# Extractive and Abstractive Methods For English and Arabic Text Summarization

**By Khaled Madkour**

**Email:** khaled.madkour@gmail.com
**Linkedin:** https://www.linkedin.com/in/khaled-madkour/
**Github:** https://github.com/KhaledMadkour

*April 5th, 2020*

# 1 Introduction

Text summarization refers to the technique of shortening long pieces of text. The intention is to create a coherent and fluent summary having only the main points outlined in the document.

Automatic text summarization is a common problem in machine learning and natural language processing (NLP).

Skyhoshi, who is a U.S.-based machine learning expert with 13 years of experience and currently teaches people his skills, says that “the technique has proved to be critical in quickly and accurately summarizing voluminous texts, something which could be expensive and time consuming if done without machines.

Machine learning models are usually trained to understand documents and distill the useful information before outputting the required summarized texts.

# 2 What’s The Need For Text Summarization?

Propelled by the modern technological innovations, data is to this century what oil was to the previous one. Today, our world is parachuted by the gathering and dissemination of huge amounts of data.

In fact, the International Data Corporation (IDC) projects that the total amount of digital data circulating annually around the world would sprout from 4.4 zettabytes in 2013 to hit 180 zettabytes in 2025. That’s a lot of data!

With such a big amount of data circulating in the digital space, there is a need to develop machine learning algorithms that can automatically shorten longer texts and deliver accurate summaries that can fluently pass the intended messages.

Furthermore, applying text summarization reduces reading time, accelerates the process of researching for information, and increases the amount of information that can fit in an area.

# 3 The Main Types Of Text Summarization

Broadly, there are two approaches to summarizing texts in NLP: **extraction** and **abstraction**.

## 3.1 Extraction-Based Summarization

In extraction-based summarization, a subset of words that represent the most important points is pulled from a piece of text and combined to make a summary. Think of it as a highlighter—which selects the main information from a source text.

In machine learning, extractive summarization usually involves weighing the essential sections of sentences and using the results to generate summaries.

Different types of algorithms and methods can be used to gauge the weights of the sentences and then rank them according to their relevance and similarity with one another—and further joining them to generate a summary. Here's an example:

*Source Text: **Peter and Elizabeth** took a taxi to **attend** the night **party** in the **city**.*
*While in the Party, **Elizabeth** collapsed and was **rushed** to the **hospital**.*

*Summary: Peter and Elizabeth attend party city. Elizabeth rushed hospital.*

As you can see above, the extracted summary is composed of the words highlighted in bold, although the results may not be grammatically accurate.

## 3.2 Abstraction-Based Summarization

In abstraction-based summarization, advanced deep learning techniques are applied to paraphrase and shorten the original document, just like humans do. Think of it as a pen—which produces novel sentences that may not be part of the source document.

Since abstractive machine learning algorithms can generate new phrases and sentences that represent the most important information from the source text, they can assist in overcoming the grammatical inaccuracies of the extraction techniques. Here is an example:

*Source Text: **Peter** and **Elizabeth** took a taxi to **attend** the night **party** in the city.*
*While in the Party, **Elizabeth** collapsed and was rushed to the **hospital**.*

*Summary: Elizabeth was hospitalized after attending a party with Peter.*

Although abstraction performs better at text summarization, developing its algorithms requires complicated deep learning techniques and sophisticated language modeling.

To generate plausible outputs, abstraction-based summarization approaches must address a wide variety of NLP problems, such as natural language generation, semantic representation, and inference permutation; that's why the use of extraction is still popular.

# 4 The Models

In this report I will try to tackle the text summarization problem for two different languages Arabic and English.

I will introduce three different models, two extractive and one abstractive, and apply them to different languages.

## 4.1 Simple Extractive Model

I started by making a simple extractive text summarization.

It is a really simple model where you give scores to words and sentences based on how frequent they are without having meaning behind words by not using any language models.

This can be easily applied to any language but it gives rather dull results due to the lack of understanding of the language itself as it is basically a language agnostic approach.

The goal of this model is not the best or even good results but rather building the foundation and the sitting stone in the project where I can improve from.

### 4.1.1 Getting the Data

In this approach I am going to use two articles from Wikipedia.

One English article article that gives an overview about the 20th century.
Another Arabic one that talks about World War II.

To enable us to fetch the article's text, we'll use the Beautiful Soup library.

The BeautifulSoup library is used for parsing the page while the urllib library is used for connecting to the page and retrieving the HTML.

BeautifulSoup converts the incoming text to Unicode characters and the outgoing text to UTF-8 characters, saving you the hassle of managing different charset encodings while scraping text from the Web.

We'll use the urlopen function from the urllib.request utility to open the web page. Then, we'll use the read function to read the scraped data object. For parsing the data, we'll call the BeautifulSoup object and pass two parameters to it; that is, the article_read and the html.parser.

The find_all function is used to return all the <p> elements present in the HTML. Furthermore, using .text enables us to select only the texts found within the <p> elements.

Read the function "read_wiki" in "preprocessing.ipynb" notebook to check the code for extracting the article from wikipedia.

### 4.1.2 Data Processing

#### *a) For English Text*

To ensure the scrapped textual data is as noise-free as possible, we'll perform some basic text cleaning. To assist us to do the processing, we'll import a list of stopwords from the nltk library.

We'll also import PorterStemmer, which is an algorithm for reducing words into their root forms. For example, cleaning, cleaned, and cleaner can be reduced to the root clean.

#### *b) For Arabic Text*

The main concepts are basically the same. But I added some Arabic stopwords obtained from **here** and used an arabic stemmer from a library called tashaphyne.

### 4.1.3 How the Model Works

The model goes through simple steps:

First, create a dictionary table having the frequency of occurrence of each of the words in the text.

Then we tokenize the article into sentences using nltk library.

To evaluate the score for every sentence in the text, we'll be analyzing the frequency of occurrence of each term. In this case, we'll be scoring each sentence by its words; that is, adding the frequency of each important word found in the sentence.

Importantly, to ensure long sentences do not have unnecessarily high scores over short sentences, we divided each score of a sentence by the number of words found in that sentence.

To further tweak the kind of sentences eligible for summarization, we'll create the average score for the sentences. With this threshold, we can avoid selecting the sentences with a lower score than the average score.

### 4.1.4 Results

Parts from the results:

#### *a) For English Wikipedia article about the 20th century:*

It was the tenth and final century of the 2nd millennium.The century had the first global-scale total wars between world powers across continents and oceans in World War I and World War II. The victorious Bolsheviks then established the Soviet Union, the world's first communist state. In total, World War II left some 60 million people dead. During the century, the social taboo of sexism fell. By the end of the 20th century, women had the same legal rights as men in many parts of the world, and racism had come to be seen as abhorrent. The people of the Indian subcontinent, a sixth of the world population at the end of the 20th century, had attained an indigenous independence for the first time in centuries.

#### *b) For Arabic Wikipedia article about World War II:*

6] .إنشاء الأمم المتحدة[

كانت الإمبراطورية اليابانية قد أعلنت الحرب على جمهورية الصين في 7 يوليو 1937، حيث هدفت إلى السيطرة على آسيا والمحيط الهادي،[7] إلا أن البداية الفعلية للحرب تعتبر في الأول من سبتمبر عام 1939،[8] وذلك عندما قامت ألمانيا باجتياح بولندا، وتوالت بعدها إعلانات الحرب على ألمانيا من قبل فرنسا والمملكة المتحدة. بقيت المعركة الأساسية في الحرب هي بين دول المحور من جهة، والمملكة المتحدة إلى جانب دول الكومنولث من جهة أخرى، إضافة إلى حملة في شمال إفريقيا وحملة أخرى في شرق أفريقيا، إضافة إلى معركة برلين الجوية وقصف لندن، وحملة البلقان، ومعركة المحيط الأطلسي.

Our model summarizes the lengthy Wikipedia article and gives a simplistic overview of the main happenings in the 20th century.

Nonetheless, the summary generator can be improved to make it better at producing a concise and precise summary of voluminous texts.

## 4.2 Extractive Model Using Text Rank Algorithm

We continue to discover new models for extractive text summarization. In this model we are improving things a bit by having two major changes.

First, instead of the last scoring system (weight sentences by how frequent words are) we are using a better scoring algorithm called Text Rank.

Second, we will have better representation of words by using word embeddings for both Arabic and English words.

### 4.2.1 Dataset and Data Cleaning

We are going to use the same two twoWikipedia articles we used in the previous model. We basically have quite the same data cleaning process except the word stemming.

### 4.2.2 How the Model Works

As I mentioned this model is similar to the last one except it uses TextRank for scoring and word embeddings for vector representation of the words. So let's dive a little into those:

#### 4.2.2.1 Word Embeddings

A word embedding is a learned representation for text where words that have the same meaning have a similar representation.

It is this approach to representing words and documents that may be considered one of the key breakthroughs of deep learning on challenging natural language processing problems.

Word embeddings are in fact a class of techniques where individual words are represented as real-valued vectors in a predefined vector space. Each word is mapped to one vector and the vector values are learned in a way that resembles a neural network, and hence the technique is often lumped into the field of deep learning.

The key to the approach is the idea of using a dense distributed representation for each word.

Each word is represented by a real-valued vector, often tens or hundreds of dimensions. This is contrasted to the thousands or millions of dimensions required for sparse word representations, such as a one-hot encoding.

The distributed representation is learned based on the usage of words. This allows words that are used in similar ways to result in having similar representations, naturally capturing their meaning. This can be contrasted with the crisp but fragile representation in a bag of words model where, unless explicitly managed, different words have different representations, regardless of how they are used.

We used 2 word embeddings: one for Arabic and another for English

##### *a) GLOVE (English Word Embedding)*

The Global Vectors for Word Representation, or GloVe, algorithm is an extension to the word2vec method for efficiently learning word vectors, developed by Pennington, et al. at Stanford.

I used pre-trained **Wikipedia 2014 + Gigaword 5** GloVe vectors available here

##### *b) AraVec (Arabic Word Embedding)*

AraVec is a pre-trained distributed word representation (word embedding) open source project which aims to provide the Arabic NLP research community with free to use and powerful word embedding models. The first version of AraVec provides six different word embedding models built on top of three different Arabic content domains; Tweets and

Wikipedia This paper describes the resources used for building the models, the employed data cleaning techniques, the carried out preprocessing step, as well as the details of the employed word embedding creation techniques.

I used pre-trained **Wikipedia-SkipGram 300 vec size unigram model** available here or [github]

#### 4.2.2.2 TextRank Algorithm

The first step is of course to extract all the sentences from the text. This can be as simple as just splitting the text at "." or newlines or more complex if you want to refine exactly what a sentence is. Parsers are never terminated, they are just abandoned.

Once you have all the sentences in the text you have to build a graph, in the graph every sentence is a node and you put links from each sentence to all others or to the k-most similar sentences weighted by similarity. You have to define a similarity score between two sentences, it can be something such as number of words in common over total number of words or a very complex formula, this is where you will spend most of your time to "tune" TextRank to work very well for whatever you are trying to summarize. Beware: Different similarity scoring functions will dramatically change the result of TextRank.

Once you have the weighted graph you have to run PageRank on it, here is a very simple approximation to PageRank.

Execute random walks over the graph, in each random walk you start from a random node, from that node based on the weights you select a random neighbor and so on. For example if you are in node "A" (remember each node represents a sentence) and your neighbors and weights are "B"(0.65) "C"(0.04) "D"(0.27) then you can compute the probability of going from A to each of those nodes as:

A=>B = 0.65/(0.65+0.04+0.27)

A=>C = 0.04/((0.65+0.04+0.27)

A=>D = 0.27/(0.65+0.04+0.27)

So you roll a random number from 0 to 1 and depending on the probability of each link you go to B, C or D. In this case you are very likely to visit B from A.

One problem with this approach is that you have to decide the length of each walk and the number of walks to execute. There's an elegant solution to this problem. Just do a single gigantic random walk, at each step with probability beta you visit a neighbor and with probability 1-beta you randomly jump to any random node, which is a way to simulate the start of a new walk. Beta is usually around 0.85.

As you execute your random walk add 1 to the score of a sentence every time you visit it.

At the end you can sort the sentences by the number of times you visited them and you have your sentences ranked. Just take the top-n sentences in the order as they appear in the text and you have a summary.

With this approach you have a way to program TextRank without doing any math and without using matrices, you just need your graph and a function to compute the similarity between sentences.

What TextRank does is very simple: it finds how similar each sentence is to all other sentences in the text. The most important sentence is the one that is most similar to all the others. With this in mind the similarity function should be oriented to the semantic of the sentence. Cosine similarity based on a bag of words approach can work well.

If you extract words instead of sentences and follow the same algorithm, using a similarity function between words, then you can use TextRank to extract keywords from the text. The idea is that the word that is most similar to all the other words is the most important one. Filtering stop-words is very important here.

### 4.2.3 Results

Parts from the results:

#### *a) For English wikipedia article about the 20th century:*

The period was marked by a new arms race as the USSR became the second nation to develop nuclear weapons, which were produced by both sides in sufficient numbers to end most human life on the planet had a large-scale nuclear exchange ever occurred. Western Europe was rebuilt with the aid of the American Marshall Plan, resulting in a major post-war economic boom, and many of the affected nations became close allies of the United States. The dissolution of the Soviet Union in 1991 after the collapse of its European alliance was heralded by the West as the end of communism, though by the century's end roughly one in six people on Earth lived under communist rule, mostly in China which was rapidly rising as an economic and geopolitical power.

#### *b) For Arabic wikipedia article about world war II*

في الأول من سبتمبر هاجمت القوات الألمانية بولندا بحجة أن بولندا شنت هجمات على الأراضي الألمانية[64]، وبعد يومين، في 3 سبتمبر 1939 ونتيجة لتجاهل ألمانيا الإنذار البريطاني بوقف العمليات العسكرية في بولندا . أعلنت كل من المملكة المتحدة وفرنسا الحرب على ألمانيا وتبعهما في إعلان الحرب دول الكومنولث البريطاني وهي كل من أستراليا (في 3 سبمتمر) ونيوزلندا (3 سبتمبر) وجنوب أفريقيا (6 سبمتبر) وكندا (10 سبتمبر)[13]، وبدأت بريطانيا بإرسال جيوشها إلى فرنسا، إلا أن هذه الجيوش لم تقدم أي مساعدة فعلية للبولنديين خلال الغزو، وبقيت الحدود الفرنسية الألمانية هادئة، وبدأت ما تعرف بالحرب الزائفة[65]، حيث اكتفت الجيوش بفرض حصار اقتصادي على ألمانيا مما دفع ألمانيا بمهاجمة السفن التجارية في المحيط الأطلسي وبداية معركة الأطلسي

## 4.3 Abstractive Approach Using Sequence To Sequence Model

We already discussed extractive approaches, now let's see an abstractive one. Here, we generate new sentences from the original text. This is in contrast to the extractive approach we saw earlier where we used only the sentences that were present. The sentences generated through abstractive summarization might not be present in the original text.

### 4.3.1 Dataset

I tried two different datasets in this problem:

#### a) Food Review data set:

This dataset consists of reviews of fine foods from Amazon. The data spans a period of more than 10 years, including all ~500,000 reviews up to October 2012. These reviews include

product and user information, ratings, plain text review, and summary. It also includes reviews from all other Amazon categories.

We'll take a sample of 100,000 reviews to reduce the training time of our model.
Data obtained from here: kaggle

### b) News Summary:

This one consists of news summary. The dataset has up to 98,000 news headlines (considered as our summaries) and texts.

The dataset can be acquire from here: kaggle

## 4.3.2 Data Preprocessing

Performing basic preprocessing steps is very important before we get to the model building part. Using messy and uncleaned text data is a potentially disastrous move. So, we will drop all the unwanted symbols, characters, etc. from the text that do not affect the objective of our problem. In short we apply the following to both text and summary (headlines in news case):

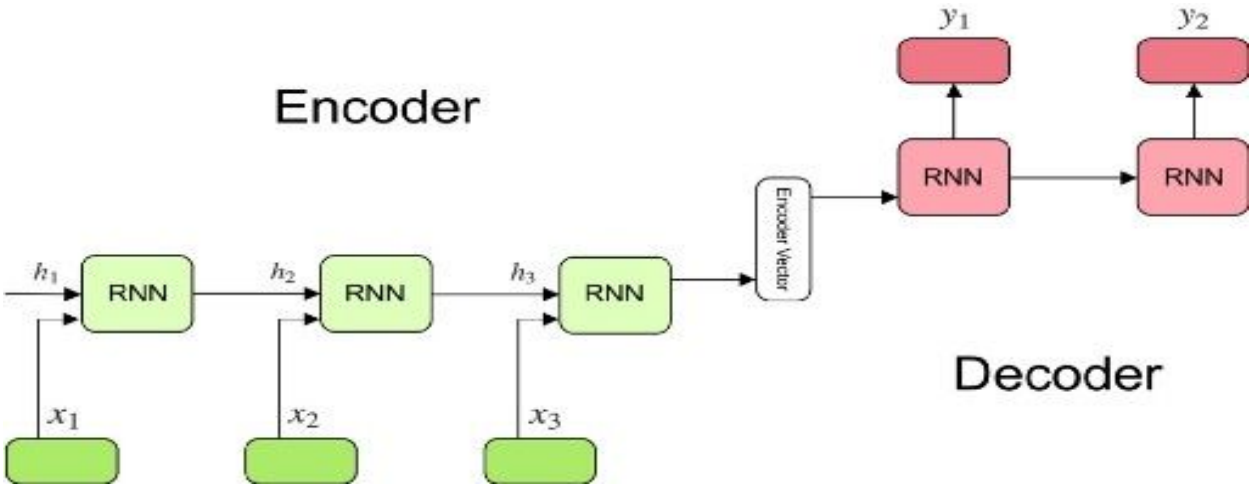
- Convert everything to lowercase
- Contraction mapping
- Eliminate punctuations and special characters
- Remove stopwords

## 4.3.3 How the Model Works

Abstractive Text Summarizer uses deep learning: it uses sequence to sequence models with an attention layer. So let's discuss everything briefly.

### 4.3.3.1 How the Sequence To Sequence Model Works?

In order to fully understand the model's underlying logic, we will go over the below illustration:



*Encoder-decoder sequence to sequence model*

The model consists of 3 parts: encoder, intermediate (encoder) vector and decoder.

## Encoder

- A stack of several recurrent units (LSTM or GRU cells for better performance) where each accepts a single element of the input sequence, collects information for that element and propagates it forward.
- In the question-answering problem, the input sequence is a collection of all words from the question. Each word is represented as *x_i* where *i* is the order of that word.
- The hidden states *h_i* are computed using the formula:

$$h_t = f(W^{(hh)} h_{t-1} + W^{(hx)} x_t)$$

This simple formula represents the result of an ordinary recurrent neural network. As you can see, we just apply the appropriate weights to the previous hidden state *h_(t-1)* and the input vector *x_t*.

## Encoder Vector

- This is the final hidden state produced from the encoder part of the model. It is calculated using the formula above.
- This vector aims to encapsulate the information for all input elements in order to help the decoder make accurate predictions.
- It acts as the initial hidden state of the decoder part of the model.

## Decoder

- A stack of several recurrent units where each predicts an output *y_t* at a time step *t*.
- Each recurrent unit accepts a hidden state from the previous unit and produces an output as well as its own hidden state.
- In the question-answering problem, the output sequence is a collection of all words from the answer. Each word is represented as *y_i* where *i* is the order of that word.
- Any hidden state *h_i* is computed using the formula:

$$h_t = f(W^{(hh)} h_{t-1})$$

As you can see, we are just using the previous hidden state to compute the next one.

- The output *y_t* at time step *t* is computed using the formula:

$$y_t = softmax(W^S h_t)$$

We calculate the outputs using the hidden state at the current time step together with the respective weight W(S). Softmax is used to create a probability vector which will help us determine the final output (e.g. word in the question-answering problem).

The power of this model lies in the fact that it can map sequences of different lengths to each other. As you can see the inputs and outputs are not correlated and their lengths can differ. This opens a whole new range of problems which can now be solved using such architecture.

#### 4.3.3.2 Inference Phase

After training, the model is tested on new source sequences for which the target sequence is unknown. So, we need to set up the inference architecture to decode a test sequence.

**How does the inference process work?**

Here are the steps to decode the test sequence:

1. Encode the entire input sequence and initialize the decoder with internal states of the encoder
2. Pass <**start**> token as an input to the decoder
3. Run the decoder for one timestep with the internal states
4. The output will be the probability for the next word. The word with the maximum probability will be selected
5. Pass the sampled word as an input to the decoder in the next time step and update the internal states with the current time step
6. Repeat steps 3 – 5 until we generate <**end**> token or hit the maximum length of the target sequence

#### 4.3.3.3 Limitations Of the Encoder – Decoder Architecture

As useful as this encoder-decoder architecture is, there are certain limitations that come with it. The encoder converts the entire input sequence into a fixed length vector and then the decoder predicts the output sequence. This works only for short sequences since the decoder is looking at the entire input sequence for the prediction. Here comes the problem with long sequences. It is difficult for the encoder to memorize long sequences into a fixed length vector

So how do we overcome this problem of long sequences? This is where the concept of attention mechanism comes into the picture. It aims to predict a word by looking at a few specific parts of the sequence only, rather than the entire sequence. It really is as awesome as it sounds!

#### 4.3.3.4 The Intuition Behind the Attention Mechanism

How much attention do we need to pay to every word in the input sequence for generating a word at timestep **t**? That's the key intuition behind this attention mechanism concept.

Let's consider a simple example to understand how attention mechanism works:

**Source sequence:** “Which sport do you like the most?
**Target sequence:** “I love cricket”

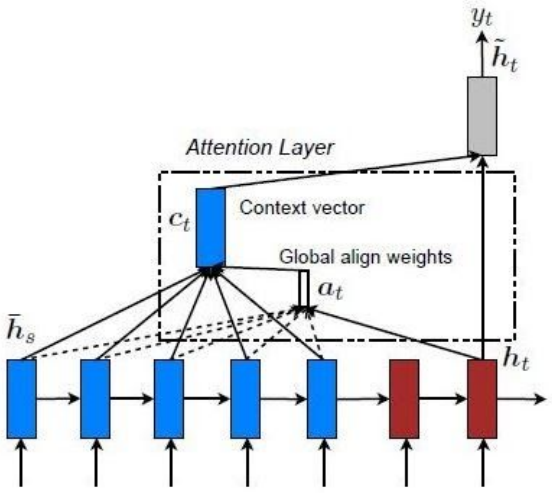
The first word **‘I’** in the target sequence is connected to the fourth word **‘you’** in the source sequence, right? Similarly, the second-word **‘love’** in the target sequence is associated with the fifth word **‘like’** in the source sequence.

**So, instead of looking at all the words in the source sequence, we can increase the importance of specific parts of the source sequence that result in the target sequence.** This is the basic idea behind the attention mechanism.

There are 2 different classes of attention mechanism depending on the way the attended context vector is derived:

## Global Attention

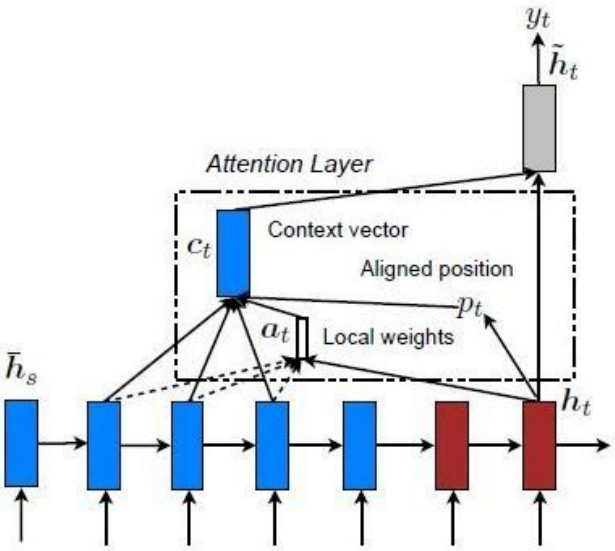
Here, the attention is placed on all the source positions. In other words, **all the hidden states of the encoder are considered for deriving the attended context vector:**




*Source: Effective Approaches to Attention-based Neural Machine Translation – 2015*

## Local Attention

Here, the attention is placed on only a few source positions. **Only a few hidden states of the encoder are considered for deriving the attended context vector:**




*Source: Effective Approaches to Attention-based Neural Machine Translation – 2015*

We will be using the Global Attention mechanism.

#### 4.3.3.5 Model Building

Our model consists of:

- 3 LSTM stacked for the encoder
- 1 LSTM for the decoder
- Attention layer (We used a custom made one downloaded from here)

**Loss Function:**

I am using sparse categorical cross-entropy as the loss function since it converts the integer sequence to a one-hot vector on the fly. This overcomes any memory issues.

## 4.3.4 Results

### a) Food Reviews

**Review:** chip flavor comes seeds n t think re salty flavor pretty well balanced thickness crunchiness comparable wheat thin one chip broken half last bag ate br br ingredients list pretty interesting n t long chemical names stone ground yellow corn high oleic sunflower oil brown rice flour flax seeds turbinado cane sugar oat fiber sesame seeds sunflower seeds quinoa soy flour sea salt take look gallery picture back bag br br bag calories contains grams fat make pretty good snack
**Original summary**: pretty tasty chips weird ingredients
**Predicted summary:** good

**Review:** three lb chocolate labs range age years years although give rawhide regularly teeth still funky went first bag chews teeth three dogs improved like much ordered another three bags br br regarding choking hazard dogs need chew need give things dogs attempt swallow pieces large gag dog owner must monitor dog s chewing even dogs gag usually learn first couple times n t gag still need monitor dog re chewing goes anything chew.
**Original summary:** really cleans teeth
**Predicted summary:** dogs love

**Review:** wife enjoyed grade syrup taste strong flavor started using diet recommendation re hooked great buy
**Original summary:** excellent taste organic
**Predicted summary:** yummy

### b) News Summary

**Review**: video shows ranveer singh rapping shared rapper divine instagram ranveer play rapper gully boy attending rapping workshop prepare role zoya akhtar directorial gully boy reportedly inspired lives rappers vivian fernandes also known divine naved shaikh also known naezy
**Original summary**: video shows actor ranveer singh rapping
**Predicted summary:** video shows ranveer singh dancing girl s son

**Review**: scammers made facebook instagram accounts name facebook ceo mark zuckerberg coo sheryl sandberg tricking individuals sending large amounts money order collect bogus lottery winnings many cases winners asked send hundreds dollars itunes gift card redemption codes delivery fees required collecting lottery funds.
**Original summary**: mark zuckerberg s name used scam people
**Predicted summary**: facebook ceo mark zuckerberg s personal personal data

# 5 Limitations And Improvements

## 5.1 Arabic Language

Unfortunately I was not able to find a text summarization Arabic dataset to make an abstractive approach. Machine learning and especially deep learning cannot improve or work its magic without data. The focus on deep learning increased dramatically after the improvements in data gathering in the past couple of years. So I think the Arab community should focus more on gathering data in the meantime.

Another solution would be to use language agnostic approaches like the LASER (Language-Agnostic SEntence Representations) project released by Facebook as it can work on 92 different languages.

## 5.2 Bigger Dataset

The use of a bigger dataset or even a better one of course might change everything completely like any deep learning problem. I found a Wikihow dataset that is really good but due to hardware and internet limitations I couldn't manage to try. The paper discussing the extraction of a dataset is available in the references.

# 6 References

AraVec:
https://github.com/bakrianoo/aravec/tree/11457255d7a6114da7ff5e16f3c936eeccaa672a
Glove: https://nlp.stanford.edu/projects/glove/
TextRank Paper: https://web.eecs.umich.edu/~mihalcea/papers/mihalcea.emnlp04.pdf
Attention Mechanism: https://towardsdatascience.com/attention-networks-c735befb5e9f
Custom Attention:https://github.com/thushv89/attention_keras/blob/master/layers/attention.py
WikiHow Paper: https://arxiv.org/pdf/1810.09305.pdf