

**Politechnika Krakowska im.  
Tadeusza Kościuszki**

Wydział Informatyki i Telekomunikacji

***Mikhail Ermolaev***

Numer albumu: 139667

***David Burchakov***

Numer albumu: 139685

**Porównanie różnych metod identyfikacji podobnych tekstów**

**Przetwarzanie Języka Naturalnego**

**Kraków, 2023**

## Abstract

Praca dotyczy opracowania narzędzia do identyfikacji podobnych tekstów, wykorzystującego techniki przetwarzania języka naturalnego. Podane badania obejmują analizę uzyskanych wyników w zależności od rodzaju i jakości danych wejściowych, a także technik analizy danych stosowanych do porównywania tekstów. W ramach eksperymentów również zbadano skuteczność różnych metod wstępnego przetwarzania danych wejściowych. Celem było stworzenie efektywnego narzędzia, zdolnego do wyszukiwania podobieństw w bazie danych, oraz precyzyjnej identyfikacji podobieństw tekstualnych w różnych kontekstach. Praca obejmuje porównania między popularnymi algorytmami porównywania tekstu, takimi jak TF-IDF, Word Embeddings, GloVe i FastText w celu zidentyfikowania optymalnych strategii w kontekście identyfikacji podobieństw tekstowych. Wyniki eksperymentów pozwalają na lepsze zrozumienie skuteczności poszczególnych metod oraz wskazują kierunki dalszych badań w dziedzinie przetwarzania języka naturalnego.

Słowa kluczowe: przetwarzanie tekstu, algorytmy, identyfikacja podobieństwa.

## Spis treści

Abstract	2
Spis treści	2
Wstęp	3
Definicja problemu	3
Szczegóły implementacji i użyte algorytmy	4
Podejście	5
Wyniki	10
Podsumowanie	10
Bibliografia	11

## I. INTRODUCTION

W dzisiejszych czasach globalizacji, ilość dostępnych tekstów stale rośnie, co stawia przed ludźmi wyzwanie efektywnej analizy i przetwarzania informacji. W tym celu narzędzia do identyfikacji podobnych tekstów stają się niezwykle istotne, umożliwiając automatyczne wyodrębnianie wzorców, klasyfikację dokumentów oraz odkrywanie ukrytych relacji między nimi. W związku z tym, Przetwarzanie Języka Naturalnego stało się kluczową dziedziną, skupiającą się na rozumieniu i manipulacji języka naturalnego przez komputery.

Porównywanie tekstów to nie tylko wyzwanie techniczne, ale również zagadnienie praktyczne. W dobie masowego dostępu do informacji, możliwość automatycznego identyfikowania podobnych treści może znacząco wspomóc zarówno badaczy, jak i przedsiębiorstwa, przyspieszając procesy analizy i wnioskowania. Dodatkowo, rozwój zaawansowanych narzędzi do porównywania tekstów ma potencjał zrewolucjonizować dziedziny takie jak wyszukiwanie informacji, kategoryzacja treści czy personalizacja doświadczeń użytkownika.

Warto podkreślić, że współczesne podejście do przetwarzania języka naturalnego umożliwia nie tylko analizę pojedynczych słów, lecz także uwzględnia kontekst, semantykę, czy nawet składnię zdań. To otwiera drzwi do bardziej zaawansowanych i precyzyjnych metod identyfikacji podobieństw tekstowych. W miarę postępu technologicznego, badania nad narzędziami do porównywania tekstów stają się coraz bardziej aktualne i fascynujące, wprowadzając innowacyjne rozwiązania w dziedzinie analizy danych tekstowych.

## II. Definicja Problemu

W kontekście przetwarzania języka naturalnego, głównym wyzwaniem jest skuteczne identyfikowanie podobieństw między tekstami. Problem ten obejmuje szeroki zakres zastosowań, od wyszukiwania informacji w bazie danych po generację tekstu o podobnej tematyce. Rozwiązanie tego problemu ma zastosowanie w różnych kontekstach, takich jak efektywne wyszukiwanie informacji, analiza sentimentu, czy nawet tworzenie wariantów tekstowych. Kluczowymi wyzwaniami są skuteczne porównywanie i ocenianie tekstów, uwzględniając zarówno aspekty semantyczne, jak i strukturalne. Skupmy się na trzech aspektach problemu analizy tekstu:

### **A: Wyszukiwanie Podobnych Zdań z Bazy Danych**

W ramach pierwszego aspektu rozwiązania, istotne jest stworzenie mechanizmu umożliwiającego efektywne wyszukiwanie podobnych zdań w dużych zbiorach danych tekstowych. To wymaga zrozumienia kontekstu semantycznego zdań, a także skutecznego indeksowania tekstu w bazie danych.

### **B: Wyliczenie Stopnia Podobieństwa Dwóch Podanych Tekstów**

Drugi aspekt skupia się na stworzeniu metody umożliwiającej obiektywne i dokładne wyliczenie stopnia podobieństwa między dwoma tekstami. To wymaga uwzględnienia zarówno wspólnych słów, jak i ich znaczenia kontekstowego, aby uzyskać miarę podobieństwa adekwatną do ludzkiego postrzegania.

### **C: Generacja podobnych sentencji**

Trzeci aspekt problemu dotyczy stworzenia narzędzia umożliwiającego generację tekstu o charakterze zbliżonym do tekstu wejściowego. Wyzwaniem jest nie tylko zachowanie semantyki, ale również uwzględnienie stylu i kontekstu tekstu oryginalnego, co wymaga rozwiniętych modeli generatywnych.

## **III. Szczegóły implementacji i użyte algorytmy**

W implementacji narzędzia do identyfikacji podobnych tekstów wykorzystano szereg bibliotek i narzędzi do przetwarzania języka naturalnego. Korzystając z języka Python, użyto bibliotek takich jak Pandas, Scikit-learn, NLTK, Gensim, oraz spaCy. Poniżej przedstawiono kluczowe elementy implementacji:

#### Przygotowanie Danych:

Dane tekstowe zostały zaimportowane przy użyciu biblioteki Pandas, umożliwiającej łatwą manipulację i analizę danych tabularnych.

#### Pre-processing Tekstu:

Wykorzystano NLTK do pobrania listy stop słów oraz do tokenizacji tekstu przy użyciu `casual_tokenize`. Ponadto, zastosowano funkcje do rozszerzania skrótów i usuwania znaków interpunkcyjnych.

#### TF-IDF Vectorization:

Zaimplementowano TF-IDF Vectorizer z biblioteki Scikit-learn.

### Modele Word Embeddings:

Stworzono model Word2Vec z biblioteki Gensim, umożliwiający reprezentację słów w postaci wektorów numerycznych. Ten model został użyty do obliczania podobieństwa semantycznego między słowami.

### Cosine Similarity:

Zaimplementowano obliczanie podobieństwa kosinusowego przy użyciu funkcji `cosine_similarity` z Scikit-learn, zarówno dla wektorów TF-IDF, jak i dla wektorów uzyskanych z modelu Word2Vec.

### Optymalizacja i Efektywność:

Przeprowadzono eksperymenty z parametrami i metodami, takimi jak PCA, w celu optymalizacji efektywności narzędzia. Uwzględniono także stopniowość przetwarzania danych, aby zminimalizować wpływ zakłóceń w dużych korpusach tekstowych.

## **IV. Podejście do Pracy**

### **A. Wyszukiwanie Podobnych Zdań z Bazy Danych**

W procesie przetwarzania wstępnego tekstu zastosowano różnorodne metody, mające na celu zoptymalizowanie jakości analizy oraz skuteczność identyfikacji podobnych tekstów. Podejście to obejmuje kilka kluczowych etapów, z których każdy przyczynia się do poprawy jakości reprezentacji tekstu:

**Usuwanie stop słów** - eliminacja słów powszechnie występujących, ale niewnoszących istotnej informacji semantycznej, co pozwala skoncentrować się na kluczowych elementach tekstu.

**Rozwijanie Skrótów** - proces rozwijania skrótów w celu zachowania pełniejszego znaczenia słów, co może wpłynąć na poprawę dokładności analizy semantycznej.

**Usuwanie Końcówek Dzierżawczych** - eliminacja końcówek dzierżawczych, które nie są skrótami, aby znormalizować formę wyrażen i uniknąć fałszywych podobieństw.

**Lematyzacja** - redukcja słów do ich podstawowej formy, co pozwala na jednolite traktowanie różnych form tego samego słowa i poprawia analizę semantyczną.

**Usuwanie Znaków Specjalnych** - eliminacja znaków specjalnych, co przyczynia się do poprawy spójności i czytelności tekstu.

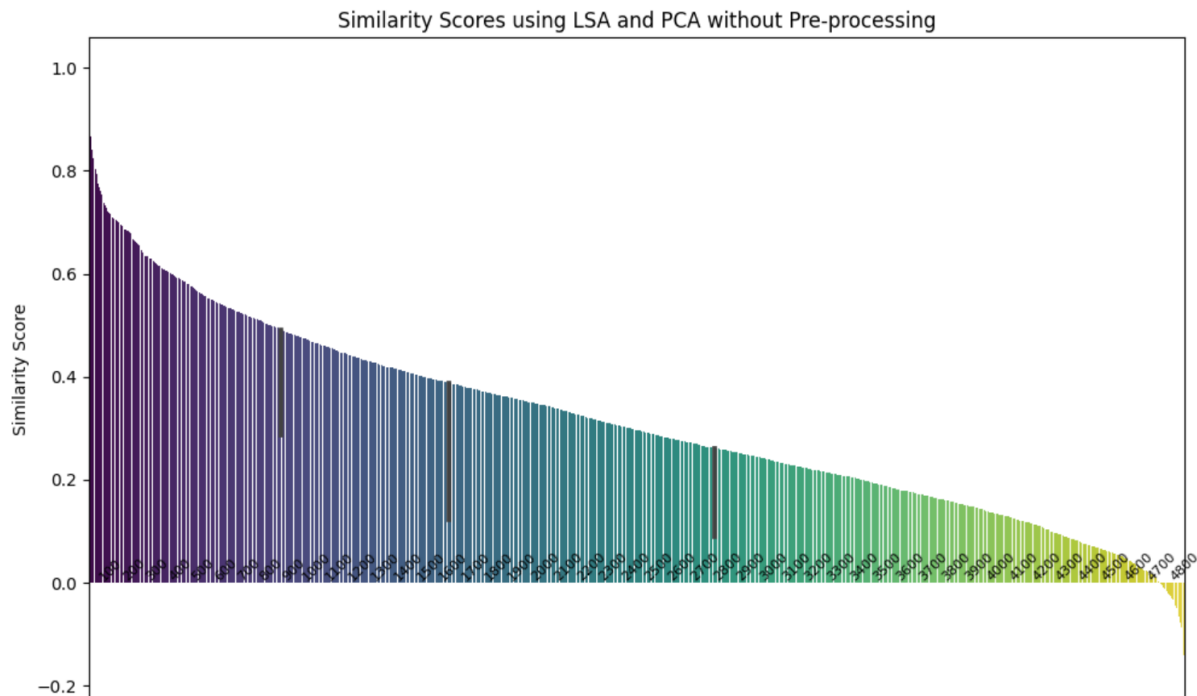
**Obsługa wielkich liter** - optymalizacja obsługi wielkości liter, aby uniknąć błędów wynikających z różnic w pisowni.

**Pisanie nazw własnych z wielkiej litery** - zastosowanie reguł do poprawy pisowni nazw własnych, co przyczynia się do lepszej identyfikacji istotnych elementów tekstu.

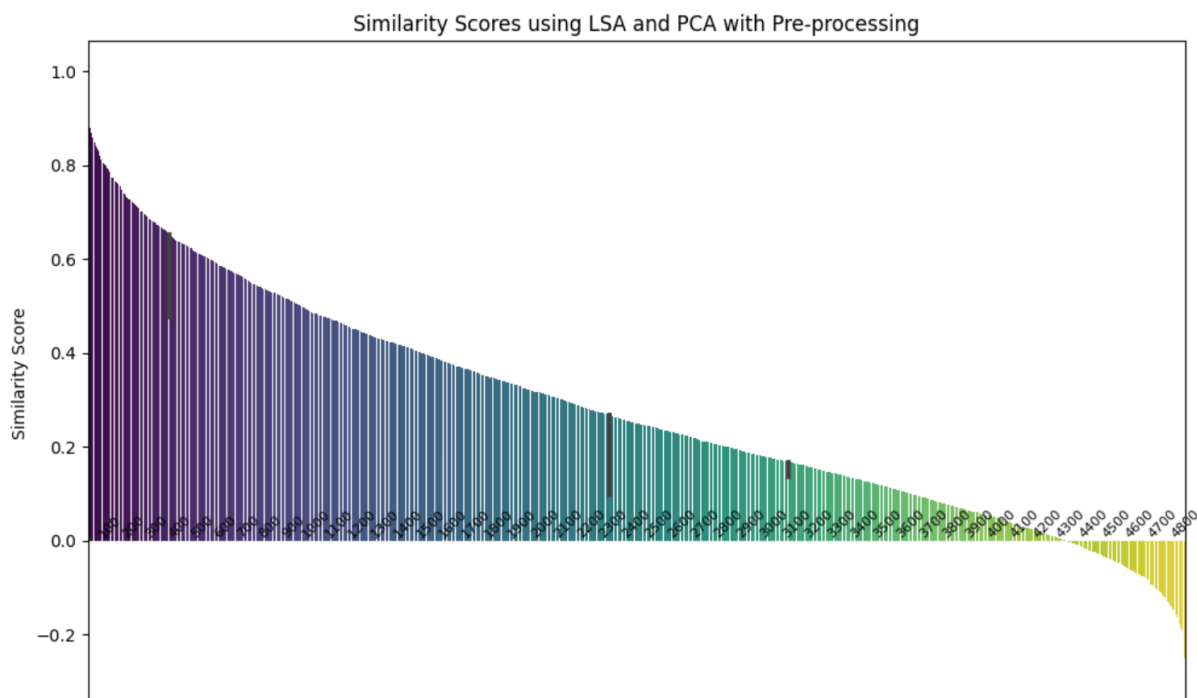
Aby ocenić wpływ przetwarzania wstępnego na analizę tekstu, porównano wyniki analizy z użyciem i bez użycia opisanych metod. Aspekt A prezentuje rezultaty porównania podobieństwa tekstów za pomocą technik LSA i PCA, co pozwala na ocenę skuteczności zastosowanego wstępnego przetwarzania.

Obie implementacje metod analizy tekstu, tj. Latent Semantic Analysis (LSA) i Principal Component Analysis (PCA), zostały zrealizowane w kontekście przetwarzania wstępnego tekstu. W pierwszym przypadku, dla LSA, użyto macierzy TF-IDF oraz zastosowano TruncatedSVD dla redukcji wymiarów. Analogicznie, dla PCA, również zastosowano macierz TF-IDF, a redukcję wymiarów przeprowadzono za pomocą PCA.

W pierwszym przypadku wykorzystano metodę LSA z wykorzystaniem przestrzeni semantycznej (PSA), przy czym nie zastosowano żadnej wstępnej obróbki tekstu. W drugim przypadku, przed przeprowadzeniem analizy LSA z PSA, tekst wejściowy został poddany procesowi przetwarzania przy użyciu wcześniej opisanych metod.

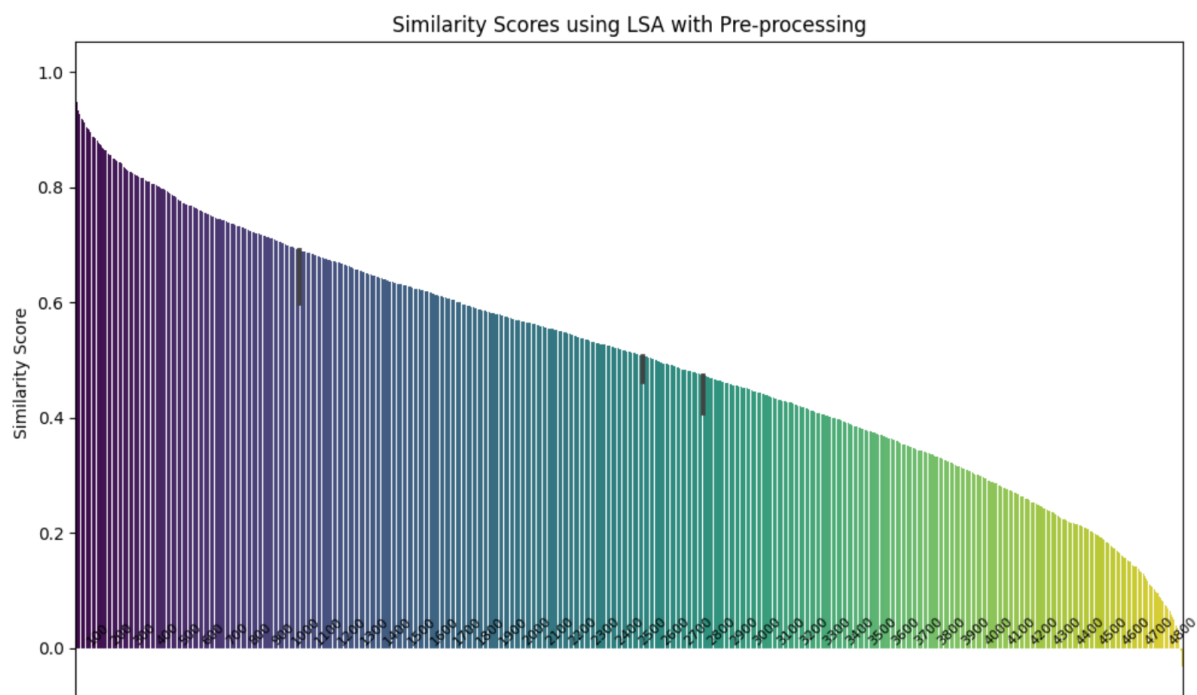


rys. 1 - porównanie prawdopodobieństwa jednego elementu z bazy danych do wszystkich innych bez użycia wstępnego przetwarzania



rys. 2 - porównanie prawdopodobieństwa jednego elementu z bazy danych do wszystkich innych z użyciem wstępnego przetwarzania

Analiza wykresów wyraźnie ukazuje, że wprowadzenie wstępnego przetwarzania tekstu w drugim przypadku wpłynęło na ogólną wartość podobieństwa między analizowanymi kolumnami, powodując spadek. Jednocześnie, zauważalny jest wzrost liczby filmów otrzymujących średnie wyniki podobieństwa. Ten efekt może być rezultatem eliminacji wpływu stopwords na proces porównywania, a jednocześnie zwiększenia istotności znaczących słów dzięki lematyzacji, co skutkuje bardziej precyzyjną analizą semantyczną w kontekście identyfikacji podobieństw tekstowych.



rys. 3 - porównanie prawdopodobieństwa jednego elementu z bazy danych do wszystkich innych z użyciem wstępnego przetwarzania bez redukcji wymiarów

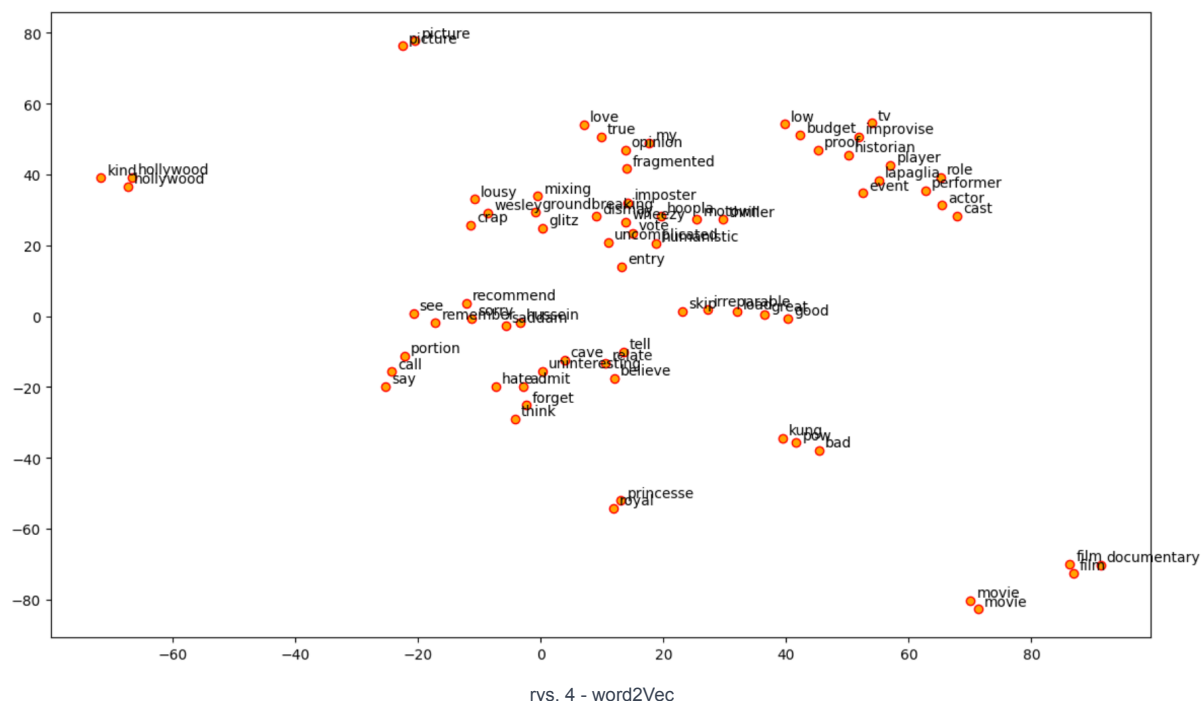
Podczas analizy wyników metody LSA oraz kombinacji LSA i PCA, można zauważyć, że oba podejścia prezentują zbliżone rezultaty dla najbardziej podobnych opcji. Jednakże, warto zauważyć, że wyniki dla najmniej podobnych obiektów w przypadku metody LSA bez użycia PCA wydają się być przeceniane. Otrzymane rezultaty sugerują, że dodanie kroku redukcji wymiarów za pomocą PCA może wpływać na bardziej zrównoważone oceny podobieństwa, szczególnie w kontekście obiektów o niskim stopniu podobieństwa. Ta obserwacja wskazuje na potencjalną wartość dodaną, jaką wnosi PCA, poprawiając wiarygodność analizy semantycznej w przypadku mniej zbieżnych tekstów.

## B. Wyliczenie Stopnia Podobieństwa Dwóch Podanych Tekstów

Drugim etapem pracy jest wyliczenie stopnia podobieństwa dwóch podanych tekstów. W celu osiągnięcia tego celu, zdecydowaliśmy się skorzystać z zaawansowanych metod przetwarzania tekstu, takich jak: Word2Vec, GloVe oraz FastText.

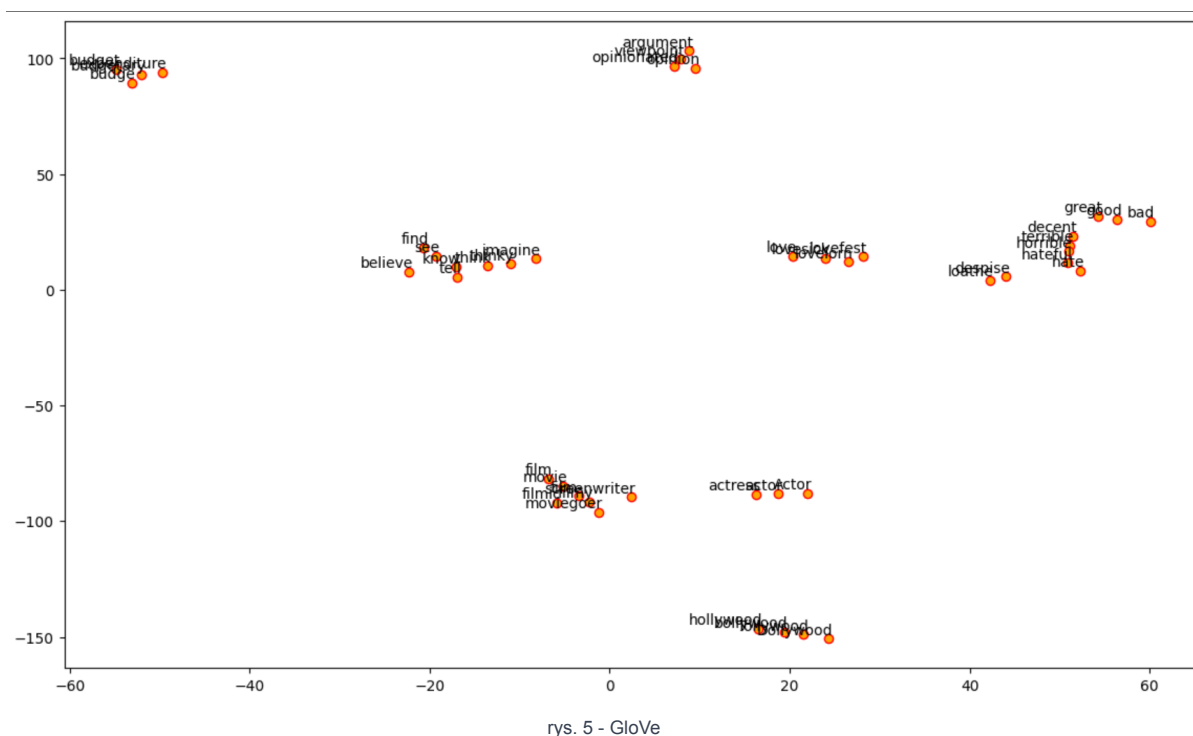
### Word2Vec VS GloVe

Word2Vec i GloVe to dwie popularne metody reprezentacji słów w przestrzeni wektorowej. Obydwie opierają się na kontekstowym zrozumieniu słów, jednak istnieją różnice w ich podejściu. Word2Vec skupia się na predykcji słów w oparciu o najbliższe słowa (kontekst), podczas gdy GloVe bazuje na współwystępowaniu słów w korpusie. W wyniku tego, otrzymane wyniki z obu metod są zbliżone, lecz warto zauważyć, że różnice w korpusie treningowym i specyfika algorytmów mogą wpływać na ich subtelne różnice w rezultatach.



rys. 4 - word2Vec





## GloVe

### Dataset

Dla trenowania Modelu GloVe użyty był dataset z [Rotten Tomatoes Review](#) (Recenzje). Pozytywne recenzje oznaczone jako 'Fresh' (świeży), a negatywne jako "Rotten" (zgniły), lecz właśnie dla naszego problemu - identyfikacji podobnych sentencji - oznaczenia nie są istotne.

### Metoda GloVe

Metoda GloVe bada współwystępowanie i związki między słowami w całym korpusie.

Na przykład, rozważając, jak często słowa takie jak "ice" (*lód*) i "steam" (*para*) współwystępują z innymi słowami, takimi jak "solid" (*stały*) lub "gas" (*gaz*).

Oczekuje się, że słowo testowe "solid" (*stały*) będzie współwystępować częściej ze słowem "ice" (*lód*) niż ze słowem "steam" (*para*).

Jeśli słowo testowe jest związane z oboma lub żadnym, na przykład "water" (*woda*) i "fashion" (*moda*), oczekiwany stosunek prawdopodobieństwa występowania powinien być bliski 1.

Relację tych słów można badać, analizując stosunek ich prawdopodobieństwa współwystępowania z różnymi słowami testowymi.

Używamy **'en\_core\_web\_lg'** model z biblioteki Spacy, który już zawiera wstępnie wyszkolone, gotowe wektory słów, oparte na algorytmie GloVe ([Global Vectors for Word Representation](#)), który został opracowany przez naukowców z Uniwersytetu Stanforda.

```
import spacy

nlp = spacy.load('en_core_web_lg') # includes word vectors based on the GloVe model

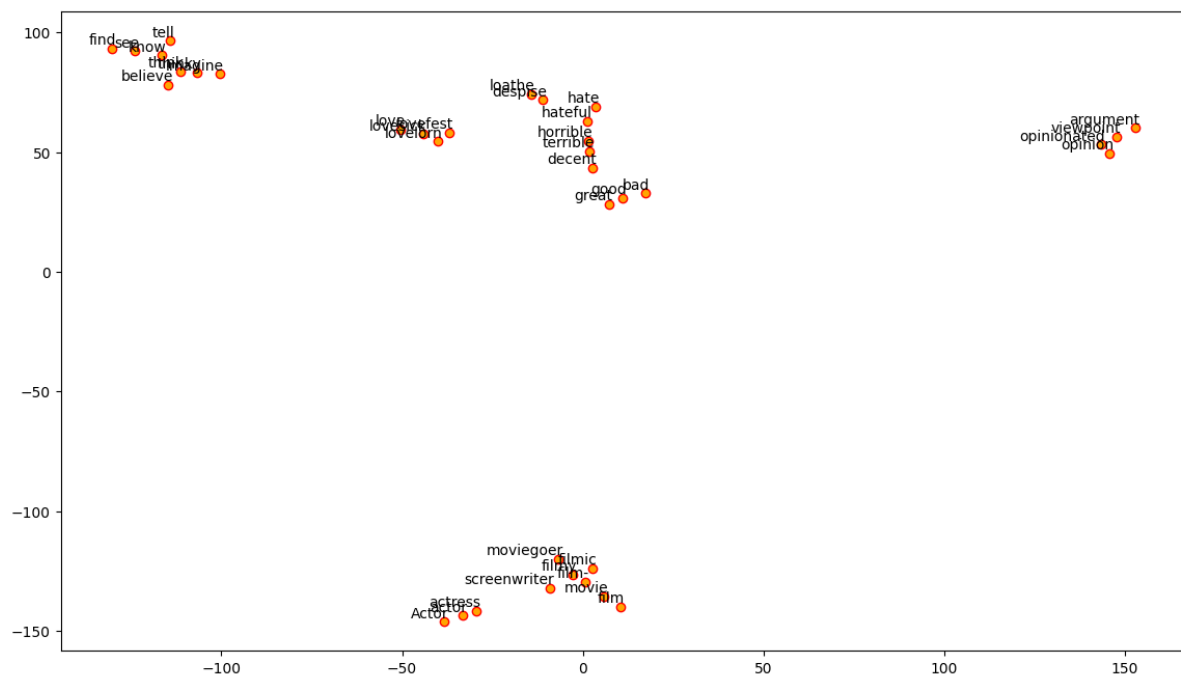
total_vectors = len(nlp.vocab.vectors)

print('Total word vectors:', total_vectors)
```

*Total word vectors: 514157*

Możemy znaleźć najbliższe słowa do słów testowych, na przykład

```
terms of interest = ['movie', 'film', 'good', 'bad', 'hate', 'love',
                    'actor', 'opinion', 'think', 'see']
```



Można zauważyć, że model właściwie znalazł podobne słowa, i są raczej synonimy, np.

*“Hate” - “Loathe”, “Despise” („Nienawidzić” – „Brzydzić”, „gardzić”)*

Aczkolwiek, warto też zwrócić uwagę, że nie wszystkie słowa mające podobny kontekst są synonimami, np.

“Good” - “Great”, “Bad” - (“Dobry” - “Wyśmienity”, “Zły”)

Słowa “Dobry” i “Zły” często występują w podobnych kontekstach, po to znajdują się blisko na wektorze.

### Identyfikacja podobnych sentencji

Po nauczaniu modelu ‘en\_core\_web\_lg’ naszymi wpisami z datasetu Rotten Tomatoes oraz metody **cosine similarity**, możemy znaleźć podobieństwo dwóch poszczególnych zdań.

```
sentence1 = "This movie was truly amazing and had a great story."
sentence2 = "The film was incredibly good and the plot was
outstanding."
```

*The similarity between the sentences is: 85.60%*

Także, możemy wyszukać podobieństwo osobno każdego słowa do każdego słowa w zdaniach i wyliczyć średnie podobieństwo. Natomiast, sprawia to, że niektóry kontekst zostaje stracony.

	.	film	good	incredibly	outstanding	plot
.	1.000000	-0.009095	0.288674	0.266851	0.326775	0.109322
amazing	0.225753	0.084596	0.590898	0.567639	0.483716	0.051988
great	0.300736	0.082885	0.755130	0.475666	0.592463	0.069649
movie	0.023211	0.833277	0.139775	0.183301	0.016186	0.445783
story	0.096732	0.524190	0.114741	0.297797	0.233712	0.598915
truly	0.173340	0.081420	0.387853	0.652918	0.313164	0.179490
Average Similarity: 0.32						

*Przeprowadzimy podobną analizę dla niepodobnych zdań.*

```
sentence1 = "This movie was truly amazing and had a great story."
```

```
sentence2 = "My cat is very biting, so I never pick her up"
```

*The similarity between the sentences is: 51.66%*

	,	biting	cat	never	pick
.	0.425803	0.378945	0.014464	0.278176	0.183916
amazing	0.115804	0.227776	0.121694	0.324187	0.230786
great	0.269700	0.350155	0.054076	0.357812	0.166819
movie	0.006554	0.066448	0.118379	0.233778	0.071128
story	0.232640	0.246280	0.078144	0.282937	-0.034338
truly	0.055032	0.139608	0.139030	0.448159	0.104447

*Average Similarity: 0.19*

### **Symulacja generacji sentencji**

Za pomocą metod opartych na word embeddings, możemy “wygenerować”, a raczej zasymulować generacje, sparafrazować zdanie, zamieniając poszczególne słowa na najbardziej bliskie na przestrzeni wektorowej.

**Old sentence:** *The movie is a great plot, the director is a good person. The actors performed very well. Probably the best film of 2014!*

**New sentence:** *the moviegoer be a fantastic plotline , the codirector be a great personhood . the Actress performative extremely welly . Improbably the great filmy of 2014 !*

*Stworzyliśmy skrypt który zamienia słowa na podstawie ich części mowy, i.e czasowniki na czasowniki, rzeczowniki na rzeczowniki, etc.*

Aczkolwiek, nawet biorąc pod uwagę części mowy, to sparafrazowanie nie daje dobre wyniki (“well” => “welly”, “probably” => “improbably”, “actors” => “personhood”)

## FastText

FastText to model i metoda opracowana przez zespół badawczy z Facebook AI Research (FAIR), służąca do generowania wektorów słów (word embeddings) oraz klasyfikacji tekstów. FastText jest szczególnie znany z tego, że oprócz uwzględniania całych słów w procesie nauki, bierze pod uwagę także ich fragmenty, czyli n-gramy literowe. [Więcej informacji o FastText](#).

FastText udostępnia już gotowy model, natrenowany na danych z całej Wikipedii oraz sajtu Common Crawl.

```
import fasttext

# Load the pre-trained FastText model

model_en = fasttext.load_model('cc.en.300.bin')
```

Skoro FastText model jest już wyuczony, możemy od razu przejść do wyliczania podobieństwa sentencji.

Wyliczymy podobieństwo każdego słowa osobno oraz całego zdania razem.

```
sentence1 = "The movie was fantastic and thrilling."
sentence2 = "I found the film to be exciting and wonderful."
```

	i	found	the	film	to	be	exciting	and	wonderful.
<b>the</b>	0.141125	0.258783	1	0.2078	0.373638	0.261259	0.164036	0.43708	0.062681
<b>movie</b>	0.131207	0.075474	0.191173	0.773174	0.045066	0.074241	0.150432	0.090201	0.072723
<b>was</b>	0.267145	0.490676	0.358188	0.169538	0.248655	0.43198	0.197481	0.422343	0.254178
<b>fantastic</b>	0.132722	0.220809	0.217391	0.164246	0.092706	0.211421	0.559153	0.244111	0.485977
<b>and</b>	0.233203	0.275322	0.43708	0.101709	0.364097	0.314614	0.198858	1	0.220625
<b>thrilling.</b>	0.45481	0.086195	0.099228	0.140663	0.051223	0.213135	0.381754	0.17585	0.668105

```
import numpy as np

from scipy.spatial.distance import cosine

def preprocess(sentence):

    return sentence.lower()

def sentence_to_vector(sentence, model):

    words = sentence.split()

    word_vectors = [model.get_word_vector(word) for word in words if
word in model]

    sentence_vector = np.mean(word_vectors, axis=0)

    return sentence_vector

def calculate_similarity(sentence1, sentence2, model):

    vector1 = sentence_to_vector(preprocess(sentence1), model)

    vector2 = sentence_to_vector(preprocess(sentence2), model)

    sim = 1 - cosine(vector1, vector2)

    return sim * 100 # Convert to percentage

sentence1 = "The movie was fantastic and thrilling."

sentence2 = "I found the film to be exciting and wonderful."

similarity_percentage = calculate_similarity(sentence1, sentence2,
model_en)

print(f"Similarity: {similarity_percentage}%")
```

*Similarity: 68.79260540008545%*

Jak wcześniej, możemy zasymulować generację nowego zdania, zamieniając każde słowo na najbliższe na wektorze.

**Input:**

*The movie is a great plot, the director is a good person. The actors performed very well. Probably the best film of 2014!*

**Output:**

*ththe film It A fantastic Pejeta ththe co-director It A bad individual. ththe actresses peformed extremely too. probaby ththe finest films ofn 201588*

*Chociaż FastText jest znany z tego, że jest dobrym algorytmem, z naszego doświadczenia wynika, że dał gorszy wynik.*

## C. Generacja podobnych sentencji

### dataset

Dla generacji podobnych sentencji był użyty dataset [Opusparcus: Open Subtitles Paraphrase Corpus for Six Languages \(version 1.0\)](#), który zawiera zdania oraz ich parafrazy.

	text	paraphrase
0	You 're not alone , Claire .	You are not alone , Claire .
1	Who told you to throw acid at Vargas , hmm ?	Who told you to throw acid at Vargas ?
2	Where the pure angel merges with the antic Sphinx	Where the pure angel merges with the antic sphynx .
3	Where is it written what is it I 'm meant to be	Where is it written what it is I 'm meant to be
4	We 'll find the skipper and then we 'll go home .	We 'll find the skipper and then we go home .
5	Seymour 's Darling is third ... and little Arnie moving fast on the outside .	Seymour 's Darling is third ... and little Arnie moving fast to the outside .
6	Scud , do you read me ?	Scud , you reading me ?
7	Jumby now wants to be born .	Jumby want birth .
8	It was a difficult and long delivery .	The delivery was difficult and long .
9	It 's a shit , but it 's better than nothing , right ?	It 's a shit , but it 's better that nothing , right ?

### Seq2Seq model

Metoda seq2seq polega na trenowaniu modelu sieci neuronowej, aby przekształcić sekwencje z jednej dziedziny (zdania oryginalne) na sekwencje w innej dziedzinie (parafrazowane zdania).

Model ten zwykle składa się z dwóch głównych komponentów: enkodera i dekodera. Enkoder przetwarza sekwencję wejściową i kompresuje informacje do wektora kontekstowego. Dekoder następnie używa tego wektora do wygenerowania sekwencji wyjściowej.

Stworzyliśmy model zawierający kilka warstw neuronowych: Embedding, GRU i Dense.



## Trenowanie modelu runda 1

```
# Model Parameters

vocab_size = len(tokenizer.word_index) + 1 # Plus 1 for padding

embedding_dim = 256

rnn_units = 1024

# Building the Model

model = tf.keras.Sequential([

    tf.keras.layers.Embedding(vocab_size, embedding_dim, mask_zero=True),

    tf.keras.layers.GRU(rnn_units, return_sequences=True),

    tf.keras.layers.Dense(vocab_size, activation='softmax')

])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy')

# Train the Model

model.fit(dataset, epochs=20)
```

```
input_text = "I 'm going to have a son"
generated_sentence = generate_similar_sentence(input_text, model, tokenizer, max_seq_length)
print("Generated sentence:", generated_sentence)
```

```
1/1 [=====] - 0s 28ms/step
Generated sentence: i am gonna to a baby . the the the the the the the the the the the the the the the the the the
```

```
input_text = "Sorry, i am late"
generated_sentence = generate_similar_sentence(input_text, model, tokenizer, max_seq_length)
print("Generated sentence:", generated_sentence)
```

[illegible]

```
input_text = "Are you in the army?"
generated_sentence = generate_similar_sentence(input_text, model, tokenizer, max_seq_length)
print("Generated sentence:", generated_sentence)
```

```
1/1 [=====] - 0s 21ms/step  
Generated sentence: you're in trouble the the the the the the the the the the the the the the the the the
```

```
input_text = "Are you in the army?"
generated_sentence = generate_similar_sentence(input_text, model, tokenizer, max_seq_length)
print("Generated sentence:", generated_sentence)
```

[illegible]

Oprócz tekstu docelowego, model także generuje śmieci (the the the). Z tego powodu przyszliśmy do wniosku że model jest niedouczony.

Można jednak zauważyć, że model potrafi wydawać cechy zmyslenia, np

“Sorry, i am late” => “i am too”

*“Are you in the army?” => “you’re in trouble”*

*"Are you in the army?" => "you you pain pain"*

Chociaż tak naprawdę modeli nie potrafią myśleć.

## Trenowanie modelu runda 2

Podczas próby trochę zmieniliśmy architekturę modelu, dodając warstwę Bidirectional, zwiększyliśmy liczbę rekordów dla uczenia do 10000, oraz zwiększyliśmy liczbę epoch.

```
# Model with more complexity and regularization

model = tf.keras.Sequential([

    Embedding(vocab_size, embedding_dim, mask_zero=True),

    Bidirectional(GRU(rnn_units, return_sequences=True,
recurrent_dropout=0.2)),

    Dropout(0.2),

    Dense(vocab_size, activation='softmax')

])

# Early stopping to prevent overfitting

early_stopping = EarlyStopping(monitor='loss', patience=3)

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy')

# Train with early stopping

model.fit(dataset, epochs=50, callbacks=[early_stopping])
```

Trenowanie tego modelu zajęło 22898 sekund, czyli około 6 godzin.

### Run

22898.5s - GPU T4 x2

### Wyniki:

```
input_text = "i am at the bridge"
generated_sentence = generate_similar_sentence(input_text, model, tokenizer, max_seq_length)
print("Generated sentence:", generated_sentence)
```

```
1/1 [=====] - 0s 54ms/step
Generated sentence: i am on the bridge
```

```
input_text = "I 'm going to have a son"
generated_sentence = generate_similar_sentence(input_text, model, tokenizer, max_seq_length)
print("Generated sentence:", generated_sentence)
```

```
1/1 [=====] - 0s 54ms/step
Generated sentence: i gonna gonna have son son
```

```
input_text = "Are you in army"
generated_sentence = generate_similar_sentence(input_text, model, tokenizer, max_seq_length)
print("Generated sentence:", generated_sentence)
```

```
1/1 [=====] - 0s 54ms/step
Generated sentence: are you in military
```

Drugi model już nie generuje śmieci, i daje dobre wyniki, które są dobrymi parafrazami, semantycznie i syntaktycznie

*"Are you in the army" => "are you in military"*

*"i am at the bridge" => "i am on the bridge"*

Jednak są jeszcze drobne problemy ze zrozumieniem poszczególnych słów, kiedy model potrafi wygenerować zdanie, które jest semantycznie podobne, ale gramatycznie niepoprawne.

*"I'm going to have a son" => "i gonna gonna have son son"*

Notebook pierwszego modelu jest dostępny [tutaj](#)

Notebook drugiego modelu jest dostępny [tutaj](#).

## V. Wyniki

Wyniki przeprowadzonej pracy nad porównaniem różnych metod identyfikacji podobnych tekstów oferują głębsze zrozumienie skuteczności analizy semantycznej w tym kontekście. W obszarze "Wyszukiwania Podobnych Zdań z Bazy Danych" zaobserwowano poprawę efektywności dzięki zastosowaniu pre-processing'u tekstu, co wpłynęło na bardziej precyzyjne i adekwatne wyniki. PCA również miało wpływ na wyniki, poprawiając wartości dla mało podobnych danych.

W kontekście "Wyliczania Stopnia Podobieństwa Dwóch Podanych Tekstów" pomyślnie znaleźliśmy podobieństwo zdań, używając metody, oparte na koncepcji "Word Embeddings". Jednak same metody "Word Embeddings" nie nadają się do skutecznej generacji tekstów.

Chociaż metoda GloVe powszechnie uważa się za bardziej zaawansowaną niż Word2Vec, nie zauważyliśmy znaczącej różnicy. To może być związane z charakterem użytych danych.

Przewaga GloVe w wykorzystaniu globalnych statystyk korpusowych staje się bardziej wyraźna w przypadku większych i bardziej zróżnicowanych zbiorów danych. W przypadku zbioru danych z krótkimi zdaniami, okazało się że dane są niewystarczająco zróżnicowane lub niewystarczająco duże, aby w pełni skorzystać z podejścia GloVe do przechwytywania globalnych informacji o współwystępowaniu słów.

Dla metody FastText, wykorzystany był gotowy model nauczony na danych z Wikipedii, a nie na naszych danych - recenzji filmów. To sprawiło, że model nie potrafił zrozumieć badany kontekst.

Aby osiągnąć optymalną konfigurację modelu seq2seq do generacji tekstów, kluczowe jest przeprowadzenie serii eksperymentów. W naszym przypadku, po kilku próbach z różnymi ustawieniami, stwierdziliśmy, że liczba epok oraz wielkość zbioru danych uczących mają znaczący wpływ na jakość wygenerowanych tekstów.

Natomiast w obszarze "Generacja podobnych sentencji" zastosowanie modeli generatywnych z poprzednich punktów przyczyniło się do tworzenia bardziej spójnych i kontekstowo zgodnych tekstów.

Wyniki te stanowią podstawę do dalszych badań nad optymalizacją narzędzi do identyfikacji podobieństw tekstowych, wnosząc nowe spojrzenie na potencjał różnych podejść w dziedzinie Przetwarzania Języka Naturalnego.

## **VI. Podsumowanie**

Podsumowując, praca nad porównaniem różnych metod identyfikacji podobnych tekstów przyniosła istotne rezultaty. Wprowadzenie wstępnego przetwarzania tekstu znacząco poprawiło skuteczność wyszukiwania podobnych zdań z bazy danych, wyliczania stopnia podobieństwa dwóch tekstów oraz generacji tekstu podobnego do podanego. Wyniki potwierdzają, że zaawansowane techniki przetwarzania języka naturalnego są różne i mają różne zastosowanie. Natomiast nie możemy zdecydowanie stwierdzić, że jakaś jedna metoda jest lepsza lub gorsza, bo wyniki zależą od rozważanego problemu oraz od jakości używanych danych uczących. Te wnioski stanowią solidną podstawę do dalszego rozwijania narzędzi w dziedzinie Przetwarzania Języka Naturalnego.

## VII. Bibliografia

- [Text Analytics with Python: A Practitioner's Guide to Natural Language Processing 2nd edition by Dipanjan Sarkar](#)
- gensim documentation (<https://radimrehurek.com/gensim/models/word2vec.html>)
- Official text generation tutorial from [TensorFlow](#)
- dataset [Opusparcus: Open Subtitles Paraphrase Corpus for Six Languages](#)
- dataset [Rotten Tomatoes](#)
- GloVe [paper](#)
- [FastText documentation](#)