Generatywna sztuczna inteligencja z dużymi modelami tekstowymi

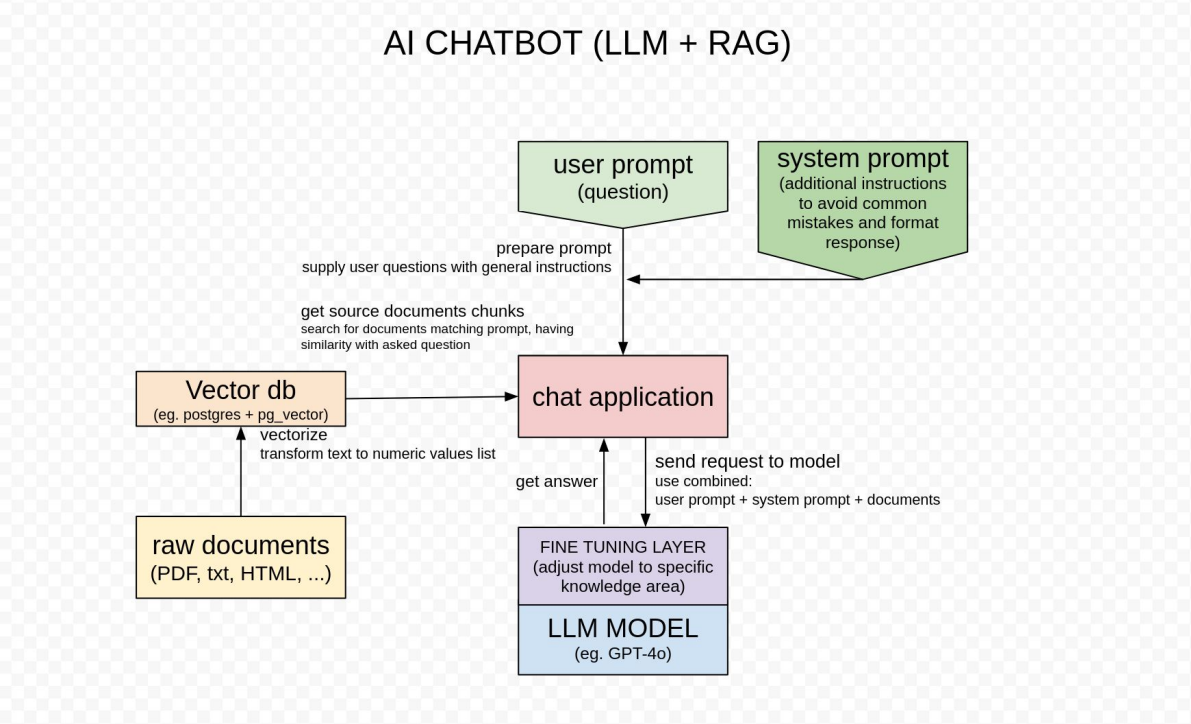
Retrieval Augmented Generation (RAG)

Michał Żarnecki

LLM - chaining

LLM chaining to proces łączenia dużych modeli językowych z innymi aplikacjami w celu uzyskania jak najlepszego wyniku z danego wejścia.
Łączenie LLM z dodatkowymi komponentami umożliwia tworzenie bogatych w funkcje generatywnych aplikacji AI, zwanych łańcuchami LLM, które są lepiej dostosowane do konkretnego przypadku użycia.
Przykładem łańcuchowania LLM jest praktyka łączenia wstępnie wyszkolonego modelu językowego z zewnętrznym źródłem danych w celu optymalizacji jego odpowiedzi, czyli retrieval augmented generation (RAG).

Retrieval Augmented Generation (RAG)



Algorytmy dopasowania

W jaki sposób porównać podobieństwo 2 tekstów?

Dystans Levenshteina

Dystans Levenshteina to miara różnicy między dwoma ciągami znaków (np. dwoma słowami lub zdaniami).
Określa minimalną liczbę operacji potrzebnych do przekształcenia jednego ciągu w drugi, gdzie dopuszczalne operacje to:

- Zamiana jednego znaku na inny.
- Usunięcie znaku.
- Wstawienie znaku.

Dystans Levenshteina jest używany m.in. do porównywania tekstów, w korekcie błędów, rozpoznawaniu mowy, czy przy wyszukiwaniu podobnych słów.

Dystans Levenshteina

Obliczanie dystansu Levenshteina można rozwiązać za pomocą algorytmu dynamicznego programowania. Zasadniczo tworzymy tablicę 2D, w której każda komórka (i,j)(i,j) reprezentuje dystans między pierwszymi ii znakami pierwszego ciągu a pierwszymi jj znakami drugiego ciągu.
Proces wypełniania tej tablicy jest wykonywany iteracyjnie na podstawie następujących zasad:

- Jeśli znaki są takie same, to dystans w komórce (i,j)(i,j) jest taki sam jak w komórce (i−1,j−1)(i−1,j−1) (czyli kopiujemy wartość z przekątnej).
- Jeśli znaki są różne, to wybieramy minimalny dystans spośród:
    - Usunięcia znaku (wartość z komórki (i−1,j)(i−1,j)) plus 1,
    - Wstawienia znaku (wartość z komórki (i,j−1)(i,j−1)) plus 1,
    - Zamiany znaku (wartość z komórki (i−1,j−1)(i−1,j−1)) plus 1.

Dystans Levenshteina

Policzmy dystans Levenshteina dla dwóch wyrazów: **"kolega"** i **"collegium"**

Krok 1:
Tworzymy tablicę, w której kolumny reprezentują litery z wyrazu "collegium", a wiersze - litery z wyrazu "kolega". Tablica będzie miała rozmiar 7×107 \times 107×10, bo długości wyrazów to odpowiednio 6 i 9, a dodatkowo wstawiamy wiersz i kolumnę na obsługę pustych ciągów.

		c	o	l	l	e	g	i	u	m
	0	1	2	3	4	5	6	7	8	9
k	1									
o	2									
l	3									
e	4									
g	5									
a	6									

Dystans Levenshteina

Krok 2: Wypełnienie wartości brzegowych

Zapełniamy pierwszy wiersz i pierwszą kolumnę wartościami reprezentującymi koszt wstawienia/usunięcia znaków:

		c	o	l	l	e	g	i	u	m
	0	1	2	3	4	5	6	7	8	9
k	1									
o	2									
l	3									
e	4									
g	5									
a	6									

Dystans Levenshteina

Krok 3: Wypełnianie tablicy

Teraz wypełnimy resztę tablicy, korzystając z zasad dystansu Levenshteina.

Porównujemy pierwsze litery
k z c: różne, więc wybieramy minimalną wartość spośród operacji: wstawienia, usunięcia lub zamiany, czyli
min(1+1,0+1,1+1) =1
min(1+1,0+1,1+1) = 1.
Dla kolejnych liter postępujemy analogicznie.

		c	o	l	l	e	g	i	u	m
	0	1	2	3	4	5	6	7	8	9
k	1	1	2	3	4	5	6	7	8	9
o	2	2	1	2	3	4	5	6	7	8
l	3	3	2	1	2	3	4	5	6	7
e	4	4	3	2	1	2	3	4	5	6
g	5	5	4	3	2	1	2	3	4	5
a	6	6	5	4	3	2	2	3	4	5

Dystans Levenshteina między słowami **"kolega"** a **"collegium"** wynosi **5**. To oznacza, że minimalna liczba operacji potrzebnych do przekształcenia słowa "kolega" w "collegium" to 5 operacji edycyjnych (zamian lub wstawień/usunięć liter).

Dystans Levenshteina

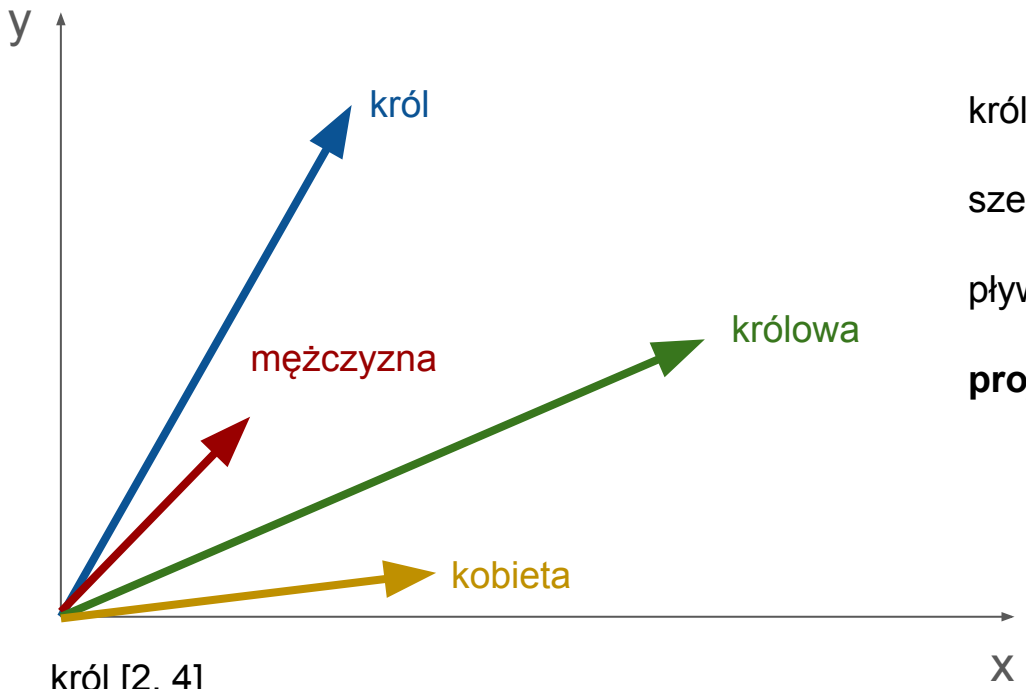
Zadanie:

Oblicz dystans Levenshteina pomiędzy słowami kot i krab.

Algorytmy dopasowania

Problem:
Dystans Levenshteina nie bierze pod uwagę znaczenia i zależności pomiędzy słowami.

Kodowanie tekstu w bazie do postaci wektorowej



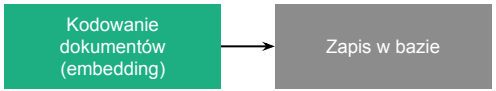
król [2, 4]
królowa [5, 3]
...

król - mężczyzna + kobieta ≈ królowa

szedłem - idę ≈ biegłem - biegnę

pływamy - my ≈ pływam - ja

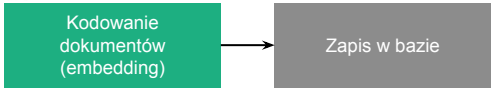
projekt + deadline ≈ stackoverflow + copy/paste



Kodowanie tekstu w bazie do postaci wektorowej

```
1 model['go']
```

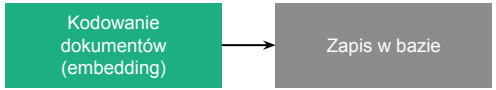
```
array([-0.078894,  0.4616  ,  0.57779 , -0.71637 , -0.13121 ,  0.4186  ,
       -0.29156 ,  0.52006 ,  0.089986, -0.35062 ,  0.51755 ,  0.51998 ,
        0.15218 ,  0.41485 , -0.12377 , -0.37222 ,  0.0273  ,  0.75673 ,
       -0.8739  ,  0.58935 ,  0.46662 ,  0.62918 ,  0.092603, -0.012868,
       -0.015169,  0.25567 , -0.43025 , -0.77668 ,  0.71449 , -0.3834  ,
       -0.69638 ,  0.23522 ,  0.11396 ,  0.02778 ,  0.071357,  0.87409 ,
       -0.1281  ,  0.063576,  0.067867, -0.50181 , -0.28523 , -0.072536,
       -0.50738 , -0.6914  , -0.53579 , -0.11361 , -0.38234 , -0.12414 ,
        0.011214, -1.1622  ,  0.037057, -0.18495 ,  0.01416 ,  0.87193 ,
       -0.097309, -2.3565  , -0.14554 ,  0.28275 ,  2.0053  ,  0.23439 ,
       -0.38298 ,  0.69539 , -0.44916 , -0.094157,  0.90527 ,  0.65764 ,
        0.27628 ,  0.30688 , -0.57781 , -0.22987 , -0.083043, -0.57236 ,
       -0.299   , -0.81112 ,  0.039752, -0.05681 , -0.48879 , -0.18091 ,
       -0.28152 , -0.20559 ,  0.4932  , -0.033999, -0.53139 , -0.28297 ,
       -1.4475  , -0.18685 ,  0.091177,  0.11454 , -0.28168 , -0.33565 ,
       -0.31663 , -0.1089  ,  0.10111 , -0.23737 , -0.64955 , -0.268   ,
        0.35096 ,  0.26352 ,  0.59397 ,  0.26741 ], dtype=float32)
```



Kodowanie tekstu w bazie do postaci wektorowej

```
1 model['away']
```

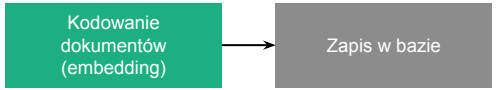
```
array([-0.10379 , -0.014792,  0.59933 , -0.51316 , -0.036463,  0.6588  ,
       -0.57906 ,  0.17819 ,  0.23663 , -0.21384 ,  0.55339 ,  0.53597 ,
        0.041444,  0.16095 ,  0.017093, -0.37242 ,  0.017974,  0.39268 ,
       -0.23265 ,  0.1818  ,  0.66405 ,  0.98163 ,  0.42339 ,  0.030581,
        0.35015 ,  0.25519 , -0.71182 , -0.42184 ,  0.13068 , -0.47452 ,
       -0.08175 ,  0.1574  , -0.13262 ,  0.22679 , -0.16885 , -0.11122 ,
       -0.32272 , -0.020978, -0.43345 ,  0.172   , -0.67366 , -0.79052 ,
        0.10556 , -0.4219  , -0.12385 , -0.063486, -0.17843 ,  0.56359 ,
        0.16986 , -0.17804 ,  0.13956 , -0.20169 ,  0.078985,  1.4497  ,
        0.23556 , -2.6014  , -0.5286  , -0.11636 ,  1.7184  ,  0.33254 ,
        0.12136 ,  1.1602  , -0.2914  ,  0.47125 ,  0.41869 ,  0.35271 ,
        0.47869 , -0.042281, -0.18294 ,  0.1796  , -0.24431 , -0.34042 ,
        0.20337 , -0.93676 ,  0.013077,  0.080339, -0.36604 , -0.44005 ,
       -0.35393 ,  0.15907 ,  0.55807 ,  0.1492  , -0.86433 ,  0.040305,
       -1.0939  , -0.26386 , -0.29494 ,  0.25696 , -0.33718 , -0.086468,
       -0.24246 , -0.21114 ,  0.099632,  0.12815 , -0.78714 , -0.51785 ,
       -0.10944 ,  0.9763  ,  0.57032 ,  0.13581 ], dtype=float32)
```



Kodowanie tekstu w bazie do postaci wektorowej

```
1 (model['go'] + model['away'])/2
```

```
array([-0.091342  ,  0.223404  ,  0.58856   , -0.614765  , -0.0838365 ,
        0.5387    , -0.43531   ,  0.349125  ,  0.163308  , -0.28223   ,
        0.53547   ,  0.52797496,  0.096812  ,  0.2879    , -0.0533385 ,
       -0.37232   ,  0.022637  ,  0.574705  , -0.553275  ,  0.385575  ,
        0.565335  ,  0.805405  ,  0.2579965 ,  0.0088565 ,  0.1674905 ,
        0.25543   , -0.571035  , -0.59926   ,  0.422585  , -0.42896   ,
       -0.389065  ,  0.19631   , -0.00933   ,  0.127285  , -0.0487465 ,
        0.381435  , -0.22540998,  0.021299  , -0.1827915 , -0.16490501,
       -0.47944498, -0.431528  , -0.20091   , -0.55665   , -0.32982   ,
       -0.088548  , -0.28038502,  0.219725  ,  0.090537  , -0.67012   ,
        0.0883085 , -0.19332   ,  0.0465725 ,  1.160815  ,  0.0691255 ,
       -2.47895   , -0.33707   ,  0.083195  ,  1.86185   ,  0.283465  ,
       -0.13081   ,  0.927795  , -0.37028   ,  0.1885465 ,  0.66198   ,
        0.505175  ,  0.37748498,  0.1322995 , -0.380375  , -0.025135  ,
       -0.1636765 , -0.45639   , -0.047815  , -0.87394   ,  0.0264145 ,
        0.0117645 , -0.427415  , -0.31048   , -0.317725  , -0.02326   ,
        0.525635  ,  0.05760051, -0.69786   , -0.1213325 , -1.2707    ,
       -0.225355  , -0.1018815 ,  0.18575001, -0.30943   , -0.211059  ,
       -0.279545  , -0.16002001,  0.100371  , -0.05461   , -0.71834505,
       -0.392925  ,  0.12075999,  0.61991   ,  0.582145  ,  0.20161   ],
      dtype=float32)
```



Baza wektorowa

Chroma

Pinecone

Faiss

drant

PostgreSQL + pg_vector

Baza wektorowa

vector_database.ipynb

{} Code | M↓ Markdown

```python
from sentence_transformers import SentenceTransformer, util

model = SentenceTransformer("all-MiniLM-L6-v2")
print(model.max_seq_length)

model.max_seq_length = 256

# Our sentences we like to encode
sentences = [
    "dinosaurs live in africa but in different time dimension",
    "this is sentence about little cat that liked to eat tomatoes",
    "this is the another sample sentence which is here just to not be matched while other one is"
]

# Sentences are encoded by calling model.encode()
embeddings = model.encode(sentences, normalize_embeddings=True)
```

✓ [2] 1m 56s

Baza wektorowa

vector_database.ipynb

perform search

```
1  queryText = "french fries"
2  embeddingSearch = model.encode([queryText], normalize_embeddings=True)
3  embeddingFound, idx = index.search(embeddingSearch, 1)  # actual search
4  print(queryText + " matches:\n" + sentences[idx[0][0]])
5
6  queryText = "not similar text"
7  embeddingSearch = model.encode([queryText], normalize_embeddings=True)
8  embeddingFound, idx = index.search(embeddingSearch, 1)  # actual search
9  print(queryText + " matches:\n" + sentences[idx[0][0]])
```

✓ [31] 32ms

```
french fries matches:
this is sentence about little cat that liked to eat tomatoes
not similar text matches:
this is the another sample sentence which is here just to not be matched while other one is
```

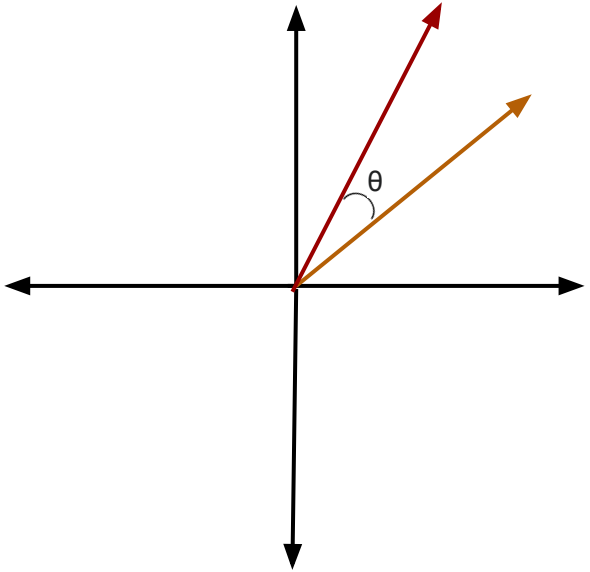
Baza wektorowa

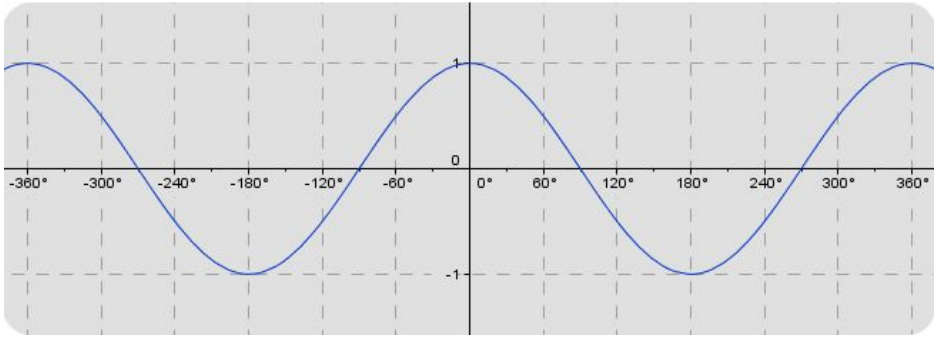
Zadanie:

Uruchom jupyter notebook vector_database.ipynb i przeanalizuj przykład wykorzystania bazy wektorowej

Załaduj do bazy wektorowej dowolny własny zbiór dokumentów (np. dokumentacja, artykuły, wiadomości). Następnie wyszukaj najbardziej pasujący artykuł do wybranej przez siebie frazy.

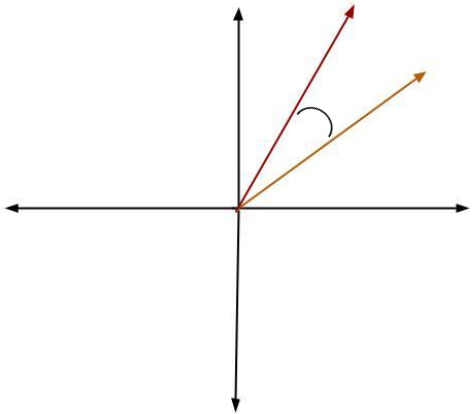
Podobieństwo cosinusowe



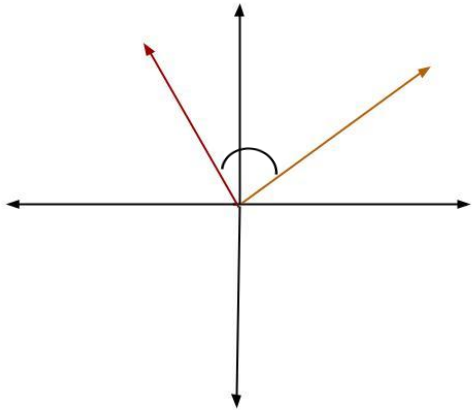


$$\cos\theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \cdot \|\vec{b}\|}$$

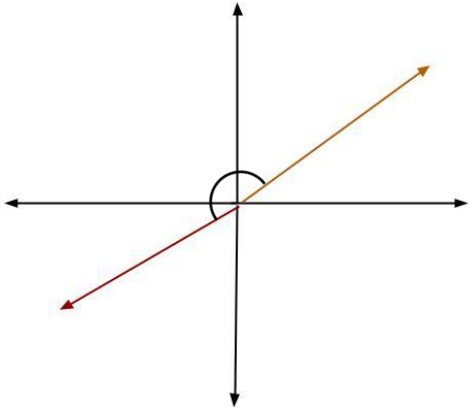
Dopasowanie "pasujących" dokumentów



kąt bliski 0°
cosinus kąta bliski 1



kąt bliski 90°
cosinus kąta bliski 0



kąt bliski 180°
cosinus kąta bliski -1

Algorytmy dopasowania

- **Euclidean Distance:** Euclidean distance is widely used in various fields, including image processing for measuring color similarity, computer vision for object recognition, and geographic information systems for calculating distances between coordinates.
- **Manhattan Distance (L1 Norm):** Manhattan distance is used in route planning and transportation, like finding the shortest path on a grid-based city map. It's also employed in feature selection for machine learning.
- **Minkowski Distance:** Minkowski distance, which generalizes both Euclidean and Manhattan distances, is applied in pattern recognition, image analysis, and clustering tasks.
- **Cosine Similarity:** Cosine similarity is widely used in information retrieval and natural language processing, including text document similarity, recommendation systems, and search engine ranking.
- **Jaccard Similarity:** Jaccard similarity is used for set-based data, such as document retrieval, plagiarism detection, and collaborative filtering in recommendation systems.
- **Hamming Distance:** Hamming distance is used in error detection and correction codes, network security for comparing binary strings, and genomics for comparing DNA sequences.
- **Levenshtein Distance (Edit Distance):** Levenshtein distance is applied in spell-checkers, string similarity for information retrieval, and data deduplication in databases.

https://crucialbits.com/blog/a-comprehensive-list-of-similarity-search-algorithms/

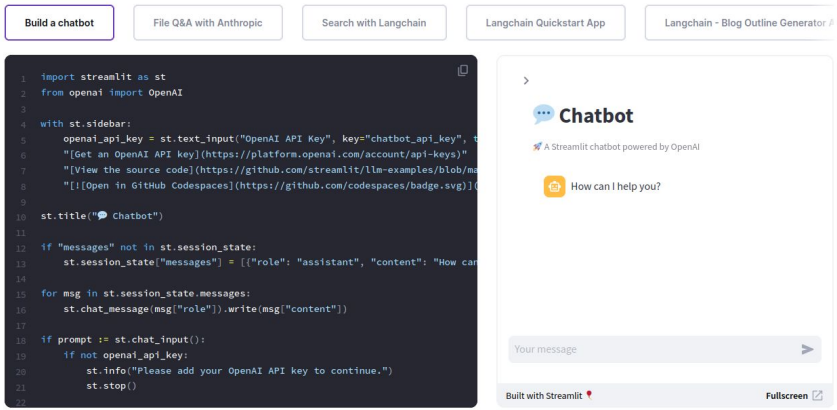
Algorytmy dopasowania

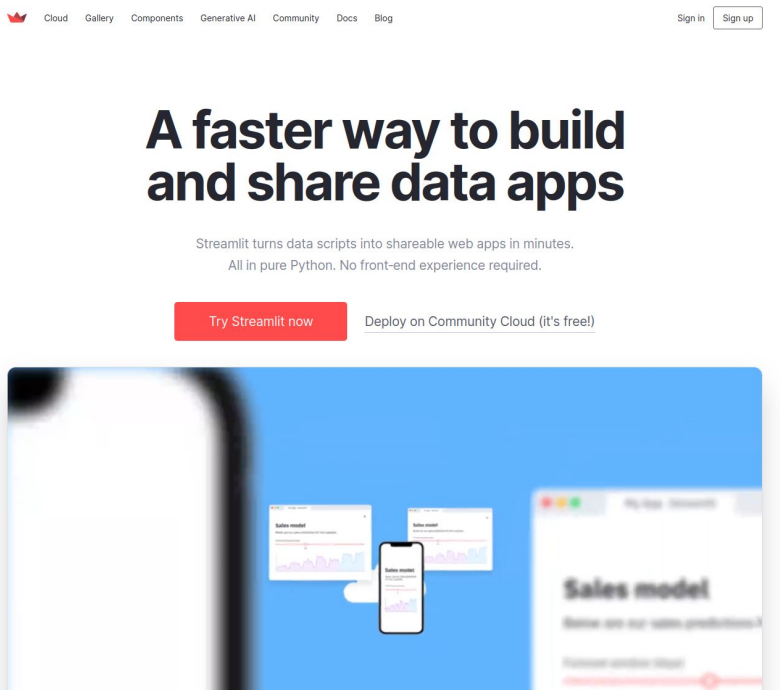
- **Chebyshev Distance (L∞ Norm):** Chebyshev distance is used in chessboard distance calculations, network delay measurements, and anomaly detection in time series data.
- **Mahalanobis Distance:** Mahalanobis distance is used in multivariate statistics for clustering, classification, and outlier detection, such as in healthcare for disease detection.
- **Canberra Distance:** Canberra distance is applied in ecological and biological studies for species diversity measurements, as well as text clustering for document similarity.
- **Bray-Curtis Dissimilarity:** Bray-Curtis dissimilarity is used in ecological community analysis for measuring compositional dissimilarity between species, as well as in market basket analysis for shopping patterns.
- **Haversine Distance:** Haversine distance is used in geographic information systems (GIS) for calculating distances between two points on the Earth's surface, such as for location-based services.
- **Chi-Squared Distance:** Chi-squared distance is applied in feature selection for machine learning, image analysis, and text classification for document similarity.
- **Earth Mover's Distance (Wasserstein Distance):** Earth Mover's Distance is used in image retrieval, computer vision, and transportation logistics to measure the distance between two probability distributions.
- **Bhattacharyya Distance:** Bhattacharyya distance is applied in statistics for measuring the similarity between probability distributions, image recognition, and document classification.

https://crucialbits.com/blog/a-comprehensive-list-of-similarity-search-algorithms/

Streamlit

Framework open-source do budowy aplikacji webowych w Pythonie. Streamlit jest zaprojektowany, aby ułatwić tworzenie interaktywnych dashboardów oraz aplikacji wizualizujących dane bez potrzeby posiadania zaawansowanej wiedzy na temat front-endu i web developmentu.





https://streamlit.io/

Streamlit - komponenty

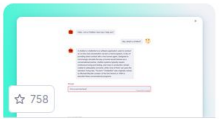
Components

Components are third-party modules that extend what's possible with Streamlit.
Built by creators, for the community 🎈

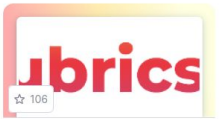
Want to build your own? Check out our **docs**.

CATEGORIES

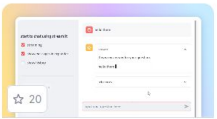
- All
- LLMs
- Widgets
- Charts
- Authentication
- Connections
- Images & video
- Audio
- Text
- Maps
- Dataframes
- Graphs
- Molecules & genes
- Code editors
- Page navigation
- Developer tools
- Integrations



Chat
AI-Yash
`$ pip install streamlit-chat`
View demo →



Trubrics
trubrics
`$ pip install trubrics`
View source →



Chatbox
liunux4odoo
`$ pip install streamlit-chatbox`
View source →

Chat Message
undo76
`$ pip install st-chat-message`
View source →

https://share.streamlit.io/ai-yash/st-chat/main/examples/chatbot.py

Streamlit

https://streamlit.io/generative-ai

```
import streamlit as st
from openai import OpenAI

with st.sidebar:
   openai api key = st.text input("OpenAI API Key", key="chatbot api key", type="password")
   "[Get an OpenAI API key](https://platform.openai.com/account/api-keys)"
   "[View the source code](https://github.com/streamlit/llm-examples/blob/main/Chatbot.py)"
   "[![Open in GitHub
Codespaces](https://github.com/codespaces/badge.svg)](https://codespaces.new/streamlit/llm-examples?quickstart=1)"

st.title("💬 Chatbot")

if "messages" not in st.session_state:
   st.session_state["messages"] = [{"role": "assistant", "content": "How can I help you?"}]

for msg in st.session state.messages:
   st.chat_message(msg["role"]).write(msg["content"])

if prompt := st.chat_input():
   if not openai api key:
       st.info("Please add your OpenAI API key to continue.")
       st.stop()

   client = OpenAI(api_key=openai_api_key)
   st.session state.messages.append({"role": "user", "content": prompt})
   st.chat message("user").write(prompt)
   response = client.chat.completions.create(model="gpt-3.5-turbo", messages=st.session_state.messages)
   msg = response.choices[0].message.content
   st.session_state.messages.append({"role": "assistant", "content": msg})
   st.chat_message("assistant").write(msg)
```

Projekt

Projekt:
streamlit + LangChain + DocArrayInMemorySearch

wzór: https://github.com/rzarno/llm-chatbot-rag-langchain/
1. podmień dokumenty źródłowe w folderze /data
2. Dodaj kilka istotnych zmian w aplikacji
3. Dostosuj prompt
4. Zmień wygląd
5. Przeanalizuj kod, tak aby rozumieć wszystkie elementy aplikacji

Dokumentacja:
https://python.langchain.com/v0.2/docs/introduction/
https://docs.streamlit.io/
https://faiss.ai/index.html

Projekt

W przypadku wystąpienia błędów typu "No module found" należy zainstalować brakujące zależności korzystając z polecenia:

pip install --upgrade nazwa_modułu

Projekt - import dokumentów

```
def import_source_documents(self):
    # load documents
    docs = []
    files = []
    for file in os.listdir("data"):
        if file.endswith(".txt"):
            with open(os.path.join("data", file)) as f:
                docs.append(os.path.join("data", f.read()))
                files.append(file)
```

Projekt - podział na "chunki"

```
# Split documents and store in vector db
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=1000,
    chunk_overlap=200
)

splits = []
for i, doc in enumerate(docs):
    for chunk in text_splitter.split_text(doc):
        splits.append(Document(page_content=chunk, metadata={"source":
files[i]}))

vectordb = DocArrayInMemorySearch.from_documents( splits, self.embedding_model)
```

Projekt - retriever

```
# Define retriever
retriever = vectordb.as_retriever(
    search_type='mmr',
    search_kwargs={'k':2, 'fetch_k':4}
)
```

Projekt - pamięć

```
# Setup memory for contextual conversation
memory = ConversationBufferMemory(
    memory key='chat history',
    output_key='answer',
    return_messages=True
)
```

Projekt - system prompt

```
system_message_prompt = SystemMessagePromptTemplate.from_template(
    """
    You are a chatbot tasked with responding to questions about the Ticos Systems
company.

    You should never answer a question with a question, and you should always
respond with the most relevant page from documents.

    Do not answer questions that are not about the Ticos Systems company.

    Given a question, you should respond with the most relevant documents page
    {context}
    """
)

prompt = ChatPromptTemplate.from_messages([system_message_prompt])
```

Projekt - chain

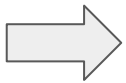
```
# Setup LLM and QA chain
qa chain = ConversationalRetrievalChain.from_llm(
    llm=self.llm,
    retriever=retriever,
    memory=memory,
    return source documents=True,
    verbose=False,
    combine_docs_chain_kwargs={"prompt": prompt}
)
```

Projekt - main app

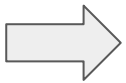
```
@utils.enable chat_history
def main(self):
    user_query = st.chat_input(placeholder="Ask for information from documents" )

    if user_query:
        qa_chain = self.import_source_documents()

        utils.display_msg( user_query, 'user')

        with st.chat message("assistant"):
            st_cb = StreamHandler(st.empty())

            result = qa chain.invoke(
                { "question":user query},
                { "callbacks": [st_cb]}
            )
            response = result["answer"]
            st.session state.messages.append({ "role": "assistant", "content": response})
            utils.print_qa(CustomDocChatbot,  user_query, response)

            # to show references
            for  doc in result['source documents' ]:
                filename = os.path.basename( doc.metadata[ 'source'])
                ref title = f":blue[Source document:  {filename}]"
                with st.popover( ref title):
                    st.caption( doc.page_content)
```

company description

Collection, processing and sale of information; development and marketing of decision-making bodies and development and marketing of risk assessment systems. The company is obliged to comply with commercial principles. The aim is also to protect contractual partners from default risks and to protect borrowers from excessive indebtedness.



MODEL



industry codes

82.91.2 - Credit rating in connection with an individual's or firm's creditworthiness or business practices

XX.XX.XX - ...

...