



Systemy rekomendacyjne

Content-based recommendations

User profile



Name: Jenny Wilson

Experience: Senior JavaScript Developer

Skills: JavaScript Node JS AWS
Kubernetes Go Lang

Job posts DB



Java Developer



Ruby Developer



JavaScript Developer



Senior JavaScript Developer

✓ Match



JavaScript Developer

mobidev

NLP-based recommendations

User profile



Name: Jenny Wilson

Experience: Senior JavaScript Developer

Skills: JavaScript Node JS AWS

Kubernetes Go Lang

Description:

I've been working in healthcare industry for over 5 years now.

Jenny specializes at developing software for hospitals, medical institution and clinics. And he does a pretty good job.

Job posts DB



Java Developer



Ruby Developer



JavaScript Developer

Looking for a JavaScript developer that has at least 3 years experience in healthcare domain



Senior JavaScript Developer

healthcare industry

5 years

hospitals, medical

institution and clinics

✓ Match



JavaScript Developer

JavaScript
3 years

healthcare

mobidev

Baza danych – z dzisiejszego przykładu

Data Science and AI Jobs Indeed

<https://www.kaggle.com/datasets/srivnaman/data-science-and-ai-jobs-indeed/data>

Index	Title	Location	Summary
1	Computer Vision Engineer	Hyderabad, Telangana	1+ years of experience is required in Computer Vision and Artificial Intelligence. Must be willing to relocate to Hyderabad. Total work: 1 year (Required).
2	Artificial Intelligence Internship	Thiruvananthapuram, Kerala	Supervised Learning in AI and ML. Unsupervised Learning in AI and M. Job Types: Full-time, Internship. Total work: 1 year (Preferred).

Podejście 1

Podobieństwo
tekstów

Vacancy summary	About the candidate	Similarity score <small>Text Similarity pre-trained model: Roberta large</small>
Should have 5-10 years experience. Should have very good experience with Automation using Python + Selenium + Panda (module) + Pytest	I'm experienced in Computer Vision	0.3
	I'm experienced in creating AI models using Python	0.4
	I have 10 years experience in Computer Vision	0.44
	I have 10 years experience in Python	0.59
	I have 10 years experience in creating AI models using Python. On top of that, I was working with Pytest and Selenium.	0.64

Zalety modeli podobieństwa tekstów

- Mogą być używane w formie surowej, zapewniając dokładniejsze wyniki.
- Wskazują, czy dwa teksty opisują to samo, bazując na analizie semantycznej.
- Mogą być stosowane w połączeniu z innymi modelami NLP, np. do streszczenia tekstu.



Czym jest podobieństwo tekstu (text similarity) w NLP?

Definicja

Podobieństwo tekstu odnosi się do procesu oceny, jak bardzo dwa teksty są podobne do siebie w kontekście znaczenia lub struktury. Jest to kluczowy element w dziedzinie przetwarzania języka naturalnego (NLP), który pozwala na analizę i porównywanie dokumentów, zdań lub fraz.

Zastosowania podobieństwa tekstu

- **Wykrywanie plagiatu:** Podobieństwo tekstu umożliwia identyfikację plagiatów poprzez porównanie tekstu z innymi znanymi tekstami.
- **Klasyfikacja dokumentów:** Możliwość klasyfikowania dokumentów na podstawie ich treści, np. określenie, czy dokument dotyczy określonego tematu.
- **Wyszukiwanie informacji:** Używane do identyfikacji relewantnych dokumentów w wyszukiwarkach lub bazach danych na podstawie podobieństwa tekstowego.
- **Tłumaczenie języków:** Pomaga w zwiększeniu dokładności systemów tłumaczeń maszynowych przez porównanie tłumaczenia z oryginalnym tekstem.
- **Analiza sentymentu:** Umożliwia identyfikację sentymentu tekstu poprzez porównanie go z zestawem tekstów o znanym sentymencie.
- **Sumaryzacja:** Podobieństwo tekstu może być wykorzystane do streszczania dokumentów poprzez identyfikację najważniejszych zdań i fraz.

Metody mierzenia podobieństw tekstów

01

Cosine Similarity

Obliczanie cosinusa kąta między ich reprezentacjami wektorowymi.

02

Levenshtein Distance

Określa minimalną liczbę operacji (wstawienia, usunięcia, zamiany), wymaganych do przekształcenia jednego tekstu w drugi.

03

Jaccard Index

Obliczany jako iloraz wielkości przecięcia dwóch zbiorów i wielkości ich sumy.

04

Euclidean Distance

Mierzenie najprostszej odległości między dwoma punktami (wektorami tekstów) w przestrzeni euklidesowej.

05

Hamming Distance

Określa liczbę pozycji, na których odpowiadające sobie znaki tych ciągów są różne.

06

Word Embeddings

Techniki reprezentowania słów w postaci gęstych wektorów, które odzwierciedlają kontekstowe znaczenie słów.

07

Pre-trained Language Models

Wytrenowane na dużych zbiorach danych tekstowych, aby zrozumieć język na poziomie głębokim.

01

Cosine similarity (Podobieństwo kosinusowe) definicja

Podobieństwo kosinusowe to miara podobieństwa między dwoma wektorami w przestrzeni wielowymiarowej, obliczana jako iloczyn skalarny tych wektorów podzielony przez iloczyn ich długości (norm). Wartość ta mieści się w zakresie od -1 do 1, gdzie 1 oznacza identyczność wektorów, 0 brak korelacji, a -1 całkowite przeciwieństwo. Jest często używana do mierzenia podobieństwa semantycznego w przetwarzaniu języka naturalnego (NLP), rekomendacji produktów, czy analizie danych, ponieważ skutecznie ignoruje różnice w wielkości wektorów, koncentrując się na ich kierunku.

01

Cosine similarity (Podobieństwo kosinusowe) przykład użycia

```
# pip install scikit-learn
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Przykładowe polskie zdania
zдания = ["Koty lubią mleko", "Psy lubią wodę"]

# Inicjalizacja wektoryzera TF-IDF
vectorizer = TfidfVectorizer()

# Przetworzenie tekstów i transformacja na wektory TF-IDF
tfidf_matrix = vectorizer.fit_transform(zдания)

# Obliczenie podobieństwa kosinusowego między wektorami
similarity_matrix = cosine_similarity(tfidf_matrix)

# Podobieństwo między pierwszym a drugim zdaniem
similarity_score = similarity_matrix[0, 1]

print(f"Podobieństwo kosinusowe między tekstami: {similarity_score}")
```

Output:

Podobieństwo kosinusowe
między tekstami:
0.20199309249791833

02

Levenshtein Distance (Odległość Levenshteina) definicja

Miara różnicy między dwoma ciągami tekstowymi, określająca minimalną liczbę operacji (wstawienie, usunięcie, zamiana znaku), potrzebnych do przekształcenia jednego tekstu w drugi. Jest szeroko stosowana w analizie podobieństwa tekstu, np. do wykrywania plagiatu czy korekty pisowni, gdzie mniejsza odległość wskazuje na większe podobieństwo między tekstami.

02

Levenshtein Distance (Odległość Levenshteina) działanie

		h	e	l	l	o
	?	?				
k	?					
e						
l						
m						

Zaczynamy od ustawień początkowych

02

Levenshtein Distance (Odległość Levenshteina) działanie

		h
	0	1
k	1	

Jeśli porównujemy dwie litery które są różne to z obszaru wybieramy minimum i zwiększamy je o jeden.

Jeśli porównujemy dwie takie same litery to wybieramy minimum.

02

Levenshtein Distance (Odległość Levenshteina) działanie

		h	e	l	l	o
	0	1	2	3	4	5
k	1	1				
e	2					
l	3					
m	4					

02

Levenshtein Distance (Odległość Levenshteina) działanie

		h	e	l	l	o
	0	1	2	3	4	5
k	1	1	2	3	4	5
e	2					
l	3					
m	4					

02

Levenshtein Distance (Odległość Levenshteina) działanie

		h	e	l	l	o
	0	1	2	3	4	5
k	1	1	2	3	4	5
e	2					
l	3					
m	4					

02

Levenshtein Distance (Odległość Levenshteina) działanie

		h	e	l	l	o
	0	1	2	3	4	5
k	1	1	2	3	4	5
e	2	2				
l	3					
m	4					

02

Levenshtein Distance (Odległość Levenshteina) działanie

		h	e	l	l	o
	0	1	2	3	4	5
k	1	1	2	3	4	5
e	2	2				
l	3					
m	4					

02

Levenshtein Distance (Odległość Levenshteina) działanie

		h	e	l	l	o
	0	1	2	3	4	5
k	1	1	2	3	4	5
e	2	2	1			
l	3					
m	4					

02

Levenshtein Distance (Odległość Levenshteina) działanie

		h	e	l	l	o
	0	1	2	3	4	5
k	1	1	2	3	4	5
e	2	2	1	2	3	4
l	3					
m	4					

02

Levenshtein Distance (Odległość Levenshteina) działanie

		h	e	l	l	o
	0	1	2	3	4	5
k	1	1	2	3	4	5
e	2	2	1	2	3	4
l	3	3	2	1	2	2
m	4	4	3	2	2	3

02

Levenshtein Distance (Odległość Levenshteina)

przykład

		s	e	t	t	i	n	g
	0	1	2	3	4	5	6	7
s	1	0	1	2	3	4	5	6
i	2	1	1	2	3	3	4	5
t	3	2	2	1	2	3	4	5
t	4	3	3	2	1	2	3	4
m	5	4	4	3	2	2	3	4
g	6	5	5	4	3	3	3	3

02

Levenshtein Distance (Odległość Levenshteina) przykład użycia

```
def levenshtein_distance(s1, s2):
    if len(s1) < len(s2):
        return levenshtein_distance(s2, s1)

    if len(s2) == 0:
        return len(s1)

    previous_row = range(len(s2) + 1)
    for i, c1 in enumerate(s1):
        current_row = [i + 1]
        for j, c2 in enumerate(s2):
            insertions = previous_row[j + 1] + 1
            deletions = current_row[j] + 1
            substitutions = previous_row[j] + (c1 != c2)
            current_row.append(min(insertions, deletions, substitutions))
        previous_row = current_row

    return previous_row[-1]

# Przykładowe wywołanie
word1 = "kot"
word2 = "kotek"
distance = levenshtein_distance(word1, word2)
print(f"Odległość Levenshteina między '{word1}' a '{word2}' wynosi: {distance}")
```

Output:

Odległość Levenshteina między
'kot' a 'kotek' wynosi: 2

03

Jaccard Index (Indeks Jaccarda) definicja

Indeks Jaccarda, znany również jako współczynnik podobieństwa Jaccarda, określa podobieństwo między dwoma zbiorami jako stosunek wielkości ich przecięcia do wielkości ich sumy. Wartość indeksu mieści się w zakresie od 0 do 1, gdzie 0 oznacza brak wspólnych elementów, a 1 pełną identyczność zbiorów. Jest on szczególnie przydatny w analizie tekstów, do porównywania zbiorów słów z dokumentów, gdzie istotna jest obecność słów, a nie ich częstotliwość czy kolejność. Indeks Jaccarda znajduje zastosowanie w wyszukiwaniu informacji, data miningu i rozpoznawaniu wzorców.

03

Jaccard Index (Indeks Jaccarda) przykład użycia

```
def jaccard_index(set1, set2):  
    # Oblicz przecięcie zbiorów  
    intersection = len(set(set1) & set(set2))  
    # Oblicz sumę zbiorów  
    union = len(set(set1) | set(set2))  
    # Oblicz indeks Jaccarda  
    jaccard = intersection / union  
    return jaccard
```

```
# Przykładowe zbiory słów
```

```
set1 = ["kot", "lubi", "mleko"]
```

```
set2 = ["pies", "lubi", "mleko"]
```

```
# Oblicz i wydrukuj indeks Jaccarda
```

```
jaccard = jaccard_index(set1, set2)
```

```
print(f"Indeks Jaccarda: {jaccard}")
```

Output:

Indeks Jaccarda: 0.5

04

Euclidean Distance (Odległość euklidesowa)

Odległość euklidesowa jest podstawową miarą odległości w przestrzeni euklidesowej, określającą najkrótszą drogę między dwoma punktami. W kontekście podobieństwa tekstu, może być używana do obliczania różnicy między wektorami słów lub dokumentów, reprezentujących ich cechy. Jest to pierwiastek kwadratowy z sumy kwadratów różnic między odpowiadającymi sobie elementami dwóch wektorów. Ta metoda jest szczególnie przydatna w analizie skupień i klasyfikacji, gdzie odległości między punktami w przestrzeni cech określają ich podobieństwo.

04

Euclidean Distance (Odległość euklidesowa) przykład użycia

```
# pip install scikit-learn
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import euclidean_distances

# Teksty do porównania
text1 = "Koty lubią mleko"
text2 = "Psy lubią wodę"

# Przekształcanie tekstów na wektory przy użyciu "bag of words"
vectorizer = CountVectorizer()
vectors = vectorizer.fit_transform([text1, text2]).toarray()

# Obliczanie odległości euklidesowej między wektorami
distance = euclidean_distances(vectors[0].reshape(1, -1), vectors[1].reshape(1, -1))

print(f"Odległość euklidesowa między tekstami: {distance[0][0]}")
```

Output:

Odległość euklidesowa między
tekstami: 2.0

05

Hamming Distance (Odległość Hamminga) definicja

Odległość Hamminga mierzy liczbę pozycji, na których dwa ciągi znaków (o tej samej długości) różnią się od siebie. W kontekście analizy podobieństwa tekstowego, jest używana do oceny stopnia zgodności dwóch tekstów, zakładając, że mają one równą długość. Im mniejsza odległość Hamminga, tym większe podobieństwo między porównywanymi ciągami. Jest to szczególnie przydatne w zadaniach takich jak korekta błędów czy kodowanie i szyfrowanie danych.

05

Hamming Distance (Odległość Hamminga) przykład użycia

```
def hamming_distance(str1, str2):  
    # Sprawdzenie, czy ciągi mają tę samą długość  
    if len(str1) != len(str2):  
        raise ValueError("Ciągi muszą być tej samej długości")  
  
    # Odległość Hamminga jako liczby różnych znaków na odpowiadających sobie pozycjach  
    distance = sum(c1 != c2 for c1, c2 in zip(str1, str2))  
    return distance  
  
# Przykładowe ciągi znaków  
str1 = "antek"  
str2 = "korek"  
  
# Obliczenie i wyświetlenie odległości Hamminga  
distance = hamming_distance(str1, str2)  
print(f"Odległość Hamminga: {distance}")
```

Output:

Odległość Hamminga: 3

06

Word Embeddings definicja

Embedingi słów to technika mapowania słów na wektory liczbowe, pozwalająca na reprezentację znaczenia słów w przestrzeni wektorowej. Dzięki temu podobne semantycznie słowa znajdują się blisko siebie. Ta metoda jest wykorzystywana w przetwarzaniu języka naturalnego (NLP) do analizy tekstów, poprawy dokładności różnych zadań NLP, takich jak klasyfikacja tekstu czy tłumaczenie maszynowe, poprzez uchwycenie zależności semantycznych i syntaktycznych między słowami.

06

Word Embeddings (Osadzenia słów) przykład użycia

```
# pip install spacy
import spacy

# Pobierz i załaduj angielski model językowy
# python -m spacy download pl_core_news_sm
nlp = spacy.load('pl_core_news_sm')

# Przykład słów do analizy podobieństwa
word1 = nlp("korek")
word2 = nlp("kotek")

# Oblicz podobieństwo między dwoma słowami
similarity = word1.similarity(word2)
print(f"Podobieństwo między '{word1}' a '{word2}': {similarity}")
```

Output:

Podobieństwo między 'korek' a
'kotek': 0.5257930824586733

07

Pre-trained Language Models definicja

Wstępnie wytrenowane modele językowe to zaawansowane narzędzia AI, które nauczyły się rozumienia i generowania języka naturalnego dzięki treningowi na ogromnych zbiorach tekstowych. Umożliwiają one wykonywanie wielu zadań NLP, takich jak tłumaczenie, generowanie tekstu, czy analiza sentymentu, i mogą być dostosowywane do konkretnych zastosowań poprzez fine-tuning. Modele takie jak BERT czy GPT stanowią podstawę dla wielu aplikacji przetwarzających język naturalny.

07

Pre-trained Language Models przykład użycia

```
from transformers import AutoTokenizer, AutoModel
import torch

# Załadowanie polskiego modelu, na przykład HerBERTa
tokenizer = AutoTokenizer.from_pretrained('allegro/herbert-base-cased')
model = AutoModel.from_pretrained('allegro/herbert-base-cased')

# Przykładowe wyrażenie
wyrazenie1 = "korek"
wyrazenie2 = "kotek"

# Tokenizacja i kodowanie wyrażen
inputs1 = tokenizer(wyrazenie1, return_tensors="pt", padding=True, truncation=True)
inputs2 = tokenizer(wyrazenie2, return_tensors="pt", padding=True, truncation=True)

# Obliczenie osadzeń dla obu wyrażen
with torch.no_grad():
    outputs1 = model(**inputs1)
    outputs2 = model(**inputs2)

# Średnia z osadzeń tokenów jako reprezentacja wyrażen
sentence_embedding1 = outputs1.last_hidden_state.mean(dim=1)
sentence_embedding2 = outputs2.last_hidden_state.mean(dim=1)

cosine_similarity = torch.nn.CosineSimilarity(dim=1, eps=1e-6)
similarity_score = cosine_similarity(sentence_embedding1, sentence_embedding2)

print(f"Podobieństwo między wyrażeniem: '{wyrazenie1}' a '{wyrazenie2}': {similarity_score.item()}")
```

Output:

Podobieństwo między
wyrażeniem: 'korek' a 'kotek':
0.777510404586792

Podejście 2

NAMED
ENTITY
RECOGNITION

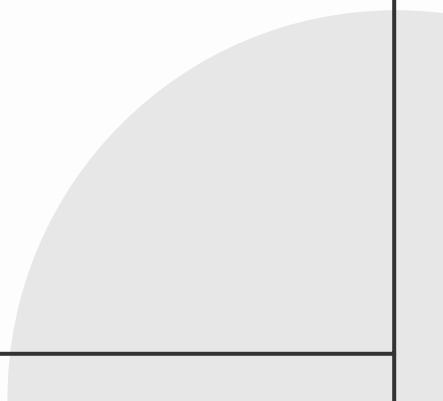
Vacancy location	About the candidate	Matching score
	Named Entity Recognition pre-trained model: Bert base	Matching method: TheFuzz
Hyderabad	I'm looking for a job in Hyderabad <small>LOC</small>	1
	I'm looking for a job in Hyderabad <small>LOC</small>	0.89
	I'm looking for a job in London <small>LOC</small>	0.13
	I'm looking for a remote job	0

Named Entity Recognition (NER)

Definicja: Proces identyfikacji i klasyfikacji nazwanych jednostek (osób, miejsc, organizacji itp.) w tekście.

Zastosowanie: Wykrywanie istotnych informacji w dużych zbiorach tekstu.

Jak działa NER?

- **Krok 1:** Przetwarzanie tekstu.
 - **Krok 2:** Identyfikacja jednostek nazwanych.
 - **Krok 3:** Klasyfikacja jednostek do odpowiednich kategorii (np. osoba, organizacja, miejsce).
- 
- A decorative gray curved shape is located in the bottom right corner of the slide, extending from the edge into the white space.

Kategorie jednostek nazwanych NER

- **Osoby:** Imiona i nazwiska ludzi (np. "Jan Kowalski").
- **Organizacje:** Nazwy firm, instytucji (np. "Google").
- **Miejsca:** Nazwy geograficzne, miasta, kraje (np. "Warszawa").
- **Daty:** Konkretnie daty i wyrażenia czasowe (np. "21 maja 2024").
- **Inne:** Produkty, wydarzenia, tytuły (np. "iPhone", "Igrzyska Olimpijskie").

Przykłady zastosowań NER

- **Analiza opinii:** Identyfikacja nazw firm w recenzjach produktów.
- **Ekstrakcja informacji:** Wydobywanie danych o osobach i miejscach z artykułów prasowych.
- **Systemy rekomendacyjne:** Wykrywanie preferencji użytkowników na podstawie ich zainteresowań.
- **Analiza finansowa:** Wyszukiwanie nazw firm i osób w raportach finansowych.

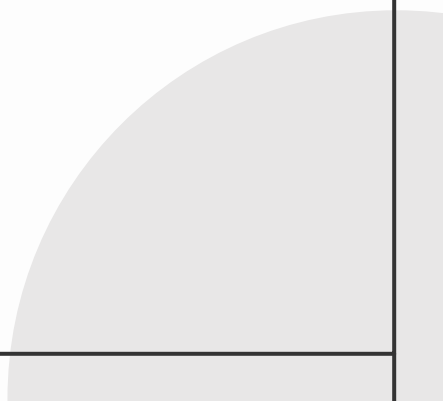
Zalety NER

- Umożliwia zastąpienie standardowych filtrów, takich jak wybór miasta czy miejsca pracy.
- Pozwala użytkownikom na podawanie zapytań w swobodnej formie.

Wyzwania w NER

- **Polisemiczność:** Jednostki nazwane mogą mieć różne znaczenia (np. "Apple" jako firma i jako owoc).
- **Wielojęzyczność:** Modele NER muszą być dostosowane do różnych języków.
- **Kontekst:** Jednostki nazwane mogą mieć różne klasyfikacje w zależności od kontekstu.

Przyszłość NER

- **Lepsze modele językowe:** Udoskonalenie modeli głębokiego uczenia (np. BERT, GPT-3).
 - **Integracja z AI:** Połączenie NER z innymi technologiami AI dla bardziej kompleksowej analizy.
 - **Przetwarzanie wielojęzyczne:** Tworzenie bardziej uniwersalnych modeli dla różnych języków.
- 

Implementacja NER z użyciem Spacy

```
import spacy

# Ładowanie modelu NER
nlp = spacy.load("en_core_web_sm")

# Przykładowy tekst
text = "Apple is looking at buying U.K. startup for $1 billion."

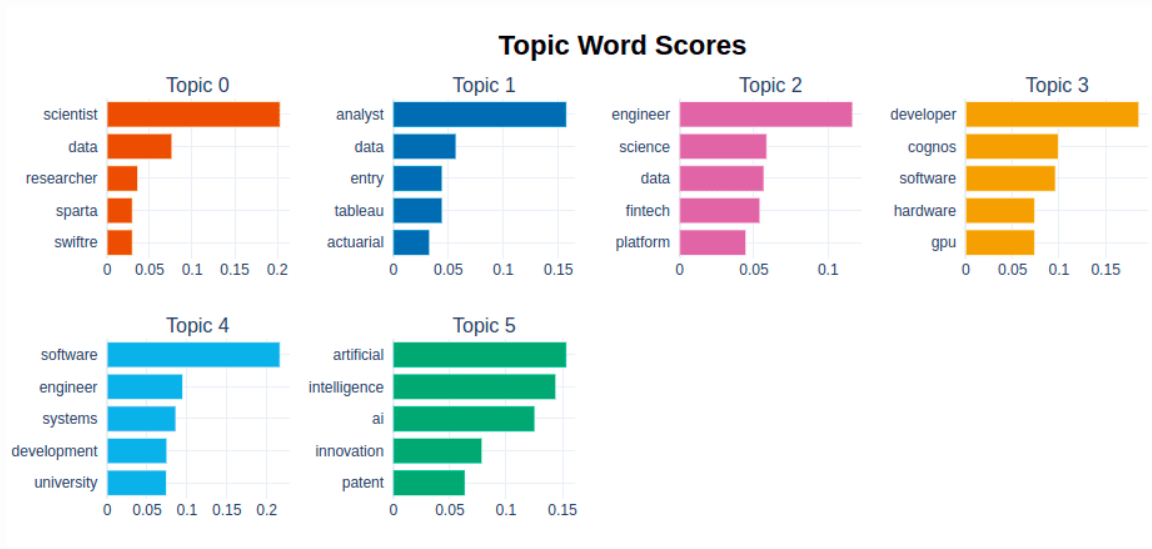
# Przetwarzanie tekstu
doc = nlp(text)

# Wypisanie jednostek nazwanych
for ent in doc.ents:
    print(ent.text, ent.label_)
```

Output:
Apple ORG
U.K. GPE
\$1 billion MONEY





Podejście 3

Topic
Extraction




Podejście 3

Topic
Extraction

Vacancy title	Found topics <small>Topic Extraction method: BERTopic</small>	About the candidate	Found topics <small>Topic Extraction method: BERTopic</small>	Matching score <small>Matching method: percent of intersection</small>
Data Scientist		I'm looking for a Data Engineer position		0.33
		I'm looking for a Data Analyst position		0.33
		I'm looking for a Data Analyst position. In addition to that, I have experience in Data Engineering.		0.67


Podejście 4

Ekstrakcja Słów Kluczowych

Vacancy summary	About the candidate	Matching score
Keywords extraction method: Bert base		Matching method: simple comparison
Good familiarity with the Windows desktop environment and use of Word, Excel, IE, Firefox etc. are required. Perform execution and report results accurately.  Good familiarity with the Windows KEY desktop environment KEY and use of Word KEY, Excel KEY, IE, Firefox KEY etc. are required. Perform execution and report results accurately.	I have an experience with PyTorch, Tensorflow, and Keras.	0
	I have an experience with different operating systems: Windows, Ubuntu, MacOS.	1
	I have experience in office applications: Word, Excel, PowerPoint, etc.	2

Podejście 4

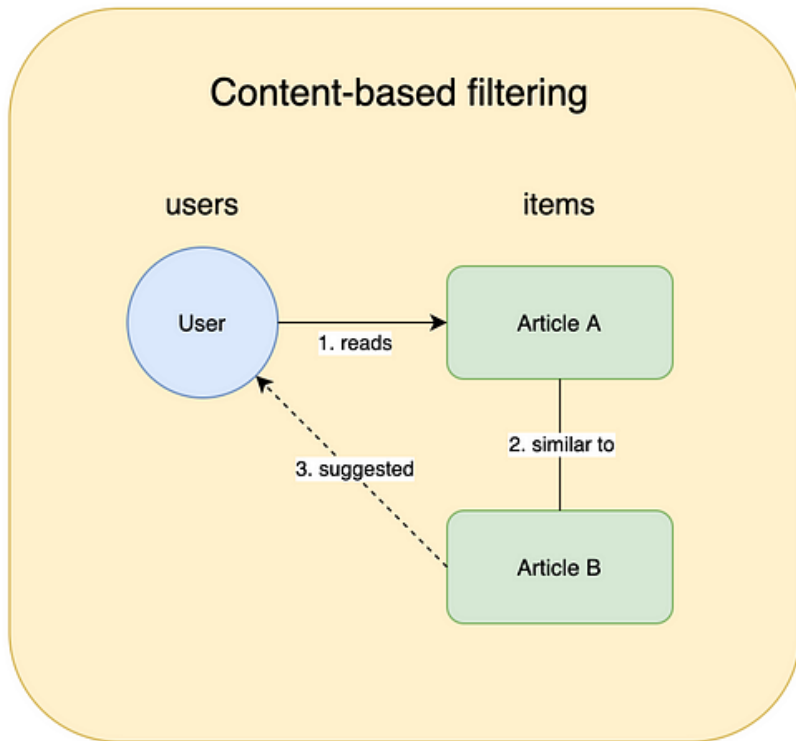
Ekstrakcja Słów Kluczowych

Vacancy summary	About the candidate	Matching score
Keywords extraction method: Bert base		Matching method: simple comparison
Good familiarity with the Windows desktop environment and use of Word, Excel, IE, Firefox etc. are required. Perform execution and report results accurately.  Good familiarity with the Windows KEY desktop environment KEY and use of Word KEY, Excel KEY, IE, Firefox KEY etc. are required. Perform execution and report results accurately.	I have an experience with PyTorch, Tensorflow, and Keras.	0
	I have an experience with different operating systems: Windows, Ubuntu, MacOS.	1
	I have experience in office applications: Word, Excel, PowerPoint, etc.	2



Rodzaje systemów rekomendacyjnych

Filtracja oparta na treści (Content-Based Filtering)



Filtracja oparta na treści (Content-Based Filtering)

Definicja: Metoda rekomendacji, która bazuje na opisach przedmiotów do rekomendacji.

Zastosowanie: Idealna w sytuacjach, gdy posiadamy dane o przedmiotach (np. nazwa, opis), ale nie o użytkownikach (np. ich wcześniej przeczytane artykuły).

Filtracja oparta na treści (Content-Based Filtering)

Podstawowa idea: Rekomendują przedmioty podobne do tych, które użytkownik lubił w przeszłości lub aktualnie przegląda.

Proces:

- Analiza cech przedmiotów (np. słowa kluczowe, tagi, opisy).
- Tworzenie profilu użytkownika na podstawie cech przedmiotów, które mu się podobały.
- Porównywanie profilu użytkownika z cechami innych przedmiotów w celu znalezienia podobieństw.

Filtracja oparta na treści (Content-Based Filtering)

Zalety:

- **Brak potrzeby danych o użytkownikach:** Nie wymaga listy przedmiotów, z którymi użytkownik wcześniej miał interakcje.
- **Efektywność od pierwszego dnia:** Działa efektywnie od pierwszego dnia użytkowania przez nowego użytkownika.
- **Personalizacja:** Rekomendacje są ściśle dopasowane do indywidualnych preferencji użytkownika.
- **Adaptacyjność:** System może łatwo dostosować się do zmian w preferencjach użytkownika.

Filtracja oparta na treści (Content-Based Filtering)

Przykład zastosowania:

- **Przykład 1:** Rekomendacje artykułów na stronie internetowej.
 - **Opis:** System analizuje treść artykułów, które użytkownik czytał, i rekomenduje podobne artykuły.
- **Przykład 2:** Rekomendacje produktów w sklepie internetowym.
 - **Opis:** System analizuje opisy produktów, które użytkownik przeglądał, i rekomenduje podobne produkty.

Filtracja oparta na treści (Content-Based Filtering)

Wyzwania i ograniczenia:

- **Ograniczona różnorodność:** System może rekomendować tylko przedmioty podobne do tych, które użytkownik już zna.
- **Problem z nowymi przedmiotami:** Nowe przedmioty bez szczegółowych opisów mogą nie być rekomendowane.
- **Czasochłonna analiza treści:** Wymaga szczegółowej analizy treści przedmiotów, co może być czasochłonne.

Filtracja oparta na treści (Content-Based Filtering)

Techniki:

- Analiza tekstu:
 - **TF-IDF**: Term Frequency-Inverse Document Frequency do ważenia słów.
 - **Word Embeddings**: Reprezentacje słów w przestrzeni wektorowej (np. Word2Vec, GloVe).
- Metody klasyfikacji:
 - **SVM (Support Vector Machines)**: Maszyny wektorów nośnych do klasyfikacji.
 - **Naive Bayes**: Prosty i skuteczny klasyfikator probabilistyczny.

Filtracja oparta na treści (Content-Based Filtering)

Porównanie z innymi metodami rekomendacji:

- Filtracja oparta na współpracy (Collaborative Filtering):
 - **Bazuje na:** Danych o interakcjach użytkowników z przedmiotami.
 - **Wymaga danych:** Od innych użytkowników, aby tworzyć rekomendacje.
- Filtracja hybrydowa (Hybrid Filtering):
 - **Łączy:** Metody filtracji opartej na treści i współpracy.
 - **Zalety:** Może korzystać z zalet obu metod, poprawiając jakość rekomendacji.

Filtracja oparta na treści (Content-Based Filtering)

Przyszłość filtracji opartej na treści:

- Rozwój technik NLP (Natural Language Processing):
 - **Transformers:** Nowoczesne modele do analizy tekstu (np. BERT, GPT).
 - **Generowanie treści:** Automatyczne generowanie opisów przedmiotów.
- Integracja z multimodalnymi danymi:
 - **Obrazy, wideo:** Analiza treści multimedialnych do lepszych rekomendacji.
- Personalizacja na większą skalę:
 - **Użycie Big Data:** Analiza dużych zbiorów danych do tworzenia dokładniejszych profili użytkowników.

Filtracja oparta na użytkownikach (User-Based Filtering)

Przykładowa implementacja:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Dane: opisy artykułów
documents = [
    "C++ is a programming language.",
    "Python is another programming language.",
    "Machine learning uses Python.",
    "JavaScript is popular for web development."
]

# Przetwarzanie tekstu
tfidf = TfidfVectorizer().fit_transform(documents)
cosine_similarities = cosine_similarity(tfidf, tfidf)

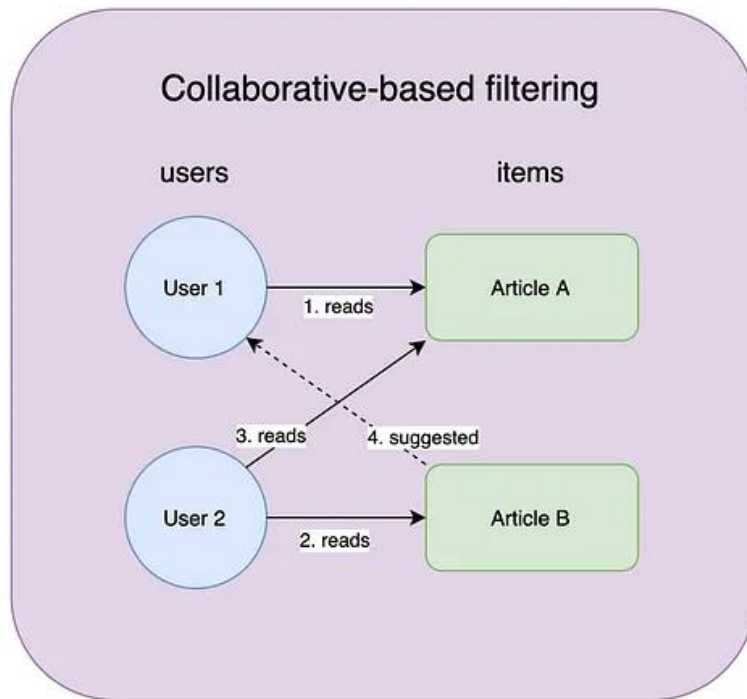
# Funkcja rekomendacji
def get_similarity_to_first_sentence(documents, cosine_similarities):
    first_sentence_similarities = cosine_similarities[0][1:] # Pomijamy podobieństwo do samego siebie
    return [(documents[i+1], first_sentence_similarities[i]) for i in range(len(first_sentence_similarities))]

# Przykład użycia
similarities_to_first_sentence = get_similarity_to_first_sentence(documents, cosine_similarities)
print(similarities_to_first_sentence)
```

Output:

```
[('Python is another programming language.',
  0.710233586557659),
 ('Machine learning uses Python.', 0.0),
 ('JavaScript is popular for web development.',
  0.13636943566562584)]
```

Filtracja Collaborative Filtering



Filtracja Collaborative Filtering

Definicja: Metoda rekomendacji, która bazuje na analizie zachowań wielu użytkowników.

Zastosowanie: Idealna w sytuacjach, gdy posiadamy dane o interakcjach użytkowników z przedmiotami (np. oceny, zakupy, przeglądane artykuły).

Filtracja Collaborative Filtering

Działanie algorytmów filtracji opartej na współpracy:

Podstawowa idea: Rekomendują przedmioty, które były lubiane przez innych użytkowników o podobnych upodobaniach.

Proces:

- Zbieranie danych o interakcjach użytkowników z przedmiotami.
- Analiza wzorców interakcji w celu znalezienia podobieństw między użytkownikami lub przedmiotami.
- Generowanie rekomendacji na podstawie tych podobieństw.

Filtracja Collaborative Filtering

Zalety:

- **Brak zależności od opisów przedmiotów:** Może dokładnie rekomendować złożone przedmioty, takie jak filmy, bez konieczności "rozumienia" samego przedmiotu.
- **Skalowalność:** Może być stosowana w dużych systemach z wieloma użytkownikami i przedmiotami.
- **Wydajność:** Podejście collaborative filtering często przewyższa filtrację opartą na treści, gdy dostępna jest wystarczająca ilość danych.

Filtracja Collaborative Filtering

Ograniczenia:

- **Cold start problem:** Wymaga znajomości listy przedmiotów, z którymi użytkownik wcześniej interagował, co może być problematyczne dla nowych użytkowników.
- **Brak wykorzystania technik NLP:** Ponieważ nie polega na cechach opisowych przedmiotów i użytkowników, techniki NLP nie mogą być zastosowane.
- **Rzadkość danych:** Niska liczba interakcji może prowadzić do słabych rekomendacji.
- **Obciążenie obliczeniowe:** Analiza dużych zbiorów danych może być czasochłonna i wymagająca pod względem zasobów.

Filtracja Collaborative Filtering

Rodzaje:

- **Filtracja użytkownik-użytkownik (User-User Collaborative Filtering):**
 - Rekomendacje oparte na podobieństwach między użytkownikami.
- **Filtracja przedmiot-przedmiot (Item-Item Collaborative Filtering):**
 - Rekomendacje oparte na podobieństwach między przedmiotami.
- **Filtracja hybrydowa (Hybrid Filtering):**
 - Łączy różne metody, aby poprawić jakość rekomendacji.

Filtracja Collaborative Filtering

Techniki:

- **Macierz użytkownik-przedmiot:**
 - Przechowuje dane o interakcjach użytkowników z przedmiotami.
- **Metody obliczania podobieństw:**
 - **Cosine Similarity:** Oblicza podobieństwo między użytkownikami lub przedmiotami.
 - **Pearson Correlation:** Mierzy korelację między ocenami użytkowników.
- **Algorytmy rekomendacji:**
 - **K-Nearest Neighbors (KNN):** Znajduje k najbliższych sąsiadów dla użytkownika lub przedmiotu.
 - **Matrix Factorization:** Techniki takie jak SVD (Singular Value Decomposition) do dekompozycji macierzy użytkownik-przedmiot.

Filtracja Collaborative Filtering

Porównanie z filtracją opartą na treści:

- **Dane wejściowe:**
 - **Collaborative Filtering:** Wymaga danych o interakcjach użytkowników z przedmiotami.
 - **Content-Based Filtering:** Wymaga danych o cechach przedmiotów.
- **Rekomendacje:**
 - **Collaborative Filtering:** Rekomenduje na podstawie podobieństw między użytkownikami.
 - **Content-Based Filtering:** Rekomenduje na podstawie cech przedmiotów.

Filtracja Collaborative Filtering

Przyszłość filtracji opartej na współpracy:

- **Użycie zaawansowanych algorytmów:**
 - **Deep Learning:** Modele neuronowe do analizy dużych zbiorów danych rekomendacyjnych.
 - **Graph-Based Models:** Modele oparte na grafach do analizy złożonych relacji między użytkownikami i przedmiotami.
- **Integracja z danymi kontekstowymi:**
 - **Lokalizacja, czas:** Uwzględnianie dodatkowych danych kontekstowych w rekomendacjach.
- **Zwiększenie personalizacji:**
 - **Dynamiczne profile użytkowników:** Aktualizowanie profili użytkowników w czasie rzeczywistym na podstawie ich bieżących interakcji.

Filtracja Collaborative Filtering

Przykładowa implementacja:

```
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

# Dane: macierz użytkownik-przedmiot
user_item_matrix = np.array([
    [4, 0, 0, 5, 1],
    [5, 5, 4, 0, 0],
    [0, 3, 0, 3, 4],
    [0, 4, 5, 0, 0],
])

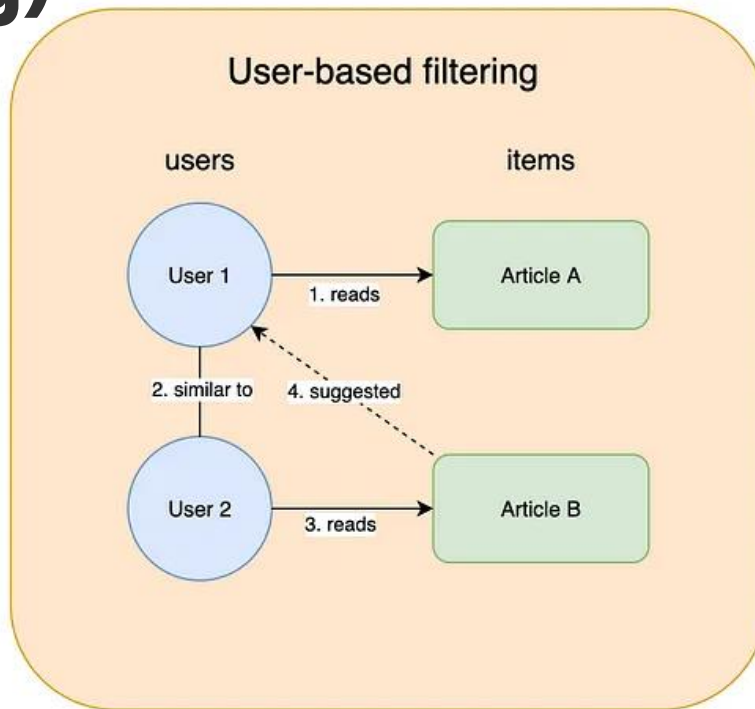
# Obliczanie podobieństw między użytkownikami
user_similarities = cosine_similarity(user_item_matrix)

# Funkcja rekomendacji
def get_user_recommendations(user_index, user_similarities, user_item_matrix):
    similar_users = user_similarities[user_index]
    weighted_ratings = np.dot(similar_users, user_item_matrix)
    recommendations = np.argsort(weighted_ratings)[::-1]
    return recommendations

# Przykład użycia funkcji rekomendacji
recommended_items = get_user_recommendations(0, user_similarities, user_item_matrix)
print(recommended_items)
```

Output:
[3 4 0 1 2]

Filtracja oparta na użytkownikach (User-Based Filtering)



Filtracja oparta na użytkownikach (User-Based Filtering)

Definicja: Metoda rekomendacji, która bazuje na podobieństwie między użytkownikami, aby rekomendować przedmioty.

Zastosowanie: Używana, gdy chcemy rekomendować przedmioty na podstawie tego, co inni użytkownicy o podobnych gustach lubili w przeszłości.

Filtracja oparta na użytkownikach (User-Based Filtering)

Podstawowa idea: Jeśli dwóch użytkowników miało podobne oceny lub interakcje z przedmiotami w przeszłości, będą mieli podobne preferencje w przyszłości.

Proces:

- Identyfikacja użytkowników o podobnych gustach.
- Analiza ocen i interakcji tych użytkowników.
- Rekomendowanie przedmiotów na podstawie tego, co podobni użytkownicy lubili.

Filtracja oparta na użytkownikach (User-Based Filtering)

Zalety:

- **Personalizacja:** Rekomendacje są dostosowane do indywidualnych preferencji użytkownika, bazując na podobieństwach do innych użytkowników.
- **Wysoka jakość rekomendacji:** Sprawdza się dobrze, gdy mamy dużo danych o użytkownikach.
- **Łatwość implementacji:** Prosta do wdrożenia przy użyciu podstawowych algorytmów.

Filtracja oparta na użytkownikach (User-Based Filtering)

Ograniczenia:

- **Cold start problem:** Problem rekomendacji dla nowych użytkowników bez historii interakcji.
- **Rzadkość danych:** Problem z rekomendacjami, gdy mamy niewiele danych o interakcjach użytkowników.
- **Skalowalność:** Może być wyzwaniem w systemach z dużą liczbą użytkowników i przedmiotów, ze względu na złożoność obliczeniową.
- **Nowy przedmiot:** nowym przedmiotom będzie brakować ocen, aby stworzyć solidny ranking.

Filtracja oparta na użytkownikach (User-Based Filtering)

Metody obliczania podobieństw między użytkownikami:

- **Cosine Similarity:** Oblicza podobieństwo między dwoma wektorami poprzez miarę kąta między nimi.
- **Pearson Correlation:** Mierzy siłę i kierunek związku liniowego między dwoma zmiennymi.
- **Jaccard Similarity:** Mierzy podobieństwo poprzez porównanie liczby wspólnych elementów do liczby wszystkich elementów.

Filtracja oparta na użytkownikach (User-Based Filtering)

Zastosowanie:

- **Serwisy streamingowe:** Rekomendowanie filmów na podstawie gustów użytkowników o podobnych preferencjach.
- **Sklepy internetowe:** Rekomendowanie produktów na podstawie historii zakupów użytkowników o podobnych upodobaniach.

Filtracja oparta na użytkownikach (User-Based Filtering)

Przyszłość filtracji opartej na użytkownikach:

- **Integracja z zaawansowanymi algorytmami:**
 - **Deep Learning:** Modele neuronowe do analizy dużych zbiorów danych rekomendacyjnych.
 - **Hybrid Models:** Łączenie różnych metod rekomendacji w celu poprawy dokładności.
- **Zwiększenie personalizacji:**
 - **Dynamiczne profile użytkowników:** Aktualizowanie profili użytkowników w czasie rzeczywistym na podstawie ich bieżących interakcji.
- **Wykorzystanie danych kontekstowych:**
 - **Lokalizacja, czas:** Uwzględnianie dodatkowych danych kontekstowych w rekomendacjach.

Filtracja oparta na użytkownikach (User-Based Filtering)

Przykładowa implementacja:

```
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

# Dane: macierz użytkownik-przedmiot
user_item_matrix = np.array([
    [3, 0, 0, 5, 5],
    [2, 5, 5, 3, 0],
    [2, 1, 0, 3, 0],
    [0, 4, 5, 3, 4],
])

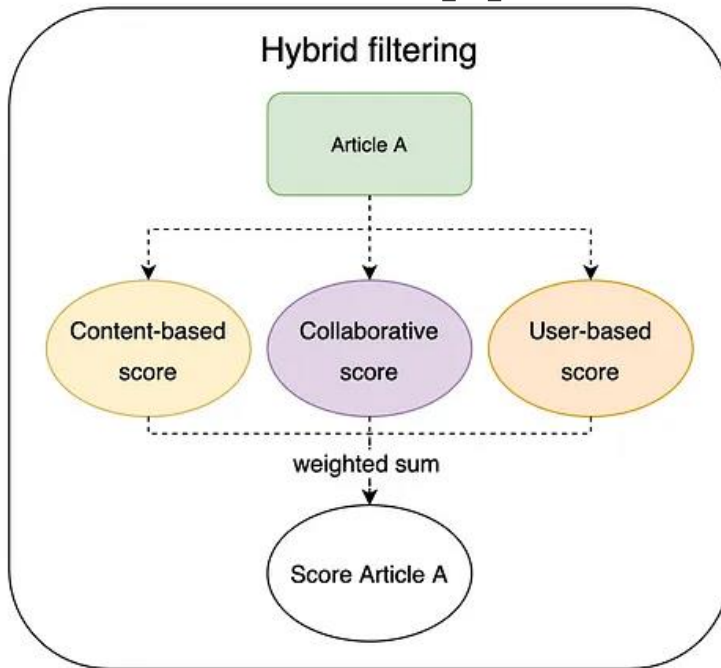
# Obliczanie podobieństw między użytkownikami
user_similarities = cosine_similarity(user_item_matrix)

# Funkcja rekomendacji
def get_user_recommendations(user_index, user_similarities, user_item_matrix):
    similar_users = user_similarities[user_index]
    weighted_ratings = np.dot(similar_users, user_item_matrix)
    recommendations = np.argsort(weighted_ratings)[::-1]
    return recommendations

# Przykład użycia funkcji rekomendacji
recommended_items = get_user_recommendations(0, user_similarities, user_item_matrix)
print(recommended_items)
```

Output:
[3 4 0 1 2]

Hybrydowe Podejścia do Rekomendacji (Hybrid Recommendation Approaches)



Hybrydowe Podejścia do Rekomendacji (Hybrid Recommendation Approaches)

Definicja: Metoda łącząca różne techniki rekomendacji, takie jak filtracja oparta na treści i filtracja oparta na współpracy, aby poprawić dokładność rekomendacji.

Zastosowanie: Idealna w sytuacjach, gdzie pojedyncze podejście nie daje zadowalających wyników.

Hybrydowe Podejścia do Rekomendacji (Hybrid Recommendation Approaches)

Zalety:

- **Lepsza dokładność:** Łącząc różne metody, można osiągnąć wyższe wskaźniki precyzji i trafności rekomendacji.
- **Redukcja ograniczeń:** Hybrydowe podejścia mogą zredukować ograniczenia poszczególnych metod, takie jak problem zimnego startu.
- **Personalizacja:** Lepsze dostosowanie rekomendacji do indywidualnych potrzeb użytkowników.

Hybrydowe Podejścia do Rekomendacji (Hybrid Recommendation Approaches)

Implementacja:

- Robienie osobnych prognoz i ich łączenie.
- Dodawanie możliwości content-based do podejścia collaborative (i vice versa).
- Jednoczenie podejść w jednym modelu.

Hybrydowe Podejścia do Rekomendacji (Hybrid Recommendation Approaches)

Typy hybrydowych podejść:

- **Podejście sekwencyjne:** Jedna metoda używana jest jako pierwsza, a druga do poprawienia wyników.
- **Podejście równoległe:** Obie metody działają równocześnie i ich wyniki są łączone.
- **Podejście kombinacyjne:** Rekomendacje z różnych metod są łączone w jeden wynik.

Hybrydowe Podejścia do Rekomendacji (Hybrid Recommendation Approaches)

Techniki hybrydyzacji:

1. **Weighted:** Łączenie wyników różnych komponentów rekomendacyjnych numerycznie.
2. **Switching:** Wybór i zastosowanie jednego z komponentów rekomendacyjnych.
3. **Mixed:** Prezentowanie rekomendacji z różnych rekomenderów razem.
4. **Feature Combination:** Łączenie cech pochodzących z różnych źródeł wiedzy i podanie ich jednemu algorytmowi rekomendacyjnemu.
5. **Feature Augmentation:** Obliczanie cechy lub zestawu cech, które są częścią wejścia dla kolejnej techniki.
6. **Cascade:** Nadawanie rekomenderom priorytetów, z niższymi priorytetami rozstrzygającymi remisy w ocenach wyższych priorytetów.
7. **Meta-level:** Jedna technika rekomendacyjna tworzy model, który jest wejściem dla kolejnej techniki.

Hybrydowe Podejścia do Rekomendacji (Hybrid Recommendation Approaches)

Przykłady:

- **Filtracja oparta na treści + filtracja oparta na współpracy:** Łączenie cech przedmiotów z historią interakcji użytkowników.
- **Filtracja oparte na współpracy + demograficzne podejście:** Uwzględnienie danych demograficznych użytkowników w celu poprawy rekomendacji.
- **Filtracja oparta na współpracy + kontekstowe podejście:** Integracja danych kontekstowych, takich jak lokalizacja i czas.

Hybrydowe Podejścia do Rekomendacji (Hybrid Recommendation Approaches)

Przykładowa implementacja cz. 1:

```
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np

# Dane: macierz użytkownik-przedmiot i opisy przedmiotów
user_item_matrix = np.array([
    [4, 0, 0, 5, 1],
    [5, 5, 4, 0, 0],
    [0, 3, 0, 3, 4],
    [0, 4, 5, 0, 0],
])

item_descriptions = [
    "Action movie with thrilling scenes.",
    "Romantic comedy with a love story.",
    "Science fiction with futuristic technology.",
    "Documentary about nature.",
    "Horror movie with scary moments."
]

# Filtracja oparta na współpracy
user_similarities = cosine_similarity(user_item_matrix)

# Filtracja oparta na treści
tfidf = TfidfVectorizer().fit_transform(item_descriptions)
content_similarities = cosine_similarity(tfidf, tfidf)
```

Hybrydowe Podejścia do Rekomendacji (Hybrid Recommendation Approaches)

Przykładowa implementacja cz. 2:

```
# Filtracja oparta na współpracy
user_similarities = cosine_similarity(user_item_matrix)

# Filtracja oparta na treści
tfidf = TfidfVectorizer().fit_transform(item_descriptions)
content_similarities = cosine_similarity(tfidf, tfidf)

# Funkcja rekomendacji hybrydowej z wagami
def weighted_hybrid_recommendations(user_index, user_similarities, content_similarities, user_item_matrix, alpha=0.7):
    similar_users = user_similarities[user_index]
    collaborative_scores = np.dot(similar_users, user_item_matrix)
    content_scores = content_similarities[user_index]

    hybrid_scores = alpha * collaborative_scores + (1 - alpha) * content_scores
    recommendations = np.argsort(hybrid_scores)[::-1]
    return recommendations

# Przykład użycia
recommended_items = weighted_hybrid_recommendations(0, user_similarities, content_similarities, user_item_matrix)
print(recommended_items)
```

Output:
[3 0 1 4 2]

Przykład

- [How Spotify's AI-Driven Recommendations Work | WSJ Tech Behind \(youtube.com\)](#)
- [Why Men Get So Few Matches on Dating Apps \(youtube.com\)](#)