

**A tool that can  
states if two texts  
have the same  
meaning**

Thomas Guegan & Mickaël Bobovitch





# Agenda

1. Text processing
2. Cosine similarity
3. Lexical Similarity
4. Graphical User Interface
5. Possible improvements



# Introduction

- Plagiarism detection: The tool can be used to check if a given text is copied or plagiarized from another source by comparing its meaning with existing texts.
- Content filtering: It can be employed to identify and filter out duplicate or redundant content
- Text summarization: By comparing the meaning of a summarized text with the original document, the tool could help to determine if the essence and key points have been preserved.
- Question-answering systems: The tool can verify if a user's question has been previously answered by comparing its meaning with a database of existing questions and answers.



# Text Preprocessing

```
stop_words = set(stopwords.words('english'))

def preprocess_text_nltk(text):
    tokens = word_tokenize(text.lower()) # Tokenize and convert to lowercase
    tokens = [word for word in tokens if word.isalpha()] # Remove non-alphabetic characters
    tokens = [word for word in tokens if word not in stop_words] # Remove stopwords

    lemmatizer = nltk.WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(word) for word in tokens] # Lemmatize words

    return tokens
```



# Cosine similarity

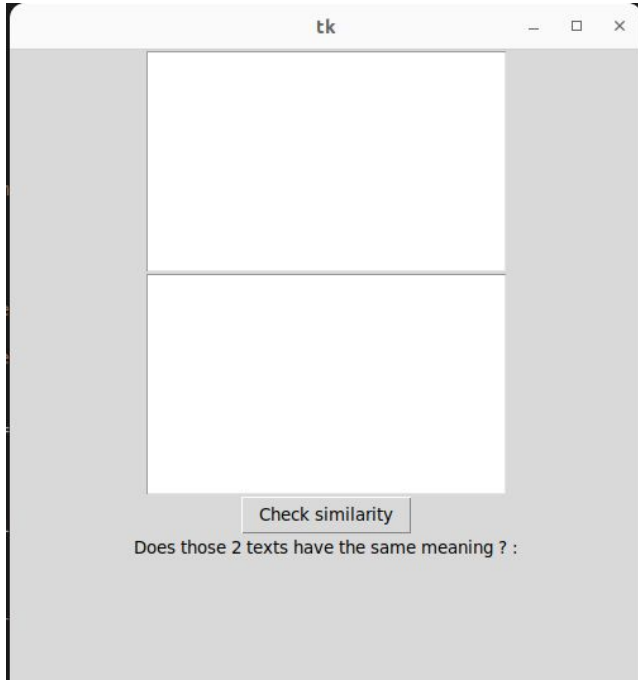
```
def calculate_soft_cosine_similarity(text1, text2):  
    tfidf_vectorizer = TfidfVectorizer()  
    tfidf_matrix = tfidf_vectorizer.fit_transform([text1, text2])  
    soft_similarity = cosine_similarity(tfidf_matrix)[0][1]  
  
    return soft_similarity  
  
def calculate_similarity_nltk(text1, text2):  
    tokens1 = preprocess_text_nltk(text1)  
    tokens2 = preprocess_text_nltk(text2)  
  
    # Create a set of unique words from both texts  
    unique_words = list(set(tokens1 + tokens2))  
  
    # Generate word vectors for the unique words  
    vector1 = [tokens1.count(word) for word in unique_words]  
    vector2 = [tokens2.count(word) for word in unique_words]  
  
    # Calculate cosine similarity  
    similarity = cosine_similarity([vector1], [vector2])[0][0]  
  
    return similarity
```



# Lexical Similarity

```
def word_similarity(sentence1,sentence2):  
    # Extract POS tags  
    nouns1, adjectives1, verbs1 = extract_pos_tags(sentence1)  
    nouns2, adjectives2, verbs2 = extract_pos_tags(sentence2)  
  
    noun_sim = []  
    for i in nouns1:  
        for j in nouns2:  
            #print(i, " ",j ," ", i.similarity(j) )  
            noun_sim.append(i.similarity(j))  
  
    n_sim_value = 0  
    for i in noun_sim:  
        n_sim_value = n_sim_value+i
```

# Graphical User Interface (GUI) using Tkinter



- Provide lot of widget to create simple user interface
- Easy to use
- High level of personnalisation



# Possible improvements

- Possible use of a Bert-model that can process text and words in their context and not alone

*It's possible to use pre-trained Bert model using the transformers library in Python*

- Improve the lexical analyzer using more accurate formulas

