ + books + ___

analysis for: محمد mHmd
    solution: (مُحَمَّد muHam~ad) [muHam~ad_1]
         pos: muHam~ad/NOUN_PROP
       gloss: ___ + Muhammad;Mohamed + ___

analysis for: كتاب ktAb
    solution: (كِتاب kitAb) [kitAb_1]
         pos: kitAb/NOUN
       gloss: ___ + book + ___

    solution: (كُتّاب kut~Ab) [kut~Ab_1]
         pos: kut~Ab/NOUN
       gloss: ___ + kuttab (village school);Quran school + ___

    solution: (كُتّاب kut~Ab) [kAtib_1]
         pos: kut~Ab/NOUN
       gloss: ___ + authors;writers + ___
```

Figure 3.6: The analysis of a sentence by pyaramorph in the command line

Since we could not integrate the pyaramorph in our code, we tried the AraComLex (Attia et al., 2011). It is an open-source large-scale finite-state morphological analyzer. The analyzer benefits from finite-state technology that handles concatenative and non-concatenatice morphotactics efficiently. It also uses Foma compiler and a lexical database that is edited, validated, and used to update and extend the morphological analyzer automatically (Attia et al., 2011). The lemma form is used as the base form in the implementation of this finite-state transducer. Lexical entries, along with their possible affixes and clitics in a lexc language. A lexc file contains several lexicons connected through 'continuation classes' which determine the path of concatenation. The readme file does not tell us more about the morphological analyzer. However, it gives details about installing the Foma compiler and the commands necessary to run our inquiries in the command line. Although Foma compiler allows sharing this morphological analyzer with third parties, we

were not able to integrate and use it in our code. Therefore, we looked for other tools. Instead of using morphological analysis in the preprocessing stage, we turned our plan into stemmers. Therefore, we used Stanford Arabic NLP toolkit to 1) stem Arabic words 2) tag Arabic words, then splitting sentences based on the position of Arabic conjunctions (waa *and* - ʔaw *or*).

3.2.2 Stanford Arabic NLP Toolkit

Arabic Stanford NLP toolkit[14] consists of a parser, word segmenter, and a part-of-speech (POS) tagger. The original Stanford POS tagger was developed for English and written in Java based on a maximum entropy model as described by Toutanova and Manning (2000). More languages were added, including Arabic, as described in (Diab et al., 2004). The tagger's version for Arabic is trained on the Arabic Penn Treebank (ATB), and the algorithm is based on a Support Vector Machine (SVM) approach, which is a supervised learning algorithm for classification. The algorithm undertakes a number of classification tasks given a number of features extracted from a predefined linguistic context to predict the class of a token (Diab et al., 2004). The accuracy of the tagger is 96.50%, and the readme file shows detailed instructions on how to use the tagger in the command line.

Since Arabic uses run-on sentences, we thought that we could use a POS tagger to tag Arabic texts. Based on these tags, we can split sentences based on 1) Arabic punctuation (commas, periods, exclamation marks and questions marks), 2) Arabic conjunctions (waa *and* - ʔaw *or*). The structure of Arabic run-on sentences involves simple short sentences that are connected using conjunctions such as (waa *and* - ʔaw *or*). Therefore, our approach was that we tag our Arabic texts using Stanford POS tagger; then, based on these tags, we can simplify Arabic run-on sentences by splitting them.

Interestingly, this tagger can be imported and used by NLTK. The following

[14]https://nlp.stanford.edu/projects/arabic.shtml

code snippet shows this process.

```
# import nltk and the stanford POS tagger
import nltk
from nltk.tag import StanfordPOSTagger
# directory to the full tagger in English, Chinese, Arabic ... etc
stanford_dir = "~/stanford-postagger-full-2018-10-16"
# add directory to the Arabic model
modelfile = stanford_dir+"/models/arabic.tagger"
# add directory to the main jar file of the tagger
jarfile = stanford_dir+"/stanford-postagger.jar"
# call the StanfordPOSTagger object
tagger = StanfordPOSTagger(model_filename = modelfile,
                           path_to_jar = jarfile)
# tag Arabic text using split() method
tagger.tag("arabic text goes here".split())
```

Although this code provides good tagging results, we encountered two problems. First, since our topic model uses a large number of texts, the tagger took too much time to tag all these texts. Second, our methodology to split tagged Arabic run-on sentences did not yield optimal results. We changed our minds and decided to follow an extractive method to text summarization, as illustrated earlier in section 2.2. Third, the above code shows that we used the POS tagger models and jar files from local machines, and we did not integrate it fully into our code. Therefore, we thought of the Stanford Core NLP[15] and using their api to either segment or tag words, but we can across another two tools that we discuss in sections 3.3.3 and 3.3.4 below.

[15]https://pypi.org/project/stanfordnlp/, and https://stanfordnlp.github.io/stanfordnlp/

3.2.3 Farasa Arabic NLP Toolkit

Farasa is considered the most up-to-date Arabic NLP toolkit that involves segmentation, spell checking, POS tagging, lemmatization, diacritization, constituency, and dependency parsing, and named entity recognition.[16] Farasa's segmenter is implemented using a support-vector-machine-based segmenter that uses a variety of features and lexicons to rank possible segmentations of a word. These features involve "likelihoods of stems, prefixes, suffixes, their combinations; presence in lexicons containing valid stems or named entities; and underlying stem templates" (Abdelali et al., 2016). Farasa is implemented in Java, and its readme files have outdated information about installation and usage. We downloaded Farasa lemmatizer and segmenter from the website indicated in footnote 15. All our attempts to run them on the command-line failed. We contacted the author to get instructions on how to use them, but we did not receive any responses. We wanted to use Farasa's lemmatizer because it gave the best lemmatization results amongst the tool we used so far.

Although we were not able to use Farasa, the website provided the following code to integrate Farasa modules into other codes from other programming languages. The following code integrates Farasa lemmatizer into our model using python, and connects to its API and passes sentences one by one.

```
import http.client
conn = http.client.HTTPSConnection('farasa-api.qcri.org')
payload = '{\"text\": \"arabic sentence here\"}'.encode('utf-8')
headers = { 'content-type': 'application/json',
            'cache-control': 'no-cache', }
conn.request('POST', '/msa/webapi/pos', payload, headers)
res = conn.getresponse()
```

[16] http://qatsdemo.cloudapp.net/farasa/demo.html

```
data = res.read()
print(data.decode('utf-8'))
```

This code gave good results; however, we encountered some problems. Technically, we could not pass sentence by sentence. It would take too much time. More interestingly, their API[17] disappeared, and has a message that says "coming soon." Although Farasa is the most up-to-date NLP toolkit, we were neither able to use its jar files on the command line due to the outdated documentation nor its API due to updating and construction. The successful solution to the lemmatization problem was the Information Science Research Institute's (ISRI) Arabic stemmer (Taghva et al., 2005).

3.2.4 NLTK ISRI Arabic Stemmer Tool

The ISRI is a root-extraction stemmer without a root dictionary, which serves as a light stemmer. Light stemming means removing diacritics represented by short vowels, stopwords, punctuation, numbers, the definite article, prefixes, and suffixes (Taghva et al., 2005). The characteristics of this stemmer are: 1) it is implemented in python and integrated into NLTK which can be invoked easily 2) it is well-documented in the NLTK documentation.[18] The following code shows a function that uses the ISRI stemmer. Although the stemmer did not yield good results compared to Farasa lemmatizer, it did a relatively good job in the preprocessing stage in our topic model.

```
from nltk.stem.isri import ISRIStemmer
stemmer = ISRIStemmer
def lemmatizer(token):
    # removes the three-letter and two-letter prefixes
```

[17] https://farasa-api.qcri.org

[18] https://www.nltk.org/_modules/nltk/stem/isri.html

```
token = stemmer.pre32(token)
# removes the three-letter and two-letter suffixes
token = stemmer.suf32(token)
# removes diacritics
token = stemmer.norm(token, num=1)
return token
```

3.2.5 Other Tools

There are several other tools that we did not try during the internship. These tools include Standard Arabic Morphological Analyzer (SAMA) (Graff et al., 2009), ElixirFM morphological analyzer (Smrž, 2007), Alkhalil morphological analyzer (Ould Abdallahi Ould Bebah et al., 2010; Boudchiche et al., 2017), Xerox Arabic morphological analysis and generation (Beesley, 1998), Khoja POS tagger (Khoja, 2001), and stemmer (Khoja and Garside, 1999).

SECTION 4

CONCLUSION AND FUTURE WORK

The present report describes the work done in a summer internship for the Language Flagship Technology Innovation Center (Tech Center) at the University of Hawaii at Manoa. The Tech Center is a leading entity in the field of education technology for the Flagship program run by the American Councils. Therefore, the Tech Center took the initiative to build a summarization tool for Moroccan news articles for L2 microlearning tasks.

This project involved building a corpus for Moroccan news articles by scraping Moroccan newspaper websites. It also developed a summarization tool using the Latent Dirichlet Allocation (LDA) algorithm that has an implementation in Python's gensim package. We use the extractive approach by which we extract the most prominent sentence that expresses the entire ideas of a given document. This is done by identifying the most salient topics in a document by the LDA model.

The results indicated that both the LDA and the summarization algorithms provided well-structured summaries for the news articles. The evaluation method showed that the grammaticality and coherence of the summaries scored the highest values, while there remained some redundant data and errors in reference to nouns and pronouns in the extracted sentences. Besides these errors, there are several imperfections of the model, such as the inability to summarize texts per document (i.e., news articles). Instead, we summarized texts based on the number of topics. In other words, the number of topics determines the number of summaries we get for each document.

For future work, more improvements will be made on both the topic model and the summarization model algorithms. For the topic model, we intend to improve

its output by enhancing the preprocessing phase and comparing these output to other latent algorithms such as the Latent Semantic Analysis. The summarization algorithm also needs more work. If we adopted the abstractive method, we would need to do more processing of the Arabic sentences. In other words, the abstractive method provides us with machine-generated summaries which involve more complex technical implementation. Finally, more evaluation measures will be used.

APPENDIX A

An example of document summary

Summary	Document
['شهادات', 'السكان', 'جاءت', 'تفاعلا', 'مع', ' 'المسيرة', 'التي', 'دعت', 'لها', 'تنسيقية', 'أدرار', 'سوس', 'ماسة', 'التي', 'أعلنت', 'أن', 'يوم', 'السبت', 'غشت', 'بأكادير', 'هو', 'يوم', 'للتعبير', 'عن', 'نطالب', 'الساكنة', 'بمشاركة', 'الساكنة', 'ومختلف', 'هيئات', 'المجتمع', 'المدني', 'بجهة', 'سوس', 'ماسة', 'كرد', 'على', 'ما', 'وصفته', 'ب', 'تمادي', 'الحكومة', 'في', 'تنفيذ', 'مخططاتها', 'الرامية', 'الى', 'الإضرار', 'بحقوق', 'الساكنة', 'وممتلكاتها'']	['AHDATH.INFO', '" نعاني مشاكل عديدة، كالنقل المدرسي، وغياب الطرق، عدم توفر الأدوية والأطر الطبية والكثير من المشاكل ... لذلك سأكون من المشاركين في مسيرة أكادير"، يقول أحد الشيوخ في تسجيل تضمن مداخلات عدد من ساكنة جهة سوس ماسة الذين عبروا عن تذمرهم من عدم التجاوب مع ما وصفوه بمطالبهم المشروعة.', 'شهادات السكان جاءت تفاعلا مع المسيرة التي دعت لها "تنسيقية أدرار سوس ماسة"، التي أعلنت أن يوم السبت 17 غشت 2019 بأكادير، هو يوم للتعبير عن نطالب الساكنة بمشاركة الساكنة ومختلف هيئات المجتمع المدني بجهة سوس ماسة، كرد على ما وصفته ب "تمادي الحكومة في تنفيذ مخططاتها الرامية الى الإضرار بحقوق الساكنة وممتلكاتها".', 'ومما طالب به المتضررون، "إيجاد حل نهائي وجدري لملف الخنزير البري والحد من أضراره على الساكنة وممتلكاتهم وتعويض المتضررين ، مع إيجاد حل نهائي وعاجل لاعتداءات الرعاة الرحل وتوفير الأمن والحماية للساكنة وممتلكاتهم."', 'التنسيقية دعت أيضا إلى الحد من ما وصفته ب" التجاوزات التي تتم خلال الترخيص لشركات الاستغلال المنجمي والمعدني في أراضي الساكنة .. مع المطالبة بتعويض السكان الذين حرموا من أراضيهم بسبب مخطط تحديد الملك الغابوي، كما طالبوا بحماية ممتلكاتهم من خلال التحفيظ".', 'مشاكل البنية التحتية من الأمور الملحة بالنسبة للساكنة، لذلك طالبت التنسيقية " بنهج سياسة تنموية عادلة تجاه مختلف الجماعات الترابية بجهة سوس ماسة من خلال الحد من الفوارق المجالية و الاجتماعية والاقتصادية و تمكين ساكنة الجهة من بنية تحتية (طرق ماء كهرباء مرافق عمومية وخدماتية . .) و خدمات صحية و تعليمية تحفظ كرامة المواطن... كما تسجل ساكنة جهة سوس ماسة الاهمال المقصود والمتواصل للحكومة ووزارة الصحة والسلطات المحلية في عدم توفيرها للأمصال المضادة لسموم الأفاعي والعقارب مما أدى إلى ارتفاع وتزايد عدد الوفيات بمنطقة سوس."']

Figure A.1: An example article from a Moroccan newspaper

REFERENCES

Abdelali, A., Darwish, K., Durrani, N., and Mubarak, H. (2016). Farasa: A fast and furious segmenter for arabic. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: Demonstrations*, pages 11–16.

Alghamdi, R. and Alfalqi, K. (2015). A survey of topic modeling in text mining. *Int. J. Adv. Comput. Sci. Appl.(IJACSA)*, 6(1).

Allahyari, M., Pouriyeh, S., Assefi, M., Safaei, S., Trippe, E. D., Gutierrez, J. B., and Kochut, K. (2017). Text summarization techniques: a brief survey. *arXiv preprint arXiv:1707.02268*.

Attia, M., Pecina, P., Toral, A., Tounsi, L., and van Genabith, J. (2011). An open-source finite state morphological transducer for modern standard arabic. In *Proceedings of the 9th International Workshop on Finite State Methods and Natural Language Processing*, pages 125–133. Association for Computational Linguistics.

Beesley, K. R. (1998). Xerox arabic morphological analysis and generation. *Romanisation, Transcription and Transliteration, The Document Company-Xerox*.

Blei, D. M. (2010). Introduction to probabilistic topic models.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022.

Boudchiche, M., Mazroui, A., Bebah, M. O. A. O., Lakhouaja, A., and Boudlal, A. (2017). Alkhalil morpho sys 2: A robust arabic morpho-syntactic analyzer. *Journal of King Saud University-Computer and Information Sciences*, 29(2):141–146.

Buckwalter, T. (2002a). Arabic morphological analyzer (aramorph). *Linguistic Data Consortium, Philadelphia*.

Buckwalter, T. (2002b). Buckwalter arabic morphological analyzer version 1.0. *Linguistic Data Consortium, University of Pennsylvania*.

Chang, J., Gerrish, S., Wang, C., Boyd-Graber, J. L., and Blei, D. M. (2009). Reading tea leaves: How humans interpret topic models. In *Advances in neural information processing systems*, pages 288–296.

Das, D. and Martins, A. F. (2007). A survey on automatic text summarization. *Literature Survey for the Language and Statistics II course at CMU*, 4(192-195):57.

Diab, M., Hacioglu, K., and Jurafsky, D. (2004). Automatic tagging of arabic text: From raw text to base phrase chunks. In *Proceedings of HLT-NAACL 2004: Short papers*, pages 149–152. Association for Computational Linguistics.

Graff, D., Maamouri, M., Bouziri, B., Krouna, S., Kulick, S., and Buckwalter, T. (2009). Standard arabic morphological analyzer (sama) version 3.1. *Linguistic Data Consortium LDC2009E73*.

Hofmann, T. (1999). Probabilistic latent semantic analysis. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 289–296. Morgan Kaufmann Publishers Inc.

Holes, C. (2004). *Modern Arabic: Structures, functions, and varieties*. Georgetown University Press.

Hug, T. and Friesen, N. (2007). Outline of a microlearning agenda. *Didactics of Microlearning. Concepts, Discourses and Examples*, pages 15–31.

Khoja, S. (2001). Apt: Arabic part-of-speech tagger. In *Proceedings of the Student Workshop at NAACL*, pages 20–25.

Khoja, S. and Garside, R. (1999). Stemming arabic text. *Lancaster, UK, Computing Department, Lancaster University*.

Landauer, T. K., Foltz, P. W., and Laham, D. (1998). An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284.

Langreiter, C. and Bolka, A. (2006). Snips & spaces: managing microlearning (on microlearning and microknowledge in a microcontent-based web). In *Micromedia & e-learning 2.0.: Gaining the Big Picture. Proceedings of Microlearning Conference 2006.*, pages 79–97. Innsbruck: Innsbruck UP.

Manning, C. D., Raghavan, P., et al. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

Ould Abdallahi Ould Bebah, M., Boudlal, A., Lakhouaja, A., Mazroui, A., Meziane, A., and Shoul, M. (2010). Alkhalil morpho sys: A morphosyntactic analysis system for arabic texts.

Řehůřek, R. and Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA. `http://is.muni.cz/publication/884893/en`.

Röder, M., Both, A., and Hinneburg, A. (2015). Exploring the space of topic coherence measures. In *Proceedings of the eighth ACM international conference on Web search and data mining*, pages 399–408. ACM.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423.

Shannon, C. E. (1951). Prediction and entropy of printed english. *Bell system technical journal*, 30(1):50–64.

Smrž, O. (2007). Elixirfm: implementation of functional arabic morphology. In *Proceedings of the 2007 workshop on computational approaches to Semitic languages: common issues and resources*, pages 1–8. Association for Computational Linguistics.

Steinberger, J. and Ježek, K. (2012). Evaluation measures for text summarization. *Computing and Informatics*, 28(2):251–275.

Taghva, K., Elkhoury, R., and Coombs, J. (2005). Arabic stemming without a root dictionary. In *International Conference on Information Technology: Coding and Computing (ITCC'05)-Volume II*, volume 1, pages 152–157. IEEE.

Toutanova, K. and Manning, C. D. (2000). Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics-Volume 13*, pages 63–70. Association for Computational Linguistics.