DigitalSkola
Uncover The World of Digital Skills with Us

Day 4: NLP - Text Classification

Memprediksi kategori data dari input berupa text

Table of Content

What will We Learn Today?

1. Tokenization dan Vectorization (Bag of Works & TF-IDF)
2. NLP Machine Learning Model
3. Eksperimentasi dari kombinasi:
   - Stemming/Lemmatization + Bag of Works/TF-IDF + Model A/B

Vektorisasi

"Proses mengubah token (kata) menjadi representasi angka agar bisa diproses oleh komputer/machine learning model"

💡Metode Vektorisasi

- **Bag of Words (BoW)**: Hitung frekuensi setiap kata di dokumen
- **TF-IDF:** Hitung frekuensi dengan memperhatikan seberapa unik kata di seluruh dokumen

Bag of Words (BoW)

BoW adalah metode representasi teks yang mengubah dokumen menjadi vektor berdasarkan frekuensi kata tanpa memperhatikan urutan atau konteks kata.

📌 Konsep Utama:

- Buat **vocabulary (kosakata unik)** dari seluruh korpus dokumen
- Representasikan setiap dokumen sebagai **vektor** dengan panjang = jumlah kata unik
- Setiap posisi berisi **jumlah kemunculan kata tersebut**

Bag of Words (BoW)

🔧 **Contoh Korpus (2 dokumen):**

1. **"Mobil listrik ramah lingkungan"**
2. **"Motor bensin tidak ramah lingkungan"**

📚 **Vocabulary:**

['mobil', 'listrik', 'ramah', 'lingkungan', 'motor', 'bensin', 'tidak']

Kata	mobil	listrik	ramah	lingkungan	motor	bensin	tidak
Doc 1	1	1	1	1	0	0	0
Doc 2	0	0	1	1	1	1	1

TF-IDF

Term Frequency - Inverse Document Frequency

📘 Definisi:
TF-IDF meningkatkan BoW dengan memberi bobot pada kata berdasarkan:

- **TF (Term Frequency):** Seberapa sering kata muncul dalam dokumen
- **IDF (Inverse Document Frequency):** Seberapa jarang kata muncul di seluruh korpus

📌 **Formula:**

- **TF(t,d)** = jumlah kemunculan term *t* di dokumen *d*
- **IDF(t)** = log(N / (1 + df(t))), di mana:
  - *N* = total jumlah dokumen
  - *df(t)* = jumlah dokumen yang mengandung *t*
- **TF-IDF(t,d)** = TF(t,d) × IDF(t)

TF-IDF

🔧 **Contoh Korpus (2 dokumen):**

1. **"Mobil listrik ramah lingkungan"**
2. **"Motor bensin tidak ramah lingkungan"**

📚 **Vocabulary:**

['mobil', 'listrik', 'ramah', 'lingkungan', 'motor', 'bensin', 'tidak']

Kata	mobil	listrik	ramah	lingkungan	motor	bensin	tidak
TF - Doc 1	1	1	1	1	0	0	0
TF - Doc 2	0	0	1	1	1	1	1
DF	1	1	2	2	1	1	1
Hitung IDF	0 log(2 / (1 + 1))	0 log(2 / (1 + 1))	-0.176 log(2 / (1 + 2))	-0.176 log(2 / (1 + 2))	0 log(2 / (1 + 1))	0 log(2 / (1 + 1))	0 log(2 / (1 + 1))

Hasil Akhir TF-IDF

Kata	mobil	listrik	ramah	lingkungan	motor	bensin	tidak
TF - Doc 1	1	1	1	1	0	0	0
TF - Doc 2	0	0	1	1	1	1	1
DF	1	1	2	2	1	1	1
Hitung IDF	0 log(2 / (1 + 1))	0 log(2 / (1 + 1))	-0.176 log(2 / (1 + 2))	-0.176 log(2 / (1 + 2))	0 log(2 / (1 + 1))	0 log(2 / (1 + 1))	0 log(2 / (1 + 1))
TF-IDF (Doc 1)	1 × 0 = 0	1 × 0 = 0	1 × -0.176 ≈ -0.176	-0.176	0	0	0
TF-IDF (Doc 2)	0	0	-0.176	-0.176	0	0	0

📌 Di sini kata “ramah” dan “lingkungan” memiliki bobot rendah karena muncul di semua dokumen kurang informatif.
Kata unik seperti “mobil”, “motor” → nilai tinggi dalam korpus lebih besar.

#BertalentaDigital

Hands-on Python

Buka Notebook Classification ML Model

DigitalSkola
Uncover The World of Digital Skills with Us

Thank You.

DigitalSkola
Uncover The World of Digital Skills with Us

Day 4: Introduction to LLM & RAG Concept

Memahami konsep implementasi AI

Table of Content

What will We Learn Today?

1. Konsep Large Language Model (LLM)
2. Konsep Retrieval Augmented Generation (RAG)
3. Hugging Face dan Perannya dalam RAG
4. Text Splitter & Chunking
5. Vector Embedding & Vector Store

Apa itu Large Language Model (LLM) dan seperti apa konsepnya?

LLM adalah model bahasa yang menimbang pentingnya setiap kata dalam kalimat terhadap kata lainnya (transformer) yang dilatih pada jumlah data teks yang cukup banyak.

📌 Konsep Token pada LLM:

- LLM menghitung input/output dalam unit token (**1 token = ~4 karakter**)
- Dimana **1 token = ~ 0.75 kata**

Sumber: openai.com/tokenizer

Fungsinya LLM pada RAG

LLM sebagai **Generator**, menghasilkan jawaban dari input user. Namun, **LLM tidak selalu dapat mengakses data eksternal baru**, maka diperlukan ***augmentasi retrieval***

Model	Max Token	Kemampuan	Cost (per 1k token)
GPT-3.5-turbo	16,000	Cepat dan murah	~$0.0015 (input), ~$0.002 (output)
GPT-4	128,000	Akurasi dan reasoning tinggi	~$0.01–$0.03 (input/output)
GPT-4o	128,000	Lebih cepat & multimodal	Lebih murah dari GPT-4

Frontier LLM

Beberapa jenis LLM

Claude

OpenAI

deepseek

Apa itu Retrieval Augmented Generator (RAG) dan seperti apa konsepnya?

RAG adalah arsitektur yang menggabungkan dua komponen, Retriever yang menemukan dokumen relevan dari knowledge base, dan Generator yang menghasilkan jawaban dari dokumen yang ditemukan

📌 Alur kerja RAG:

1. User input query
2. Retriever mencari dokumen relevan (via vector database)
3. Dokumen dikombinasikan dengan prompt
4. LLM menghasilkan jawaban

Kenapa perlu RAG?

RAG mengurangi hallucination dan meningkatkan akurasi dengan mengambil konteks dari sumber eksternal.

Format	Contoh Ekstensi	Cara Ekstraksi Konten
Dokumen Word	.doc, .docx	python-docx, docx2txt
PDF	.pdf	PyMuPDF, pdfminer, pdfplumber
Spreadsheet	.xlsx, .xls, .csv	pandas, openpyxl
Email	.eml, .msg	email (Python stdlib), extract_msg
JSON/YAML	.json, .yaml	json, PyYAML
Markdown	.md	Parser markdown biasa
Slide	.ppt, .pptx	python-pptx
Scan/Gambar	.jpg, .png, .tiff	OCR (Tesseract, easyocr)

Hugging Face dan Fiturnya

Hugging Face

Platform open-source untuk model ML/NLP seperti BERT, LLaMA, Falcon, dll. Menyediakan:

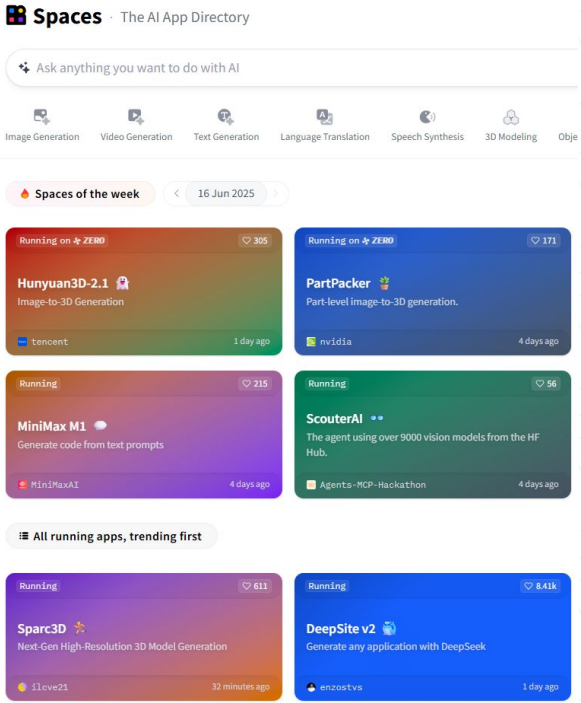
- Model siap pakai (🤗 Transformers)
- Dataset & pipeline
- Inference API dan Space

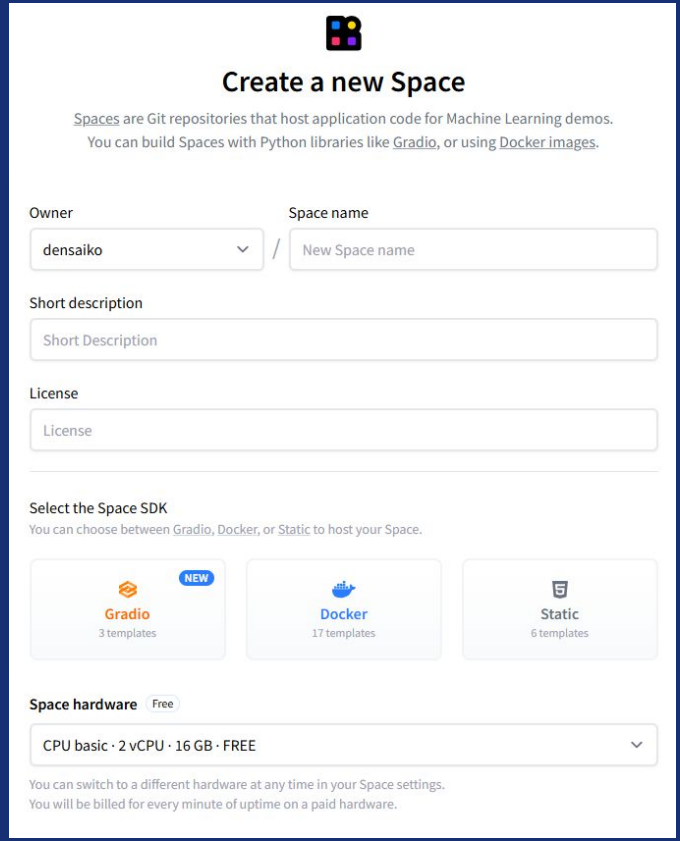
Register dan create token

→ https://huggingface.co/

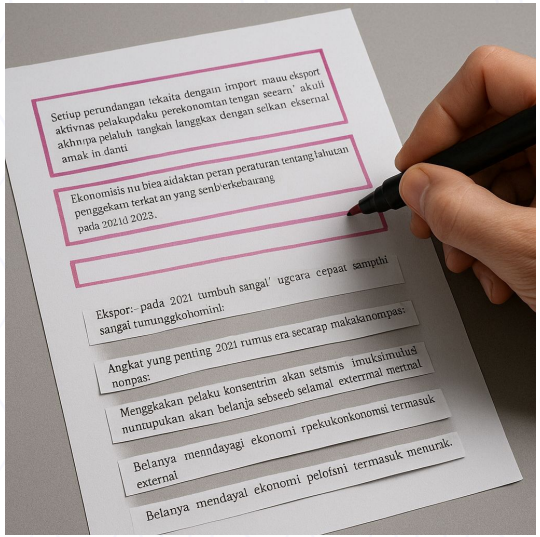
Space

fitur hosting aplikasi interaktif (seperti demo model AI)





Pernah Kepikiran Bagaimana RAG dibentuk?



Text Splitting

+

Chunking

Text Splitter

Fungsi untuk memecah dokumen panjang menjadi potongan-potongan kecil (chunk), agar sesuai dalam token limit LLM.

📌 Best Practice:

- Ideal chunk: 300–500 token
- Overlap: 20–50 token → agar tidak kehilangan konteks

```
text_splitter = CharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
chunks = text_splitter.split_documents(documents)
```

Chunk

potongan dokumen hasil split, yang akan diproses selanjutnya menjadi vektor

📌 Karakteristik:

- Idealnya mengandung satu ide atau paragraf utuh
- Masih mengandung konteks (karena overlap)
- Format: text biasa (string), bisa disertai metadata

Vector Embedding & Vector Store

Vector Embedding

Teknik untuk mengubah teks (kalimat, paragraf, atau dokumen) menjadi representasi numerik (vektor) di ruang berdimensi tinggi.

📌 Contoh

- Kalimat: "Indonesia is a country in Southeast Asia"
- Menjadi: [0.023, -0.122, 0.981, ..., 0.005]

Vector Store

Sistem penyimpanan dan pencarian vektor embedding dari dokumen, yang memungkinkan pencarian berbasis makna (semantic search)

📌 **Contoh Vector Store:**

- **Chroma:** vector database terintegrasi langsung dengan Langchain, fokus untuk prototipe RAG dan NLP apps.
- **FAISS:** library open-source buatan Meta (Facebook AI) untuk melakukan pencarian vektor secara efisien dan cepat, terutama pada dataset berskala besar.

Chroma VS FAISS

Vector Store

Aspek	Chroma	FAISS
Kelebihan	- Python native, sangat mudah digunakan - Mendukung metadata secara native - Integrasi langsung dengan Langchain - Cocok untuk prototyping cepat	- Sangat cepat untuk jutaan vektor - Mendukung banyak jenis index (Flat, IVF, PQ, HNSW) - Teruji untuk production skala besar
Keterbatasan	- Kurang efisien untuk dataset sangat besar - Fitur indexing lebih terbatas - Masih berkembang, belum sekuat FAISS di industri	- Tidak menyimpan metadata secara native - Perlu pembungkus tambahan di Langchain - Lebih kompleks untuk setup awal
Use Case	- Prototyping chatbot RAG - Aplikasi skala kecil hingga menengah - Edukasi dan penelitian NLP	- Sistem pencarian skala besar - Recommender system real-time - Deployment production enterprise

DigitalSkola
Uncover The World of Digital Skills with Us

Thank You.