Natural Language Processing

March 21, 2023

INSTITUTO TECNOLOGICO AUTONOMO DE MEXICO (ITAM)

Economic Research Seminar / Applied Research 2[1]

Professor: Horacio Larreguy

TA: Eduardo Zago

[1]We give credits to the Deep Learning course taught at ITAM's Master in Data Science because some of the content in this tutorial is taken from the notes made by PhD. Edgar Francisco Roman-Angel

1 Introduction to Natural Language Processing

1.1 What is NLP?

NLP stands for Natural Language Processing, which is a field of study focused on enabling computers to understand, interpret, and generate human language. NLP is at the intersection of computer science, Artificial Intelligence and Linguistics.

NLP has a wide range of applications, including language translation, sentiment analysis, speech recognition, chatbots, and text summarization, among others. Let us look at some introductory definitions of concepts very important to the understanding of NLP:

1. **Document** is the unit for NLP. If a dataset contains 100 news articles, then the dataset contains 100 documents.
2. **Corpus** is a collection of documents.
3. The **vocabulary** of a corpus is the collection of **unique** words or tokens.

1.2 Cleaning the Text

To start doing some analysis and, more importantly, before representing our texts as numbers, we must do a bit of cleaning as to improve our chances of getting an accurate mathematical representation.

To motivate this notice that the same word, but one written with an upper case letter at the beginning (as it could be at the beginning of the sentence) and one written at the end would be considered as two distinct words in our vocabulary, which would make our analysis intrinsically wrong, since its the same word. Let us look at an example in

Python:

```
import pandas as pd

# Define the string to work on:
string1 = 'Creo que el ITAM es la mejor universidad del país #ITAM.
           Aunque creo que no de todos los paises de latinoamerica @AMLO.'
```

We have the following string. Let's first tokenize this string and keep the unique words as to obtain the vocabulary of our Corpus (which contains only one document). **Tokenization** refers to the process of breaking down the raw text into small chunks called tokens, these could be words, sentences, etc. To do this we will introduce the package Natural Language Toolkit (`nltk`), which is the go-to for text analysis, since it contains easy-to use interfaces to over 50 corpora, and very simple functions for text pre-processing. We install it and load into the kernel:

```
!pip install nltk
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize
```

We obtain the unique tokens of our sentence:

```
tokens = word_tokenize(string1)
unique_tokens = []
[unique_tokens.append(x) for x in tokens if x not in unique_tokens]
unique_tokens

## Output:
#['Creo', 'que', 'el', 'ITAM', 'es', 'la', 'mejor', 'universidad', 'de',
#'todas', 'las', 'del', 'país', '#', '.', 'Aunque', 'creo', 'no', 'todos', 'los',
#'paises', 'latinoamerica', '@']
```

Notice a few things. The word *creo*, is repeated twice, as the first appearance is at the beginning with an upper case later and the second one is lower cased. The first task is easy, we first turn all words to lower case letters and remove accents:

```
# Defining the function to remove accents:
```

```
import unidecode
def remove_accents(a):
    return unidecode.unidecode(a)

# Applying it to the string:
string2 = string1.lower()
string2 = remove_accents(string2)
string2

### Output:
# 'creo que el itam es la mejor universidad de todas las del pais
# #itam. aunque creo que no de todos los paises de latinoamerica @amlo.'
```

Also, we use **regular expressions** to clean any character we do not want our model to use such as # or a point. **Regular expressions** is a formal language to specify text strings. Most of this expressions that we would like to match are already in Google so it is not necessary to actually learn the language. But, for example, if we wanted to match the character and every word to the right we would use the next regular expression:

```
# Remove URLs
import re # Import regular expression package
def remove_url(text):
    return re.sub(r'https?:\S*','',text)
string2 = remove_url(string2)

# Remove mentions and hashtags
def remove_mentions_and_tags(text):
    text = re.sub(r'@\S*','',text)
    return re.sub(r'#\S*','',text)
string2 = remove_mentions_and_tags(string2)
string2

## Output:
# 'creo que el itam es la mejor universidad de todas las del pais
# aunque creo que no de todos los paises de latinoamerica '
```

Final thing we can do is **removing stopwords**. **Stopwords** are a collection of words that are commonly used in a language but do not provide much meaning to the text. Some examples in Spanish are *a*, *tu*, *y*, etc. So it is common practice to remove them

in contexts where they are not needed[2].

```
from nltk.corpus import stopwords
stop_words = set(stopwords.words('spanish'))

word_tokens = word_tokenize(string2)
# converts the words in word_tokens to lower case and then checks whether
#they are present in stop_words or not
filtered_sentence = [w for w in word_tokens if not w.lower() in stop_words]
#with no lower case conversion
filtered_sentence = []

for w in word_tokens:
    if w not in stop_words:
        filtered_sentence.append(w)

print(filtered_sentence)

### Output:
# ['creo', 'itam', 'mejor', 'universidad', 'todas', 'pais', '#', 'itam', '.',
# 'aunque', 'creo', 'paises', 'latinoamerica', '@', 'amlo', '.']
```

2 Unsupervised Learning Techniques: Topic Modeling

We are going to take a look at some unsupervised learning techniques that can be useful to reduce the dimensionality of a Corpus with a big number of sentences: Topic Modeling. To do so we will use a model called LDA (Latent Dirichlet Allocation).

Topic modeling refers to an unsupervised method of classification that enables us to find natural groups of words that belong to a certain topic. The basic assumption for LDA is that each of the documents can be represented by the distribution of topics,

[2]Some models, like BERT, actually can benefit from stop-words as they help contextualize words in sentences.

which in turn can be represented by some word distribution.

The main issue with this algorithm is that we have to choose the number of topics for which we want to fit the LDA, and after obtaining the results, evaluate which model gives us the most human interpretable model. We will see that there will be groups that clearly contain information about a particular topic (e.g. the economy, politics, health, etc), and others that are not related at all. Let's look at an example using a data set containing posts labeled as fake by Africa Check.

2.1 Topic Modeling using LDA in Python

We will introduce in this section two packages that are extremely useful for NLP tasks: `SpaCy` and `gensim`. Let's install them and download the Corpora that we will use:

```
!pip install pyLDAvis -qq
!pip install -qq -U gensim
!pip install spacy -qq

# Load the corpora:
!python -m spacy download en_core_web_md -qq
```

We will also be using the specific for this task package named `pyLDAvis`. Let's load the necessary packages into the kernel:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
import spacy
import pyLDAvis.gensim_models
pyLDAvis.enable_notebook()# Visualise inside a notebook
import en_core_web_md
from gensim.corpora.dictionary import Dictionary
from gensim.models import LdaMulticore
from gensim.models import CoherenceModel
```

As I said, let us use the data set of fake posts labeled by Africa Check:

```
fake = pd.read_parquet("../data/training_fake_final.parquet")
fake.head()
```

Figure 1: Fake Data Set

	text	label
0	Jacob Zuma and wife admitted to hospital after...	False
1	Apparently Bisi Olatilo and Bolu Akin-Olugbade...	False
2	@lyne_ian @Jeremy05749458 3 hours ago Pope Fra...	False
3	"We commend President Buhari for the swiftness...	False
4	So the hippo that was in Fourways has been sla...	False

Let us clean our text for analysis, in this example we will use SpaCy models to remove stopwords, other special characters as punctuation (PUNCT) and **lemmatize** the text. The objective of **lemmatization** is to reduce words to their root form, which is called **lemma**. For example, the word *eating* would be identified as *eat*.

```
# Our spaCy model:
nlp = en_core_web_md.load()

# Tags I want to remove from the text
removal= ['ADV','PRON','CCONJ','PUNCT','PART','DET','ADP','SPACE', 'NUM', 'SYM']
tokens = []

for summary in nlp.pipe(fake['text']):
    proj_tok = [token.lemma_.lower() for token in summary if token.pos_
    not in removal and not token.is_stop and token.is_alpha]
    tokens.append(proj_tok)

# Add tokens to new column
fake['tokens'] = tokens
fake['tokens']
```

```
#### Output:
#0 [jacob, zuma, wife, admit, hospital, contract]
```

We turn our tokens to a **dictionary** or **vocabulary** of unique tokens:

```
dictionary = Dictionary(fake['tokens'])
print(dictionary.token2id)

#### Output:
#{'admit': 0, 'contract': 1, 'hospital': 2, 'jacob': 3, 'wife': 4,...}
```

It is common practice to filter tokens that do not appear very often in the document. To do this we can filter the dictionary using the function `filter\_extremes`. We also create our Corpus.

```
# Filter dictionary
dictionary.filter_extremes(no_below=5, no_above=0.5, keep_n=2000)

# Create corpus
corpus = [dictionary.doc2bow(doc) for doc in fake['tokens']]
```

And we can start estimating our models:

```
lda_model5 = LdaMulticore(corpus=corpus, id2word=dictionary, iterations=100,
    num_topics=5,
workers = 4, passes=100)
```

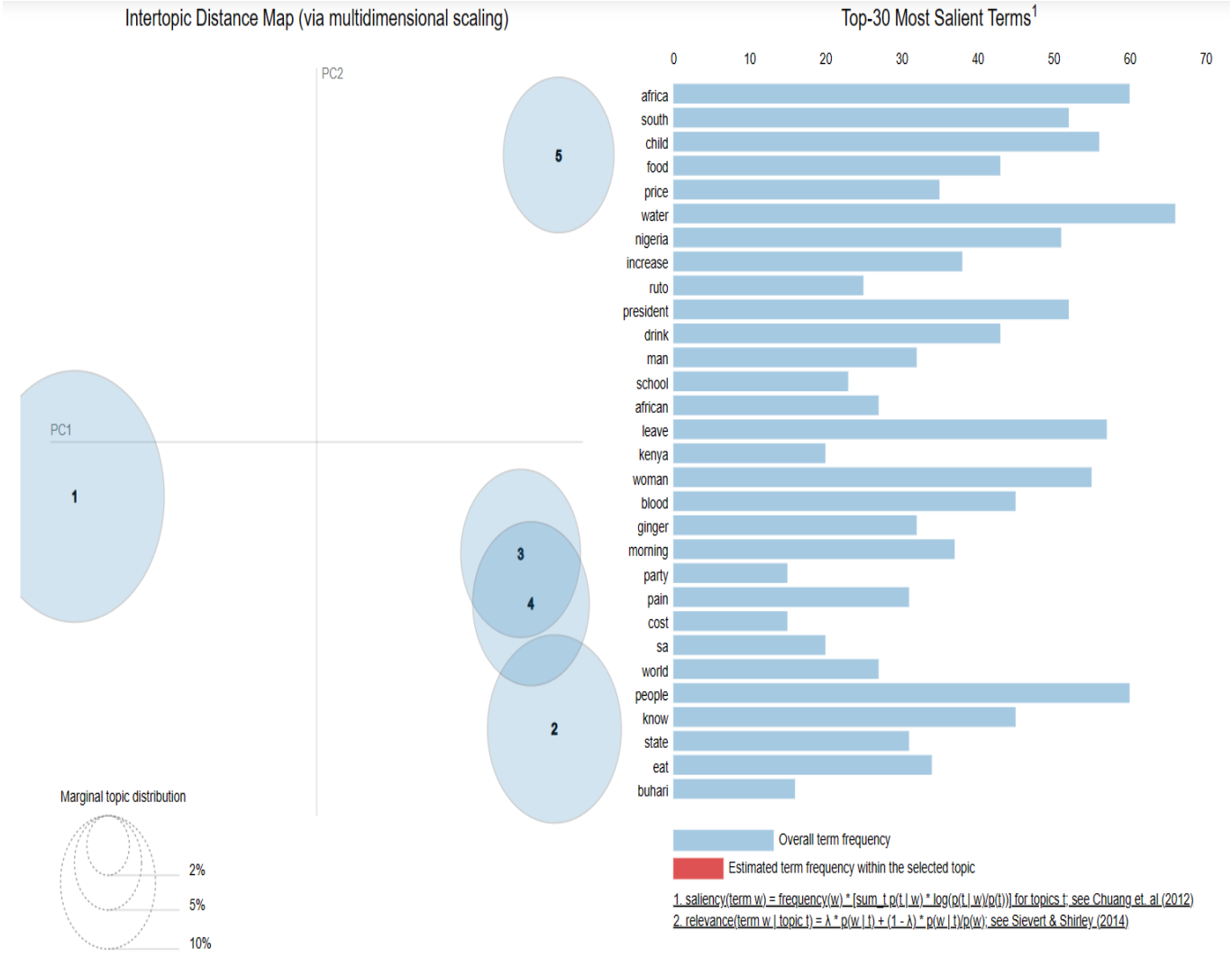
We need to visualize our results to understand if it is a good fit. Remember we need to keep the model with the most human interpretable results[3].

```
lda_display = pyLDAvis.gensim_models.prepare(lda_model5, corpus, dictionary)
pyLDAvis.display(lda_display)
```

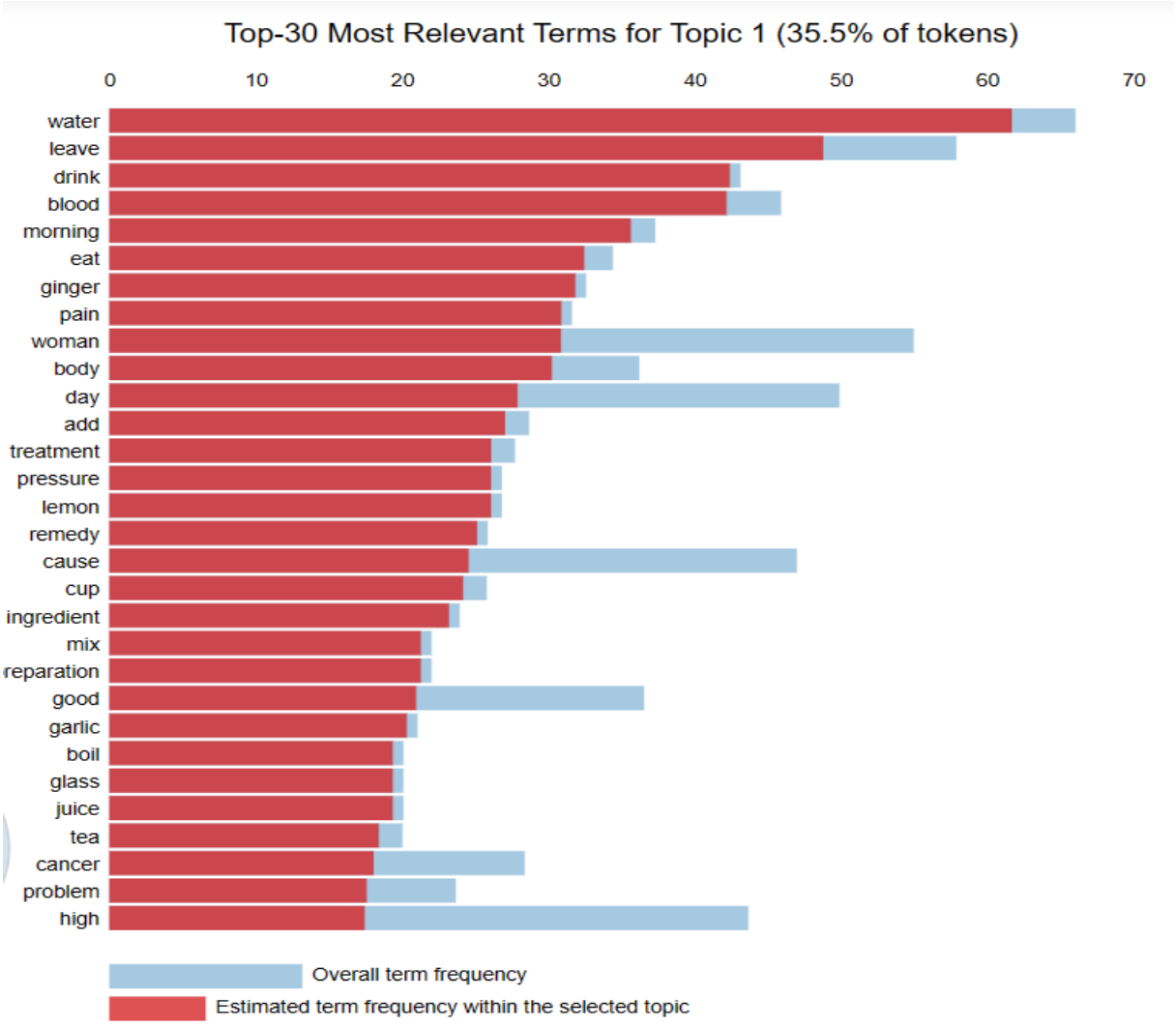
We first get the **Intertopic Distance** between groups, which basically provides us information on how different each group is by looking at the number of similar words in them. For this particular model we observe that Topic 1, 2 and 5 really capture different aspects of the text, while 3 and 4 overlap a lot.

[3]For better visualization please go to the notebook.

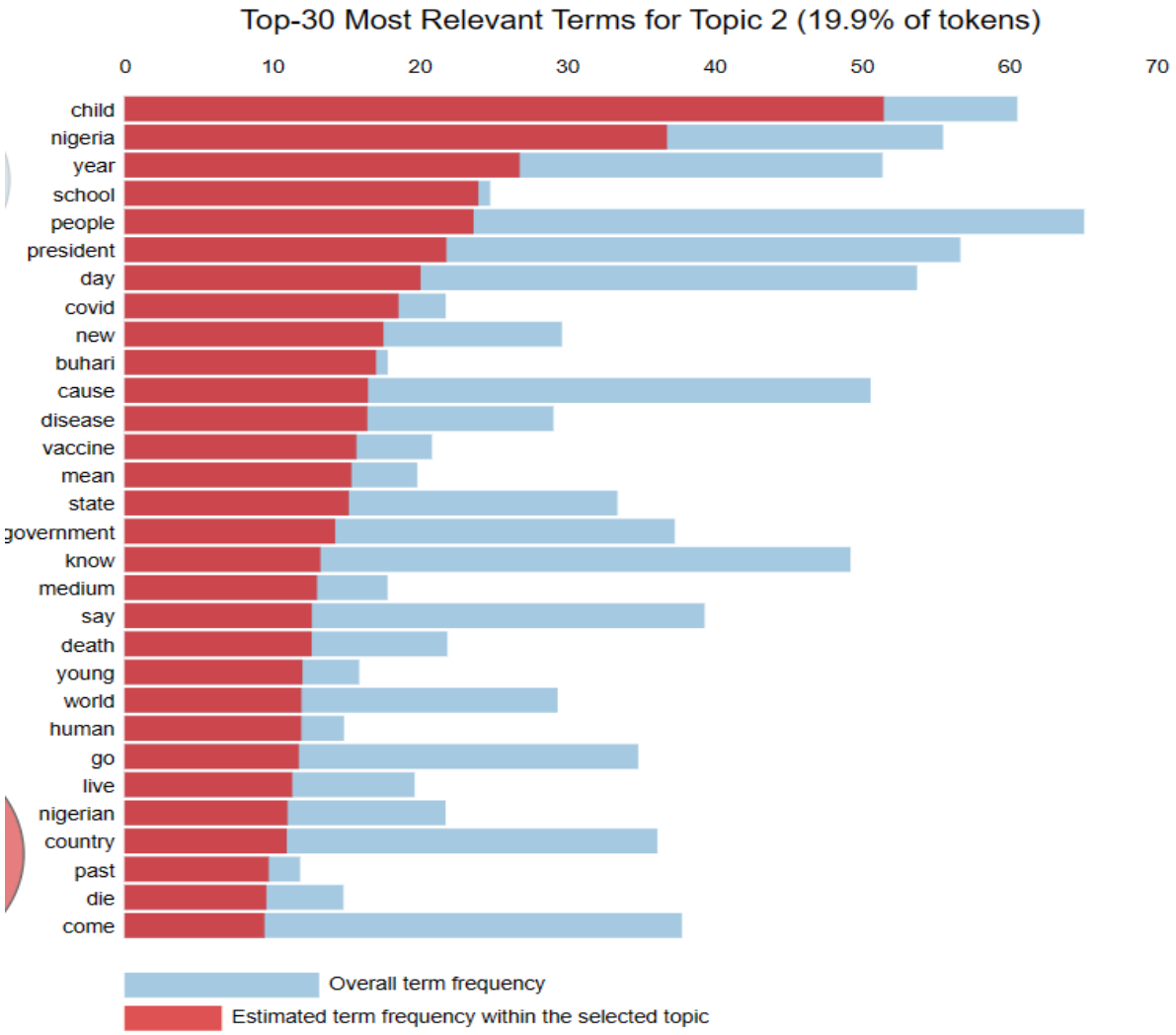
Figure 2: LDA Metrics, Intertopic Distance



And we can also visualize the most relevant terms for each group, this can help us assign a human interpretable topic to them. Let us visualize Topic 1 and 2:



We notice there are a lot of medical words, but also spices, food and other supplements used in Traditional Medicine. By looking manually at some of these texts, we concluded that these topic contained all posts related to Health Misinformation, using Traditional Medicine. So in this Topic there were tons of posts of people curing cancer using herbs, reducing blood pressure using tea, and so on.

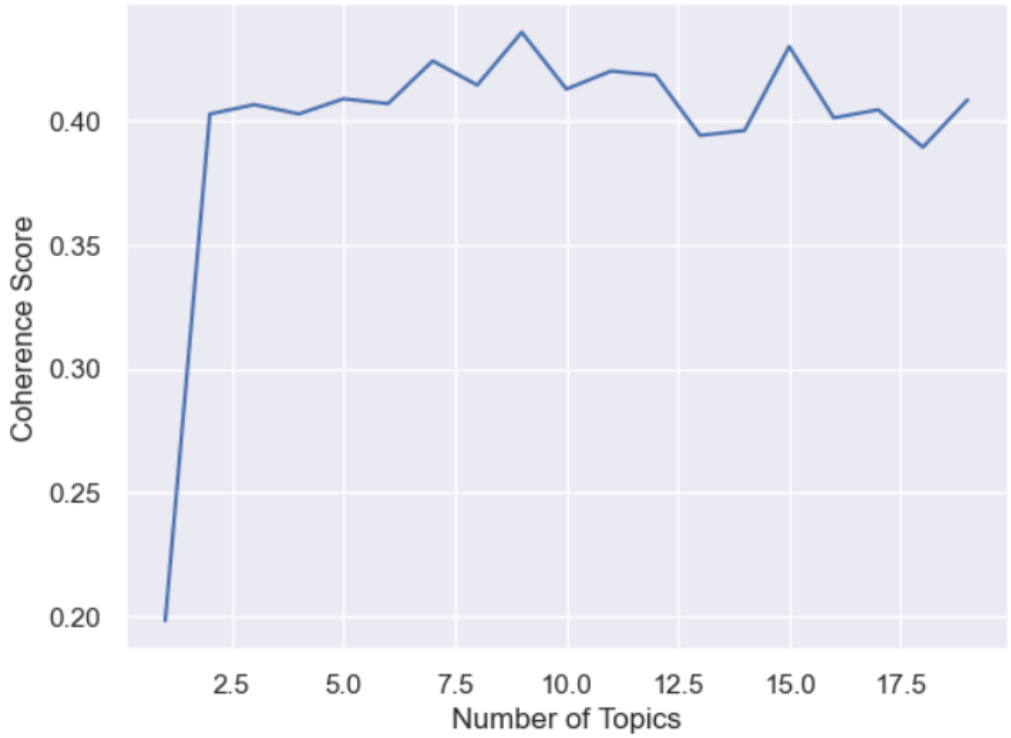


Topic 2 is also really interesting, since it is also about Health Misinformation, but as we will see, regarding Modern Medicine. Notice we have words such as vaccine, COVID, government and disease. So this topic contains posts related with COVID, vaccine and government Fake News.

As we have seen, since we are dealing with an Unsupervised Model, we do not have an objective metric such as accuracy to evaluate the goodness of the fit. Most of our results are judged subjectively and depend mostly in how well we can or cannot interpret them. However, there is a metric used to evaluate these types of models

called **Coherence**. The **Coherence** scores a single topic by measuring the degree of semantic similarity between high scoring words in the topic. These measurements help distinguish between topics that are semantically interpretable topics and topics that are artifacts of statistical inference, as stated in Towards Data Science. We are looking for the model with the best Coherence metric, thus we will evaluate several models with different number of topics to see which one is the best fit:

```
# Coherence score using C_v:
topics = []
score = []
for i in range(1,20,1):
    lda_model = LdaMulticore(corpus=corpus, id2word=dictionary, iterations=10,
        num_topics=i, workers = 4, passes=10, random_state=100)
    cm = CoherenceModel(model=lda_model, texts = fake['tokens'], corpus=corpus,
        dictionary=dictionary, coherence='c_v')
    topics.append(i)
    score.append(cm.get_coherence())
_=plt.plot(topics, score)
_=plt.xlabel('Number of Topics')
_=plt.ylabel('Coherence Score')
plt.show()
```



The model with the highest Coherence Score is the one with 9 topics. If we decided which model to use by this metric alone, we would have to choose the 9-Topic Model. However, it is important to note that **optimizing for coherence may not yield human interpretable topics**. By looking at the Notebook we can see that most topics are overlapping and some are not humanly distinguishable from one another, thus this model is not a good option for our analysis.

3 Sentiment Analysis in Spanish Using PiPy

While working with Neural Networks, we more often than not, will not have enough data points to train a complicated model such as BERT from scratch. However, we can use trained models (trained for our specific task) and apply them to our own small data set. In this chapter we will see how to use a `keras`' model for Sentiment Analysis in Spanish, to predict the sentiment of posts.

`sentiment-spanish` is a python library that uses convolutional neural networks to predict the sentiment of spanish sentences. The model was trained using over 800000 reviews of users of the pages eltenedor, decathlon, tripadvisor, filmaffinity and ebay. The model returns a number between 0 and 1, the closer to 0 the more negative the text is, and viceversa. For more information visit the package documentation.

We first load the necessary packages:

```
import pandas as pd
import numpy as np
import spacy
import nltk

from nltk.corpus import stopwords
from unicodedata import normalize

import re
import scipy as sc

from nltk.tokenize import word_tokenize, sent_tokenize
from keras.preprocessing.text import text_to_word_sequence
from sentiment_analysis_spanish import sentiment_analysis
from spacy.lang.es import Spanish

# Load the models:
# For lemmatization in Spanish
nlp = spacy.load('es_core_news_lg')}
# For sentiment analysis
from sentiment_analysis_spanish import sentiment_analysis
```

We load data that we will analyze and clean the text using functions defined in the Notebook (too long for this document, but all the code is defined in the Notebook for you to check and use). We will use 20 Tweets containing the word *feminazi* that we scrapped to analyze the prevalence of hate speech in Social Media during the 8M Marches.

```
# Load the data:
df = pd.read_parquet('../data/hate_speech.parquet')
```

```
# Clean it:
df = clean_data(df, 'text')
df = df[~((df['text_clean'].isna()) | (df['text_clean']==''))].reset_index(drop=
    True)
df[['author_id', 'text','text_clean']].head()
```

	author_id	text	text_clean
0	203579995	RT @Lady_Chiyome: Femenina, nunca #FEMINAZI 🤦\...	rt femenina nunca
1	1358301364384890880	@PabloEchenique @IreneMontero Prefiero escucha...	prefiero escuchar personas con mas neuronas qu...
2	232758195	RT @danielalozanocu: Todo el año: feminazi, lo...	rt todo el an feminazi loca abortera deberia m...
3	1325368543614013440	Feminismo≠Feminazi\nFeminismo es igualdad\nUn ...	feminismo feminazi feminismo es igualdad un ho...
4	551967420	RT @jmrivas6911: RADFEM\n\nHay cavada una trin...	rt radfem hay cavada una trinchera entre el od...

We define the functions that we will use to compute the sentiment and instantiate the model:

```
# Function to calculate the sentiment of a text
def sentiment_metrics(text, sentiments, sentiment_score=True):
    try:
        if sentiment_score:
            sentimiento = sentiments.sentiment(text)
        else:
            sentimiento = np.nan
    except:
        sentimiento = np.nan, np.nan
    return sentimiento

# Function to work with Data Frames:
def compute_sentiment(df, text):
    # Instantiate the model:
    sentiments = sentiment_analysis.SentimentAnalysisSpanish()
    df["sentiment_score"] = df.apply(
        lambda x: sentiment_metrics(x[text], sentiments),
        axis=1,
        )
    df_sentiment = pd.DataFrame(df["sentiment_score"].values.tolist(),
    index=df.index)
    df_sentiment.rename(columns={0: "sentiment_score"}, inplace=True)
    df_sentiment = pd.concat(
        [df, df_sentiment["sentiment_score"]], axis=1)
```

```
    return(df_sentiment)
```

Finally, we can do the prediction in our posts:

```
df = compute_sentiment(df, 'text_clean')
df[['author_id', 'text','text_clean', 'sentiment_score']].head()
```

Figure 3: Sentiment Prediction

	author_id	text	text_clean	sentiment_score
0	203579995	RT @Lady_Chiyome: Femenina, nunca #FEMINAZI 🤬\...	rt femenina nunca	0.152140
1	1358301364384890880	@PabloEchenique @IreneMontero Prefiero escucha...	prefiero escuchar personas con mas neuronas qu...	0.006189
2	232758195	RT @danielalozanocu: Todo el año: feminazi, lo...	rt todo el an feminazi loca abortera deberia m...	0.069884
3	1325368543614013440	Feminismo≠Feminazi\nFeminismo es igualdad\nUn ...	feminismo feminazi feminismo es igualdad un ho...	0.000006
4	551967420	RT @jmrivas6911: RADFEM\n\nHay cavada una trin...	rt radfem hay cavada una trinchera entre el od...	0.004116

Notice these are all negative Tweets, as they contain a word related with Hate Speech, and all of the predictions are close to 0. The model seems to be well trained and could be use for bigger data sets.

4 Mathematical Representations of Language

By now you are probably already wondering, how are we going to feed text into a Neural Net that does Mathematical Computations? Recall that language is a complex and dynamic aspect of human communication which involves the use of symbols, sounds, and grammar rules to convey meaning. Representing language mathematically is an incredibly hard task, but advancements have been made.

We will explore some of the key mathematical representations of language, including vector space models such as **One-Hot Encoding**, probabilistic models such as **n-grams** and finally, neural language models such as **Word Embeddings**.

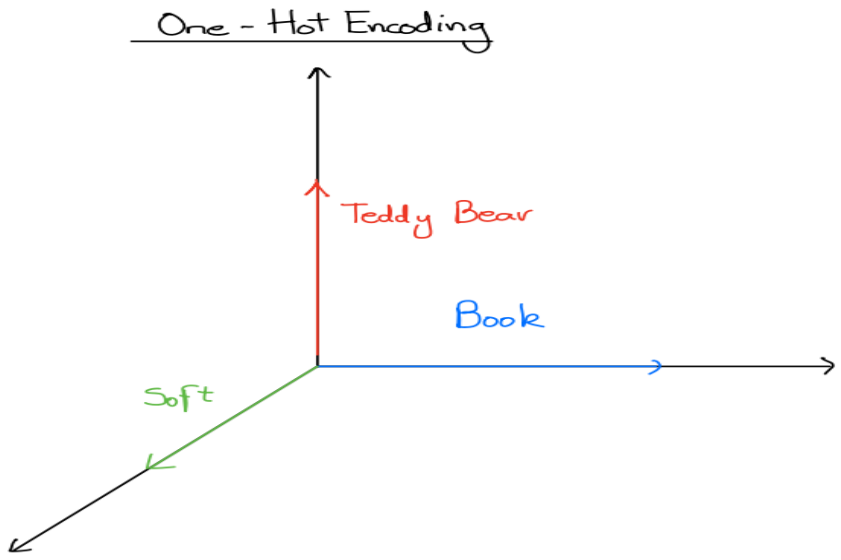
4.1 One-Hot Encoding Vectors

One-Hot Encoding is a method of representing language where each word or character is assigned a unique integer index, and then a vector is created for each word or character, with a dimension equal to the total number of unique words or characters in the dataset. We are representing every word as an $\mathbb{R} : |V| \times 1$ vector with all 0s and one 1 at the index of that word in the sorted English language. Thus, if we have a corpus consisting of Bear, Teddy, Soft and Book we would have 4 vectors such that:

$$x^{Bear} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, x^{Book} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, x^{Soft} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, x^{Teddy} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Notice that in $\mathbb{R}^3$ we would have the following representation of the three words:

Figure 4: One-Hot Encoded Vectors



Also notice that:

$$(x^{Soft})^T x^{Bear} = (x^{Bear})^T x^{Book} = 0 \tag{1}$$

Two things that are bad here:

1. Languages have millions of words. This is a practical nuisance since we would need vectors of millions of entries, and matrices of millions of vectors, to correctly define a whole Corpus.

2. When using One-Hot Vectors we are unable to extract relationships in the meaning of words. As seen in Equation 1, these vectors are **orthogonal** to each other.

4.2 N-grams

To perform a good analysis of texts we must capture some relationship between the words, to understand the context. As previously seen, One-Hot Encoding does not capture any similarity information over the words, thus most of the time the context is lost. One initial way to fix this is looking at the co-occurrence of words in our corpus.

Suppose we have a set of words $[w^{(1)}, w^{(2)}, ..., w^{(n)}]$, we would probably be interested in the probability of this words occurring together at any given sentence:

$$P(w^{(1)}, w^{(2)}, ..., w^{(n)})$$

So sentences that make sense, should have a high probability, in contrast to words just sequenced together with no particular meaning. We are trying to estimate the probability of these words co-occurring in the same sentence. How would we do this? We could start by taking the naive approach of assuming each word's occurrences are completely independent from each other:

$$P(w^{(1)}, w^{(2)}, ..., w^{(n)}) = \Pi_{i=1}^{n} P(w^{(i)})$$

We know this is probably wrong since the next word is highly contingent upon the previous sequence of words. Also, it might be that a nonsense sentence scores highly.

So perhaps we let the probability of the sequence depend on the last 2 words in the sequence:

$$P(w^{(1)}, w^{(2)}, ..., w^{(n)}) = \Pi_{i=2}^{n} P(w^{(i)}|w^{i-1})$$

This is a Bi-gram model. If we only cared for the last word it would be a uni-gram, and so on. Again this is certainly a bit naive since we are only concerning ourselves with pairs of neighboring words rather than evaluating a whole sentence. However, these models go very far in their way of representing language and are really useful for some tasks. How to estimate this probabilities goes outside the scope of this course, but the go-to models to do this would be CBOW (Continuous Bag of Words) and Skip-Gram[4].

4.3 Word Embeddings

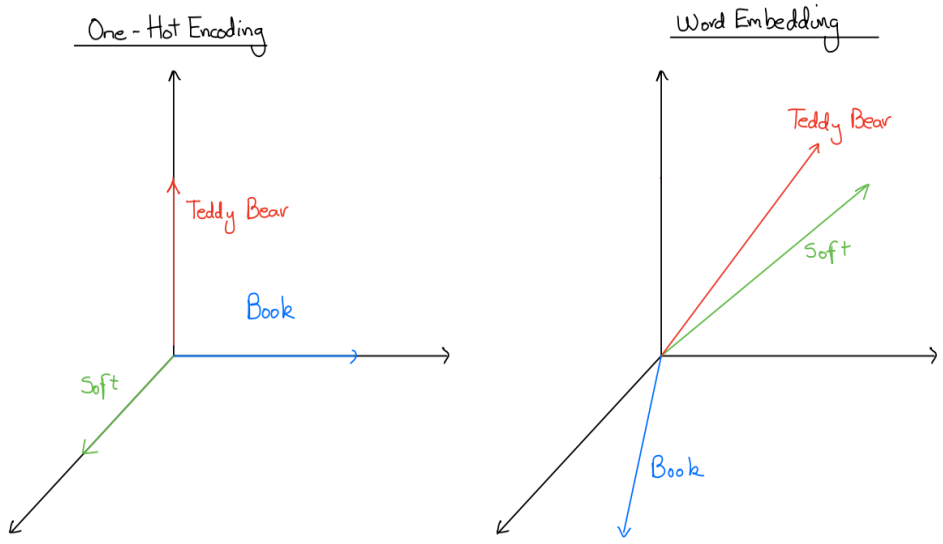
Word embeddings are numeric representations (vectors) of words, in which words with similar meaning result in similar representations. They are far superior to One-Hot Encoder Vectors since they take into account words similarity.
Recall the vectorial representation of Teddy, Bear, Soft and Book, for this case we should obtain something like this:

$$x^{Bear} = \begin{bmatrix} 2.3 \\ 3.4 \\ 0.4 \\ 5.2 \end{bmatrix}, x^{Book} = \begin{bmatrix} -2 \\ -0.05 \\ -.9 \\ -3 \end{bmatrix}, x^{Soft} = \begin{bmatrix} .5 \\ 3 \\ 4.12 \\ 1.2 \end{bmatrix}, x^{Teddy} = \begin{bmatrix} 2 \\ 3.3 \\ .65 \\ 0.03 \end{bmatrix}$$

Graphically, we would obtain the following:

[4]For more information regarding this topic you can look at the following Stanford Notes

Figure 5: One-Hot vs Word Embeddings

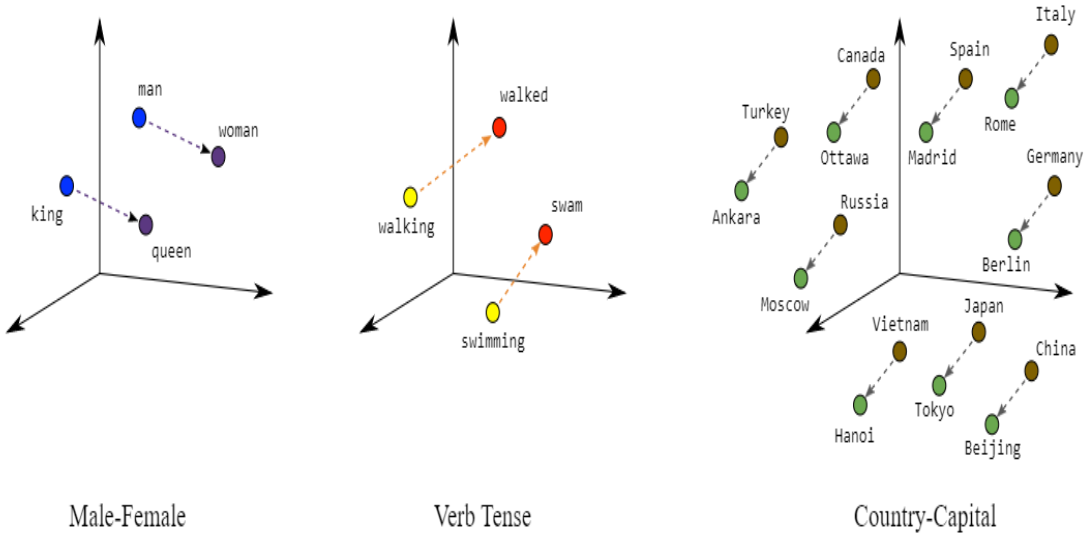


Notice that words that often go together, like Teddy Bear and Soft, are actually related geometrically, while Book, which is not a soft object, does not.

Embeddings are actually amazingly great at capturing semantic relations. We can see this by observing the following geometrical relationships between, for example, words like a Country and its Capital[5]:

[5]Image taken from Developers Google.

Figure 6: Geometrical Relationships



5 Transformer Revolution: BERT

The BERT model was pre-trained in a self-supervised fashion, meaning no human labeling of the data was involved. This model was initially built for Masked Language Modelling and Next sentence prediction tasks and was pre-trained on BookCorpus, a dataset consisting of 11,038 unpublished books and English Wikipedia. In contrast to deep learning neural networks which require very large amounts of data, BERT has already been pre-trained which means this model has already learnt the inner structure of the English language and its features can be used to train a standard classifier. For our purposes, we can Fine-Tune the bert-base-uncased for prediction, by adding and training (with our own dataset) a classifier layer on top of the BERT architecture.

However, before jumping into the model and how to use it in Python, let's understand What makes BERT so powerful and special. Two main things:

1. The way it uses and estimates two different word embeddings.

2. The Attention Mechanism.

5.1 Word Embeddings

5.1.1 Token Embeddings

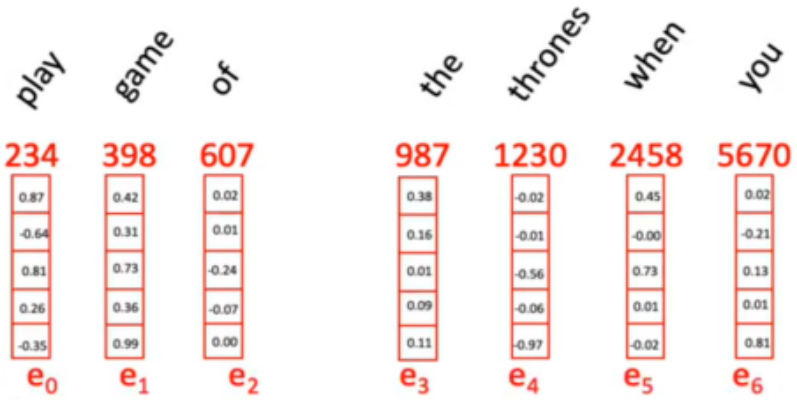
BERT uses two different types of embeddings for its tasks: The first ones are **Token embeddings**, these are the standard word embeddings that represent each token (word or subword) in the input text. This is the first layer of the NN, and it is named the Embeddings Layer.

The Embedding layer maps integers into One-Hot-Encoded Vectors with length equal to the whole Corpus. Then, this vectors are mapped into a dense vector representation (word embeddings). The author in the model uses vectors of size 512. These vectors are weights learned during training and are similar across words that appear in the same contexts. Again, since BERT was already pre-trained in a big English Data Set, it has already learnt the structure of the words and sentences from the English Language. Thus these embedding can be used in other models to represent numerically an entirely different data set.

Figure 13 provides an example of how this embeddings are seen[6].

[6]Image taken from the Deep Learning course, from ITAM's Master in Data Science Program.

Figure 7: Token Embeddings

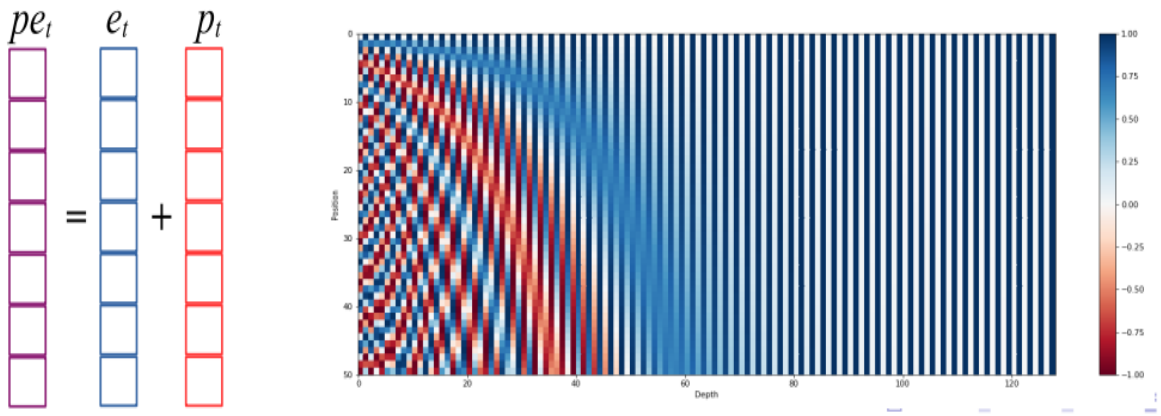


5.1.2 Position Embeddings

These embeddings are used to encode the positional information of each token in the input text. Since BERT is a pre-trained language model that takes variable-length input sequences, it needs to understand the position of each token in the input. This is a problem since transformers process all embeddings at once, in parallel. This layer is used to compensate for the lack of recurrence operations.

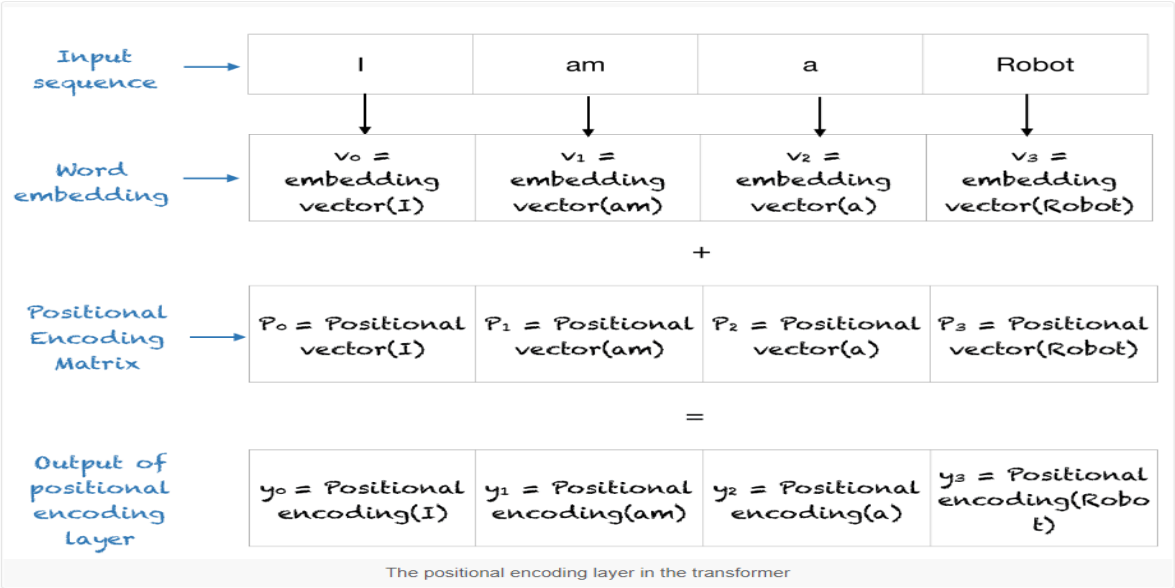
Position information is used through wave frequencies, using a cosine functions. The farther apart from the beginning, the smaller the amplitude of the wave. These then are ranked by amplitude of waves to provide positional information to the Word Embeddings. These are also trainable weights.

Figure 8: Positional Embeddings



Position embeddings are added to the token embeddings to provide the model with this positional information. The position embeddings are learned during the pre-training stage and are added to the token embeddings before feeding them into the transformer layers[7].

Figure 9: Final Embeddings



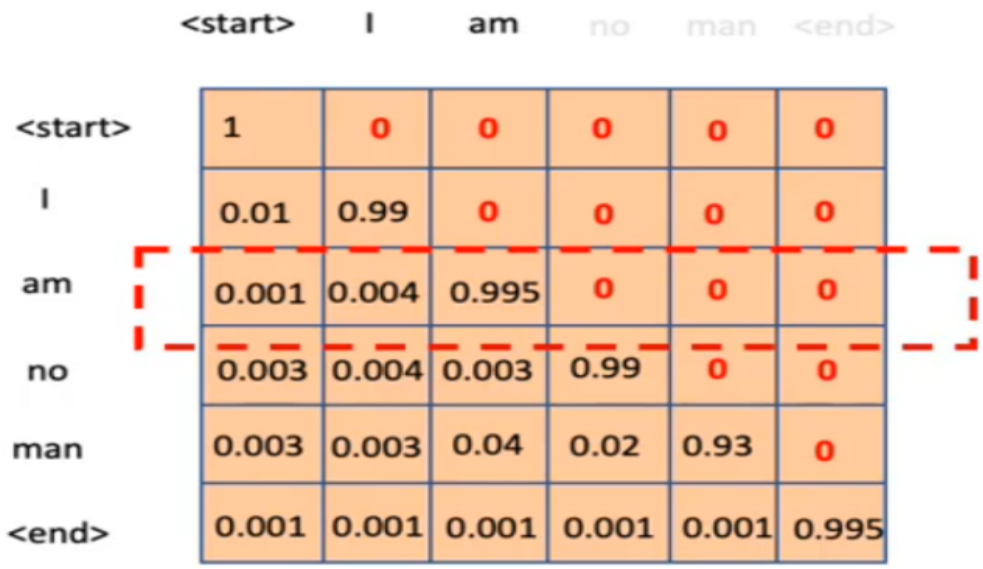
[7]Image taken from Machine Learning Mastery

5.2 Attention Mechanism

BERT uses a self-attention mechanism to understand the context of each word in a sentence. Self-attention allows BERT to **weigh the importance of each word in a sentence** based on the context of the entire sentence[8]

The attention mechanism in the simplest of terms, involves generating a score for each word. The score indicates how relevant the word is to the current word being processed. Before attention, models only could see at the last word (last hidden state), therefore losing tons of context. With attention, BERT i able to access all past hidden states, and is also able to assign to them a score of relevance with respect to the word being processed.

Figure 10: Attention Mechanism



By using self-attention, BERT can capture complex relationships between words in a

[8]For more information of this, look at the following paper Vaswani et al. (2017) and/or the following article.

sentence and generate high-quality representations of them. It also solves the problem of models not being able to contextualize words properly.

6 Fine-Tuning BERT For Classification Tasks

In this section we will take a look at how to Fine-Tune BERT to solve a Classification problem: Is a post verifiable or not? Let's first install all necessary packages[9]:

```
!pip install -q tensorflow-text  # A dependency of the preprocessing for BERT
    inputs
!pip install -q tf-models-official # For the AdamW optimizer from tensorflow/
    models
!pip install bert-for-tf2
!pip install sentencepiece
```

And loading all necessary packages into the Kernel:

```
import os
import shutil
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_text as text
import re
from official.nlp import optimization  # to create AdamW optmizer
tf.get_logger().setLevel('ERROR')

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Embedding, Dense, Dropout

# Data cleaning
from drive.MyDrive.bert_train.src.utils.clean import *
import nltk
nltk.download('stopwords')
```

[9]The Notebook in the DropBox Folder, as well as this code, is meant to be run in a Google Colab Notebook.

We import the data and define the functions to clean the text:

```
# Importing the data:
df = pd.read_parquet('drive/MyDrive/bert_train/6-labeled/2-training_2_classes.
    parquet')

# Defining the functions to clean the text
TAG_RE = re.compile(r'<[^>]+>')
def preprocess_text(sen):
    sentence = TAG_RE.sub('', sen) # html tags
    sentence = re.sub('[^a-zA-Z]', ' ', sentence) # punctuations and numbers
    sentence = re.sub(r"\s+[a-zA-Z]\s+", ' ', sentence) # single character
    sentence = re.sub(r'\s+', ' ', sentence) # multiple spaces
return sentence.lower()
```

Figure 11: Verifiable Data Set

	text	URL	label
0	Controversial cross dresser Idris Okuneye aka ...	https://www.facebook.com/161915460542267/posts...	Verifiable
1	Tinubu's presidency will produce wealth, prosp...	https://www.facebook.com/161915460542267/posts...	Not Verifiable
2	PDP was corrupt in 2015 and Nigerians needed t...	https://www.facebook.com/100044230039479/posts...	Not Verifiable
3	Underwater energy is WoW	https://twitter.com/MrOdanz/status/15447596547...	Not Verifiable
4	Mbappe's ego is getting way too annoying. Man ...	https://twitter.com/MrOdanz/status/15591191764...	Not Verifiable
...	...	...	...
1072	With 523 points, Karim Benzema wins UEFA POTY ...	https://twitter.com/MrOdanz/status/15628684937...	Verifiable
1073	My mum never changed her surname after marriag...	https://twitter.com/MrOdanz/status/15619785734...	Not Verifiable
1074	Heroic welcome for Raila as he votes in Kibera	https://www.facebook.com/178342827608/posts/59...	Verifiable
1075	Take this Homemade drink for 5days and say Goo...	https://www.facebook.com/peterfatomilolaoffici...	Verifiable
1076	#KenyaDecides Results Update: Raila: 129,751 -...	https://www.facebook.com/100044230039479/posts...	Verifiable

1077 rows × 3 columns

We clean the text and generate our X (text, no chosen features) and our y (labels, 1 if *verifiable* 0 if *not verifiable*).

```
# Cleaning the text and generating the data sets:

sentences = list(df['text'])
text = np.array([str.encode(preprocess_text(sen)) for sen
        in sentences], dtype=object)
```

```
y = df['label']
y = np.array(list(map(lambda x: 1 if x=="Verifiable"
        else 0, y)), dtype='int32')
text[:3]

### Output:
# 1 'controversial cross dresser idris okuneye aka bobrisky has
# unveiled his plans for his st birthday ',
# 2 'tinubu presidency will produce wealth prosperity'
# 3 'pdp was corrupt in and nigerians needed to remove them from power '
```

We divide our data set into training and validation:

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(text,
            y, test_size=0.20)
```

Until now we have already seen how to do all of this. Now, we must do several things: define the model, download and initiate our model with the already trained weights, define the hyper-parameters and train our model. We start by defining the input layer:

```
# URLs to download the weights for both layers (encoder and pre-process):
tfhub_handle_encoder = "https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L
    -4_H-512_A-8/2"
tfhub_handle_preprocess = "https://tfhub.dev/tensorflow/bert_en_uncased_preprocess
    /3"

# Defining the input layer:
text_inputs = [tf.keras.layers.Input(shape=(), dtype=tf.string)]

# Download the Pre-Processor weights:
preprocessor = hub.load("https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3
    ")
tokenize = hub.KerasLayer(preprocessor.tokenize)
tokenized_inputs = [tokenize(segment) for segment in text_inputs]

# Compile the whole input layer:
bert_pack_inputs = hub.KerasLayer(preprocessor.bert_pack_inputs,
        arguments=dict(seq_length=256))
```

Notice we have chosen already one hyper-parameter: **the maximum sequence length**, which we put out as 256. This hyper-parameter is really important since it will determine how big our sequence vectors are. If we have a sentence of length 250, the model will pad out the remaining six slots with 0s. Since this are mostly Facebook and Twitter posts, we chose 256 as to not have vectors with tons of 0s. Now we must define the encoder layer:

```
# Bridge between tokens and embeddings

encoder_inputs = bert_pack_inputs(tokenized_inputs)

# Download and define the encoder layers
encoder = hub.KerasLayer(tfhub_handle_encoder, trainable=True, name='BERT_encoder'
    ) # BERT embedding and encoding

embedded = encoder(encoder_inputs)
```

Finally we aggregate the final classification layer, which will have a linear activation function, as well as a DropOut for regularization:

```
# Connection between the trained model and the additional layers:
net = embedded['pooled_output']

# Adding DropOut
net = tf.keras.layers.Dropout(0.1)(net)

# Adding final Dense Layer (Classifier)
net = tf.keras.layers.Dense(1, activation=None, name='classifier')(net)

# Final Compilation:
model_BERT = tf.keras.Model(text_inputs, net)
```

Now, before training, we must define another set of hyper-parameters: loss function, optimization algorithm, learning rate and epochs:

```
epochs = 8
steps_per_epoch = 500
num_train_steps = steps_per_epoch * epochs
num_warmup_steps = int(0.1*num_train_steps)
```

```
init_lr = 3e-5
optimizer = optimization.create_optimizer(init_lr=init_lr,
        num_train_steps=num_train_steps,
        num_warmup_steps=num_warmup_steps,
        optimizer_type='adamw')

model_BERT.compile(optimizer=optimizer,
        loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
        metrics=tf.metrics.BinaryAccuracy())
```

We also define a callback function to go back to the model with the best accuracy (Early Stopping):

```
checkpoint_path = 'drive/MyDrive/bert_train/models/tensor/cp7.cpkt'
checkpoint_dir = os.path.dirname(checkpoint_path)

# Create a callback that saves the model's weights
cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
    save_weights_only=True,
    save_best_only=True,
    verbose=1)
```

And we can finally train the model:

```
history = model_BERT.fit(x=x_train, y=y_train, epochs=8,
        validation_data = (x_test, y_test), callbacks=[cp_callback])
```

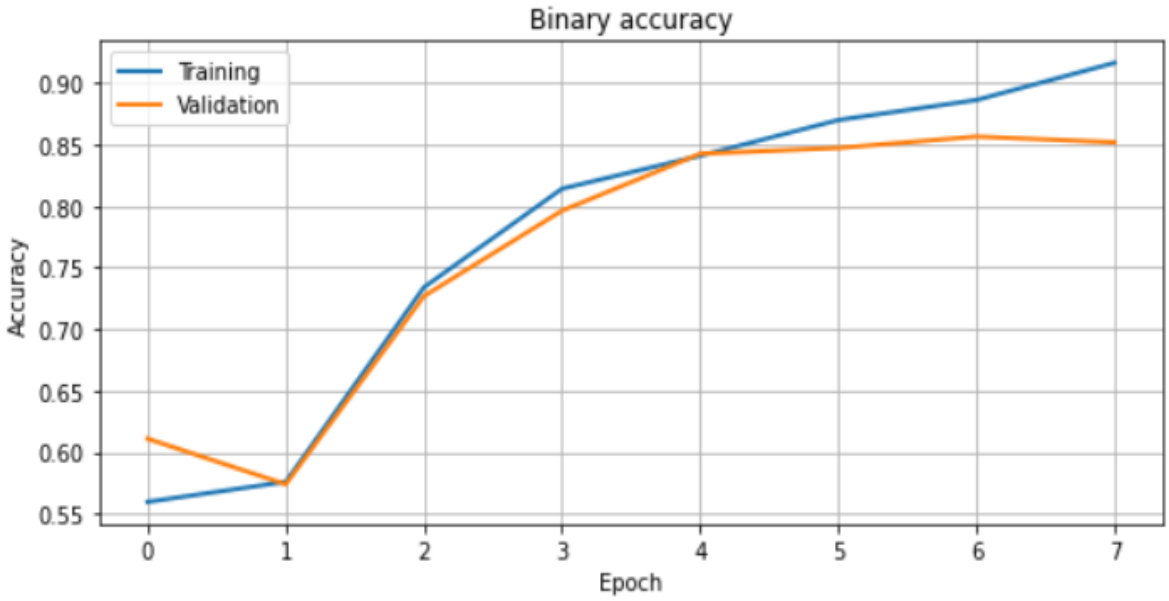
We evaluate the best model using the usual metrics. First the learning curve:

```
# Plot loss
plt.figure(figsize=(20, 4))

plt.title('Binary accuracy')
plt.plot(history.history['binary_accuracy'], label='Training', linewidth=2)
plt.plot(history.history['val_binary_accuracy'], label='Validation', linewidth=2)
plt.legend()
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.grid(True)

plt.show()
```

Figure 12: Learning Curve



We recover the best model weights to check the accuracy:

```
latest = tf.train.latest_checkpoint(checkpoint_dir)
# Load the previously saved weights
model_BERT.load_weights(latest)

predictions = model_BERT.predict(x_test).flatten()
y_pred_nn = (predictions > 0.5).astype(np.int32)
```

We obtained an accuracy of 86% in validation, which is pretty good for this type of task:

```
loss, accuracy = model_BERT.evaluate(x_test, y_test)

print(f'Accuracy: {accuracy}')

### Output:
0.8564814925193787
```

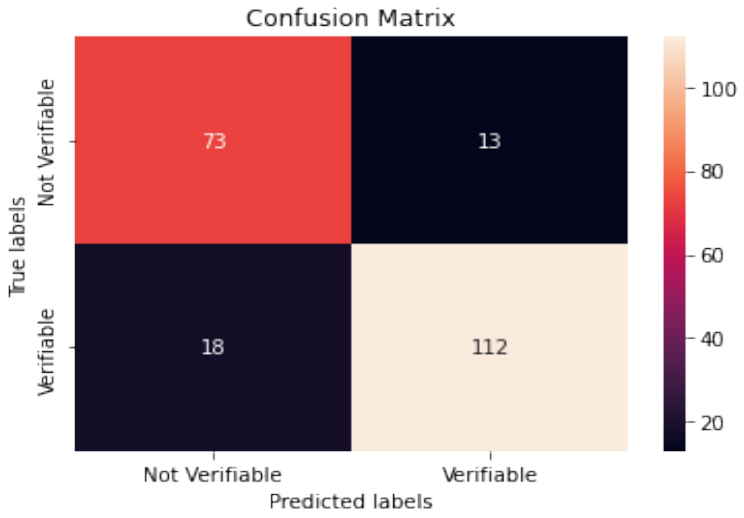
Finally we can check which class is the most problematic, by plotting the confusion matrix:

```
cm = tf.math.confusion_matrix(y_test, y_pred_nn)
```

```
    ax= plt.subplot()
    sns.heatmap(cm, annot=True, fmt='g', ax=ax);

# labels, title and ticks
    ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
    ax.set_title('Confusion Matrix');
    ax.xaxis.set_ticklabels(['Not Verifiable', 'Verifiable']); ax.yaxis.set_ticklabels
        (['Not Verifiable', 'Verifiable'])
```

Figure 13: Confusion Matrix



References

Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning*. https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf. Springer.

Devlin, Jacob et al. (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *arXiv:1810.04805*.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2008). *The Elements of Statistical Learning*. https://hastie.su.domains/Papers/ESLII.pdf. Springer Series in Statistics.

Ioffe, Sergey and Christian Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *arXiv:1502.03167v3*.

Kingma, Diederik P. and Jimmy Ba (2017). "Adam: A Method for Stochastic Optimization". In: *arXiv:1412.6980v9*.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep Learning". In: *Nature* 521, pp. 436–444.

Reed, Russell and Robert J. Marks (2016). *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. MIT Press.

Ruder, Sebastian (2016). "An overview of gradient descent optimization algorithms". In: *arXiv:1609.04747v2*.

Srivastava, Nitish et al. (2014). "Dropout: A Simple Way to Prevent Neural Networks from Over-fitting". In: *Journal of Machine Learning Research* 15(1), pp. 1929–1958.

Vaswani, Ashish et al. (2017). "Attention is all You Need". In: *arXiv:1706.03762*.

Zhang, Chiyuan et al. (2017). "Understanding Deep Learning Requires Rethinking Generalization". In: *arXiv:1611.03530v2*.