Enterprise Data Management Laboratory

NLP in a Nutshell

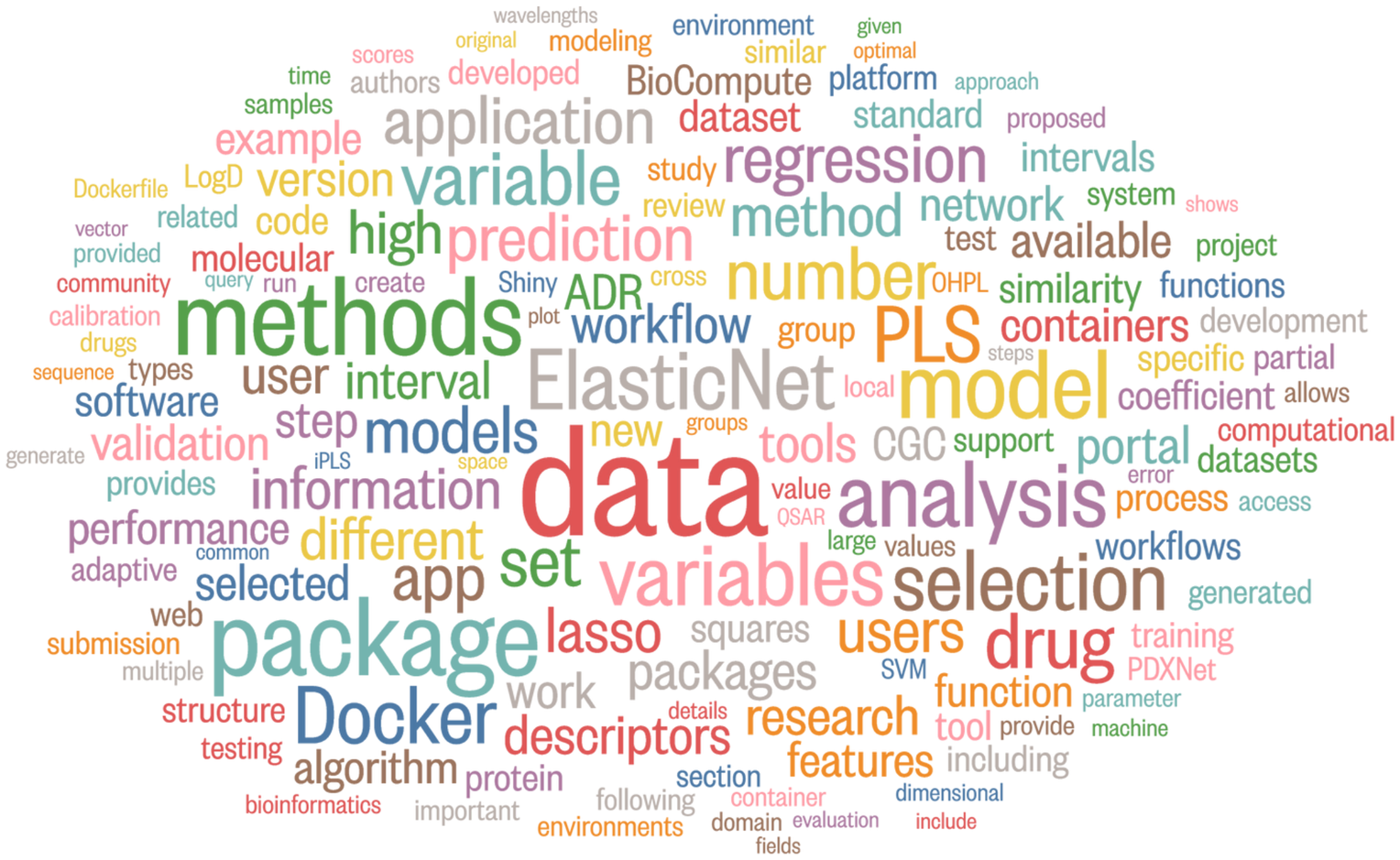
Natural Language Processing

Deep Learning for Text Mining

GWE #7

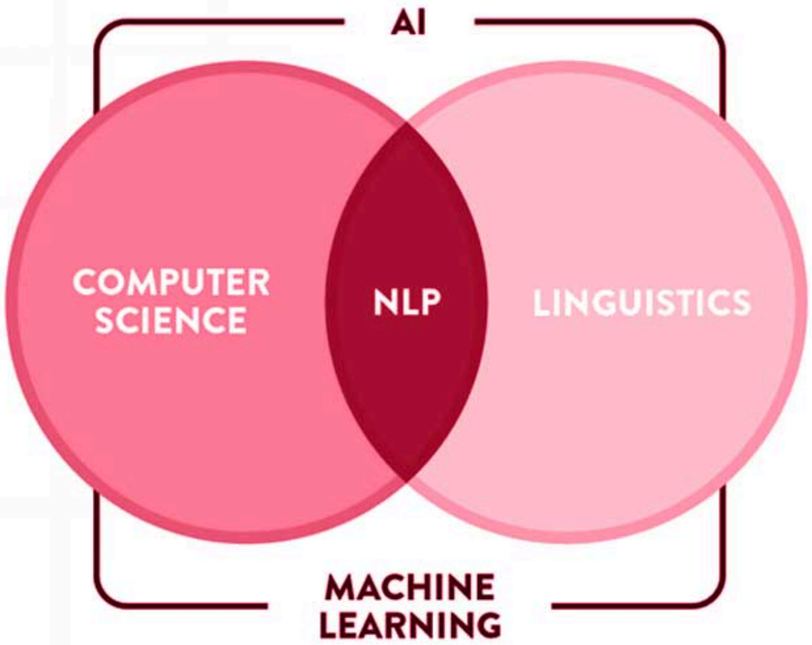
California	5,593	5,495	5,159	4,795	4,463
Colorado	419	454	402	442	353
Idaho	1,440	1,566	1,688	1,706	1,530
Kansas	416	551	569	592	613
Michigan	1,299	1,846	1,971	2,061	1,960
Minnesota	2,297	2,756	2,645	2,600	2,487
Mississippi	701	863	958	1,056	1,169
Montana	2,375	3,144	3,149	3,313	3,271
Nebraska	1,010	1,229	1,339	1,396	1,267
Nevada	1,373	1,611	1,508	1,468	1,328
New Mexico	2,533	2,853	2,966	2,890	2,809
New York	380	384	369	452	350
North Carolina	584	736	743	700	671
North Dakota	3,840	4,800	4,976	5,661	5,569
Oklahoma	25,678	29,012	31,042	33,588	32,795
Oregon	678	871	800	785	687
South Dakota	7,457	8,123	8,208	8,505	8,525
Texas	117	131	142	124	114
Utah	117	167	217	421	384

"Computers are great at working with structured data like spreadsheets and database tables."



But us humans usually communicate in words, not in tables. A lot of information in the world is unstructured — raw text in English or another human language.

Solusi — *Natural Language Processing (NLP)*

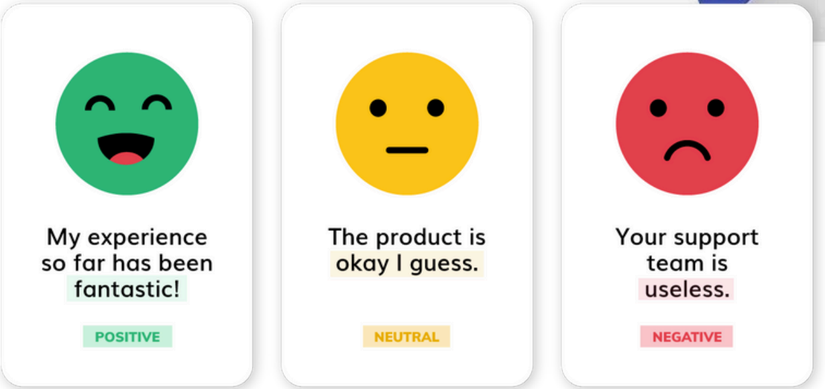


Tujuan utama:

- Memahami (understand)
- Menafsirkan (interpret)
- Menghasilkan (generate) teks

ChatGPT

Google Translate



Tantangan Bahasa Indonesia dalam NLP

- Bahasa manusia sangat kompleks dan ambigu
- Satu kalimat bisa punya banyak arti tergantung konteks
- Bahasa Indonesia punya berbagai dialek, gaya bicara, dan bahasa gaul

NLP Pipelines

alur kerja bertahap untuk memproses teks

01 Text Acquisition

02 Text Cleaning

03 Text Preprocessing

04 Feature Extraction

05 Modeling

06 Evaluation

07 Hands-On!

Text Acquisition

(Pengumpulan Teks)

Definisi

Pengumpulan teks adalah langkah awal dalam pipeline NLP yang melibatkan proses mendapatkan data teks mentah dari berbagai sumber untuk diproses lebih lanjut.

Lebih spesifik di modul 1

Web Scraping

Mengekstrak data teks dari halaman web secara otomatis.

Library : BeautifulSoup & Request

API Integration

Menggunakan API untuk mengakses data teks dari platform seperti Twitter, Reddit, atau layanan berita.

Library : tweepy (buat twitter)

File Processing

Membaca data teks dari berbagai format file (TXT, CSV, PDF, DOCX, dll).

Library : pandas (buat csv & excel), PyPDF2 (pdf), docx, dll

Text Cleaning

(Pembersihan Teks)

Text Cleaning

Pembersihan teks adalah proses menghilangkan noise dan data yang tidak relevan dari teks mentah untuk meningkatkan kualitas analisis selanjutnya.

NLTK

spaCy

Menghapus HTML dan XML Tags

Membersihkan markup HTML/XML dari teks yang diekstrak dari web.

Menghapus HTML dan XML Tags

```python
import re

def clean_html(text):
    """
    Menghapus tag HTML/XML dari teks
    """
    # Pola untuk mencocokkan tag HTML/XML
    pattern = re.compile('<.*?>')

    # Menghapus semua tag
    clean_text = re.sub(pattern, '', text)

    # Mengganti multiple whitespace dengan single space
    clean_text = re.sub(r'\s+', ' ', clean_text)

    return clean_text.strip()

# Contoh penggunaan
html_text = "<p>Ini adalah <b>contoh</b> teks <i>dengan</i> <span style='color:red'>tag HTML</span>.</p>"
clean_text = clean_html(html_text)
print(clean_text)  # Output: "Ini adalah contoh teks dengan tag HTML."
```

Menghapus Karakter Khusus dan Angka

Menghilangkan karakter non-alfabet yang tidak diperlukan.

Menghapus Karakter Khusus dan Angka

```python
def remove_special_chars(text, keep_numbers=False):
    """
    Menghapus karakter khusus dan opsi untuk menghapus angka
    """
    if keep_numbers:
        # Menyimpan hanya huruf, angka, dan spasi
        pattern = re.compile(r'[^a-zA-Z0-9\s]')
    else:
        # Menyimpan hanya huruf dan spasi
        pattern = re.compile(r'[^a-zA-Z\s]')

    # Menghapus karakter khusus
    clean_text = re.sub(pattern, '', text)

    # Mengganti multiple whitespace dengan single space
    clean_text = re.sub(r'\s+', ' ', clean_text)

    return clean_text.strip()

# Contoh penggunaan
text = "Hello! This is a sample text with numbers (123) and special characters #$%."
clean_text1 = remove_special_chars(text, keep_numbers=True)
print(clean_text1)  # Output: "Hello This is a sample text with numbers 123 and special characters"

clean_text2 = remove_special_chars(text, keep_numbers=False)
print(clean_text2)  # Output: "Hello This is a sample text with numbers  and special characters"
```

Menghapus URL, Email, dan Username

Menghilangkan alamat web, email, dan username dari teks.

Menghapus URL, Email, dan Username

```python
def remove_urls_emails_usernames(text):
    """
    Menghapus URL, alamat email, dan username dari teks
    """
    # Pola untuk URL
    text = re.sub(r'https?://\S+|www\.\S+', '[URL]', text)

    # Pola untuk alamat email
    text = re.sub(r'\S+@\S+', '[EMAIL]', text)

    # Pola untuk username (diawali dengan @)
    text = re.sub(r'@\S+', '[USERNAME]', text)

    return text

# Contoh penggunaan
text = "Kunjungi website kami di https://example.com atau hubungi info@example.com. Jangan lupa follow @username di Twitter!"
clean_text = remove_urls_emails_usernames(text)
print(clean_text)  # Output: "Kunjungi website kami di [URL] atau hubungi [EMAIL]. Jangan lupa follow [USERNAME] di Twitter!"
```

snappify.com

Text Cleaning

Pembersihan teks adalah proses menghilangkan noise dan data yang tidak relevan dari teks mentah untuk meningkatkan kualitas analisis selanjutnya.

NLTK

spaCy

Menangani Masalah Encoding

Memperbaiki masalah encoding seperti karakter Unicode yang rusak.

Menghapus URL, Email, dan Username

```python
import ftfy

def fix_encoding(text):
    """
    Memperbaiki masalah encoding pada teks
    """
    # Menggunakan ftfy untuk memperbaiki encoding
    fixed_text = ftfy.fix_text(text)

    return fixed_text

# Contoh penggunaan
text_with_encoding_issues = "Teks dengan masalah encoding â€" seperti tanda kutip "smart quotes" dan tanda dolar â‚¬."
fixed_text = fix_encoding(text_with_encoding_issues)
print(fixed_text)  # Output: "Teks dengan masalah encoding — seperti tanda kutip "smart quotes" dan tanda dolar €."
```

Menangani Case Folding

Mengubah semua teks menjadi huruf kecil atau kapital.

Menghapus URL, Email, dan Username

```python
def case_folding(text, to_lower=True):
    """
    Mengubah teks menjadi huruf kecil atau kapital
    """
    if to_lower:
        return text.lower()
    else:
        return text.upper()

# Contoh penggunaan
text = "Ini adalah Contoh TEKS dengan Kapitalisasi Campuran."
lower_text = case_folding(text, to_lower=True)
print(lower_text)  # Output: "ini adalah contoh teks dengan kapitalisasi campuran."

upper_text = case_folding(text, to_lower=False)
print(upper_text)  # Output: "INI ADALAH CONTOH TEKS DENGAN KAPITALISASI CAMPURAN."
```

Menangani Whitespace

Menormalkan whitespace dan menghapus baris kosong.

Menghapus URL, Email, dan Username

```python
def normalize_whitespace(text):
    """
    Menormalkan whitespace dalam teks
    """
    # Mengganti tab, newline, dll dengan spasi
    text = re.sub(r'[\t\n\r\f\v]', ' ', text)

    # Mengganti multiple whitespace dengan single space
    text = re.sub(r'\s+', ' ', text)

    return text.strip()

# Contoh penggunaan
text = """Ini adalah
contoh     teks dengan
    whitespace     yang tidak normal."""
normalized_text = normalize_whitespace(text)
print(normalized_text)  # Output: "Ini adalah contoh teks dengan whitespace yang tidak normal."
```

Text Cleaning (indo version)

Pembersihan teks adalah proses menghilangkan noise dan data yang tidak relevan dari teks mentah untuk meningkatkan kualitas analisis selanjutnya.

remove_html

Menghapus tag HTML dalam teks

remove_url

Menghapus URL dari teks

replace_slang

Mengganti kata gaul (alay) menjadi kata formal

replace_word_elongation

Mengubah emoji menjadi kata-kata deskriptif

emoji_to_words & words_to_emoji

Mengubah emoji menjadi kata-kata deskriptif (& sebaliknya)

indoNLP Text Cleaning

```
from indoNLP.preprocessing import remove_html
remove_html("website <a href='https://google.com'>google</a>")
# Output: "website google"

from indoNLP.preprocessing import remove_url
remove_url("retrieved from https://gist.github.com/gruber/8891611")
# Output: "retrieved from "

from indoNLP.preprocessing import replace_slang
replace_slang("emg siapa yg nanya?")
# Output: "memang siapa yang bertanya?"

from indoNLP.preprocessing import replace_word_elongation
replace_word_elongation("kenapaaa?")
# Output: "kenapa?"

from indoNLP.preprocessing import emoji_to_words
emoji_to_words("emoji 😀😄")
# Output: "emoji !wajah_gembira!!wajah_gembira_dengan_mata_bahagia!"
```

Text Preprocessing

(Pra-pemrosesan Teks)

1) Tokenization

Tokenisasi adalah proses memecah teks menjadi unit-unit yang lebih kecil (token) seperti kata, frasa, atau karakter.

NLTK

spaCy

Word Tokenization

Memecah teks menjadi kata-kata individual.

Sentence Tokenization

Memecah teks menjadi kalimat-kalimat.

Word Tokenization

```python
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize

def tokenize_words(text):
    """
    Memecah teks menjadi token kata
    """
    tokens = word_tokenize(text)
    return tokens

# Contoh penggunaan
text = "Natural language processing (NLP) is a subfield of artificial intelligence."
word_tokens = tokenize_words(text)
print(word_tokens)
# Output: ['Natural', 'language', 'processing', '(', 'NLP', ')', 'is', 'a', 'subfield', 'of', 'artificial', 'intelligence', '.']
```

Sentence Tokenization

```python
from nltk.tokenize import sent_tokenize

def tokenize_sentences(text):
    """
    Memecah teks menjadi kalimat-kalimat
    """
    sentences = sent_tokenize(text)
    return sentences

# Contoh penggunaan
text = "NLP is fascinating. It has many applications in various fields. Companies are investing heavily in this technology."
sentence_tokens = tokenize_sentences(text)
print(sentence_tokens)
# Output: ['NLP is fascinating.', 'It has many applications in various fields.',
'Companies are investing heavily in this technology.']
```

1) Tokenization

Tokenisasi adalah proses memecah teks menjadi unit-unit yang lebih kecil (token) seperti kata, frasa, atau karakter.

Character Tokenization

Memecah teks menjadi unit terkecil yaitu karakter satuan.

Subword Tokenization

Memecah kata menjadi unit yang lebih kecil dari kata, sangat berguna untuk model Transformer. (DL)

Character Tokenization

```python
def tokenize_characters(text):
    """
    Memecah teks menjadi karakter individual
    """
    return list(text)

# Contoh penggunaan
text = "NLP!"
char_tokens = tokenize_characters(text)
print(char_tokens)
# Output: ['N', 'L', 'P', '!']
```

Subword Tokenization

```python
from transformers import BertTokenizer

def tokenize_subwords(text):
    """
    Memecah teks menjadi subword menggunakan BERT tokenizer
    """
    tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
    tokens = tokenizer.tokenize(text)
    return tokens

# Contoh penggunaan
text = "Transformers revolutionized NLP with unsupervised pretraining."
subword_tokens = tokenize_subwords(text)
print(subword_tokens)
# Output: ['transform', '##ers', 'revolution', '##ized', 'nl', '##p', 'with', 'un', '##super', '##vised', 'pre', '##train', '##ing', '.']
```

2) Normalization

Normalisasi adalah proses mengubah kata menjadi bentuk standar untuk mengurangi variasi yang tidak perlu.

NLTK

spaCy

Stemming

Menghilangkan akhiran kata untuk mendapatkan bentuk dasar.

Stemming

```python
from nltk.stem import PorterStemmer

def stem_words(tokens):
    """
    Melakukan stemming pada daftar token
    """
    stemmer = PorterStemmer()
    stemmed_words = [stemmer.stem(word) for word in tokens]
    return stemmed_words

# Contoh penggunaan
words = ["jumping", "jumps", "jumped", "connection", "connecting", "connected"]
stemmed_words = stem_words(words)
print(stemmed_words)
# Output: ['jump', 'jump', 'jump', 'connect', 'connect', 'connect']
```

Lemmatization

Mengubah kata menjadi bentuk lemma (kata dasar dalam kamus).

Lemmatization

```python
from nltk.stem import WordNetLemmatizer
import nltk
nltk.download('wordnet')

def lemmatize_words(tokens, pos_tags=None):
    """
    Melakukan lemmatization pada daftar token
    """
    lemmatizer = WordNetLemmatizer()

    if pos_tags:
        # Menggunakan POS tag untuk lemmatization yang lebih akurat
        lemmatized_words = [lemmatizer.lemmatize(token, pos=pos_tag) for token, pos_tag in zip(tokens, pos_tags)]
    else:
        # Lemmatization tanpa POS tag
        lemmatized_words = [lemmatizer.lemmatize(token) for token in tokens]

    return lemmatized_words

# Contoh penggunaan
words = ["better", "worst", "running", "says", "mice", "feet"]
lemmatized_words = lemmatize_words(words)
print(lemmatized_words)
# Output: ['better', 'worst', 'running', 'says', 'mouse', 'foot']
```

2) Normalization (indo version)

Normalisasi adalah proses mengubah kata menjadi bentuk standar untuk mengurangi variasi yang tidak perlu.

Stemming with Sastrawi 🇮🇩

proses mengubah kata menjadi bentuk dasarnya (akar kata) dengan memotong imbuhan (prefix, infix, suffix) sesuai dengan aturan bahasa Indonesia.

Sastrawi Stemming

```python
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory

factory = StemmerFactory()
stemmer = factory.create_stemmer()

text = "Pemerintah sedang mempercepat pembangunan infrastruktur."
stemmed_text = stemmer.stem(text)
print(stemmed_text)
# Output: 'perintah sedang cepat bangun infrastruktur'
```

- Stemming: Mengubah kata berimbuhan menjadi bentuk dasarnya.
- Berbasis aturan linguistik: Menggunakan aturan-aturan bahasa Indonesia yang sudah baku.

3) Stopword Removal

Proses menghilangkan kata-kata yang sangat umum dan tidak memberikan informasi signifikan.

NLTK

spaCy

Stopword Removal - NLTK (eng version)

```python
from nltk.corpus import stopwords
import nltk
nltk.download('stopwords')

def remove_stopwords(tokens, language='english'):
    """
    Menghapus stopwords dari daftar token
    """
    stop_words = set(stopwords.words(language))
    filtered_tokens = [token for token in tokens if token.lower() not in stop_words]
    return filtered_tokens

# Contoh penggunaan
text = "This is an example of stopword removal in the context of NLP."
tokens = word_tokenize(text)
filtered_tokens = remove_stopwords(tokens)
print(filtered_tokens)
# Output: ['This', 'example', 'stopword', 'removal', 'context', 'NLP', '.']
```

3) Stopword Removal (indo version)

Proses menghilangkan kata-kata yang sangat umum dan tidak memberikan informasi signifikan.

Stopword Removal with Sastrawi

StopWord - Sastrawi

```python
from Sastrawi.StopWordRemover.StopWordRemoverFactory import StopWordRemoverFactory
from Sastrawi.StopWordRemover.StopWordRemover import StopWordRemover

# Inisialisasi stopword remover dari Sastrawi
factory = StopWordRemoverFactory()
stopword_remover = factory.create_stop_word_remover()

def sastrawi_stopword_removal(text):
    """
    Menghapus stopword dari teks Bahasa Indonesia menggunakan Sastrawi
    """
    return stopword_remover.remove(text)

# Contoh penggunaan
text = "Saya pergi ke pasar dan membeli sayur untuk memasak di rumah."
cleaned = sastrawi_stopword_removal(text)
print(cleaned)
# Output bisa: "Saya pergi pasar membeli sayur memasak rumah"
```

Stopword Removal with indoNLP

Stopword Removal - indoNLP

```python
from indoNLP.preprocessing import remove_stopwords

def indonlp_stopword_removal(text):
    """
    Menghapus stopword dari teks Bahasa Indonesia menggunakan indoNLP
    """
    return remove_stopwords(text)

# Contoh penggunaan
text = "Saya pergi ke pasar dan membeli sayur untuk memasak di rumah."
cleaned = indonlp_stopword_removal(text)
print(cleaned)
# Output bisa: "pergi pasar membeli sayur memasak rumah."
```

4) Part-of-Speech Tagging

POS Tagging adalah proses memberikan label pada setiap kata berdasarkan kelas gramatikal (kata benda, kata kerja, dll).

NLTK

spaCy

PoS Tagging with NLTK

PoS Tagging - NLTK

```python
import nltk
nltk.download('averaged_perceptron_tagger')

def pos_tag_nltk(text):
    """
    Melakukan POS tagging dengan NLTK
    """
    tokens = word_tokenize(text)
    tagged_tokens = nltk.pos_tag(tokens)
    return tagged_tokens

# Contoh penggunaan
text = "The quick brown fox jumps over the lazy dog."
pos_tags = pos_tag_nltk(text)
print(pos_tags)
# Output: [('The', 'DT'), ('quick', 'JJ'), ('brown', 'JJ'), ('fox', 'NN'), ('jumps', 'VBZ'),
# ('over', 'IN'), ('the', 'DT'), ('lazy', 'JJ'), ('dog', 'NN'), ('.', '.')]
```

4) Named Entity Recognition (NER)

NER adalah proses mengidentifikasi dan mengklasifikasikan entitas bernama dalam teks ke dalam kategori seperti orang, organisasi, lokasi, dll.

NLTK

spaCy

NER with NLTK

NER - NLTK

```python
import nltk
nltk.download('maxent_ne_chunker')
nltk.download('words')

def ner_nltk(text):
    """
    Melakukan Named Entity Recognition dengan NLTK
    """
    tokens = word_tokenize(text)
    pos_tags = nltk.pos_tag(tokens)
    ne_tree = nltk.ne_chunk(pos_tags)

    # Ekstrak entitas bernama
    named_entities = []
    for subtree in ne_tree:
        if hasattr(subtree, 'label'):
            entity_text = ' '.join([word for word, tag in subtree.leaves()])
            entity_type = subtree.label()
            named_entities.append((entity_text, entity_type))

    return named_entities

# Contoh penggunaan
text = "Apple Inc. is planning to open a new store in New York City next month. Tim Cook made the announcement yesterday."
entities = ner_nltk(text)
print(entities)
# Output: [('Apple Inc.', 'ORGANIZATION'), ('New York City', 'GPE'), ('Tim Cook', 'PERSON')]
```

Feature Extraction

(Ekstraksi Fitur)

1) Count-Based Features

Fokus pada menghitung kemunculan token dalam dokumen.

Bag-of-Words (BoW)

- Apa itu?: Menghitung frekuensi setiap kata dalam dokumen.
- Kelebihan: Sederhana dan cepat.
- Kekurangan: Mengabaikan urutan kata dan konteks.

Count Vectorizer (BoW)

```python
from sklearn.feature_extraction.text import CountVectorizer

korpus = [
    "saya suka belajar NLP",
    "belajar NLP itu menyenangkan",
]
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(korpus)
print(vectorizer.get_feature_names_out())
print(X.toarray())
```

```
['belajar' 'itu' 'menyenangkan' 'nlp' 'saya' 'suka']
[[1 0 0 1 1 1]
 [1 1 1 1 0 0]]
```

with n-grams

- Apa itu?: Token berurutan sepanjang n (contoh: unigram = 1 kata, bigram = 2 kata).
- Manfaat: Menangkap konteks lokal (frasa umum).

Count Vectorizer (BoW) with ngram(1,2)

```python
from sklearn.feature_extraction.text import CountVectorizer

korpus = [
    "saya suka belajar NLP",
    "belajar NLP itu menyenangkan",
]
vectorizer = CountVectorizer(ngram_range=(1, 2))
X = vectorizer.fit_transform(korpus)
print(vectorizer.get_feature_names_out())
print(X.toarray())
```

```
['belajar' 'belajar nlp' 'itu' 'itu menyenangkan' 'menyenangkan' 'nlp'
 'nlp itu' 'saya' 'saya suka' 'suka' 'suka belajar']
[[1 1 0 0 0 1 0 1 1 1 1]
 [1 1 1 1 1 1 1 0 0 0 0]]
```

2) Weight-Based Features

Memberi bobot pada kata berdasarkan seberapa penting kata dalam dokumen vs seluruh korpus.

TF–IDF (Term Frequency–Inverse Document Frequency)

- Term Frequency (TF): Menghitung seberapa sering kata muncul dalam dokumen.
  - Contoh: Jika kata "NLP" muncul 3 kali di dokumen dengan total 100 kata, maka TF = 3/100 = 0.03.
- Inverse Document Frequency (IDF): Mengukur seberapa jarang kata muncul di seluruh korpus.
  - Rumus: log(total dokumen / (1 + jumlah dokumen yang mengandung kata)).
  - Contoh: Jika ada 10.000 dokumen, dan 100 dokumen mengandung kata "NLP", maka IDF = log(10.000 / 101) ≈ 4.6.
- TF–IDF: Perkalian TF × IDF sehingga kata yang spesifik memiliki bobot lebih tinggi, sedangkan kata umum mendapat bobot rendah.

TfIdfVectorizer with ngram(1,2)

```python
from sklearn.feature_extraction.text import TfidfVectorizer

korpus = [
    "saya suka belajar NLP",
    "belajar NLP itu menyenangkan",
]

vectorizer = TfidfVectorizer(ngram_range=(1, 2), max_features=1000)
X = vectorizer.fit_transform(korpus)
X = X.toarray()
X
```

```
array([[0.30287281, 0.30287281, 0.        , 0.        , 0.        ,
        0.30287281, 0.        , 0.42567716, 0.42567716, 0.42567716,
        0.42567716],
       [0.30287281, 0.30287281, 0.42567716, 0.42567716, 0.42567716,
        0.30287281, 0.42567716, 0.        , 0.        , 0.        ,
        0.        ]])
```

3) Static Word Embeddings

Memberi bobot pada kata berdasarkan seberapa penting kata dalam dokumen vs seluruh korpus.

Word2Vec

- Pendekatan: Predictive (menggunakan jaringan saraf shallow).
- Dua arsitektur utama:
  - CBOW (Continuous Bag of Words): memprediksi kata berdasarkan konteks.
  - Skip-gram: memprediksi konteks berdasarkan kata tenga

FastText

- Dasar: Word2Vec + subword level (ngram karakter)
- Kelebihan:
  - Bisa menghasilkan embedding untuk kata Out-of-Vocabulary (OOV).
  - Menangani kata turunan atau typo lebih baik.

Cari contoh di google :)

GloVe (Global Vectors)

- Pendekatan: Count-based → Matrix factorization dari matriks co-occurrence.
- Inti ide: Vektor kata diturunkan dari statistik global kata yang muncul bersama (bukan prediktif).
- Kelebihan: Menangkap informasi statistik secara global, bisa sangat akurat di dataset besar.

4) Contextual Embeddings

Representasi kata dalam bentuk vektor yang berubah **tergantung konteks kalimat.** Berbeda dari static embedding (seperti Word2Vec, GloVe, atau FastText) yang memberikan vektor tetap untuk setiap kata, contextual embedding memahami arti kata berdasarkan kata-kata di sekitarnya. Contextual embedding dihasilkan oleh model language berbasis Transformer, seperti: BERT, RoBERTa, GPT, dl.

indoBERT

indoBERT embedding

```python
from transformers import AutoTokenizer, AutoModel
import torch

# Load tokenizer dan model IndoBERT
tokenizer = AutoTokenizer.from_pretrained("indobenchmark/indobert-base-p1")
model = AutoModel.from_pretrained("indobenchmark/indobert-base-p1")

# Kalimat input
kalimat = "Saya suka belajar NLP di kampus"

# Tokenisasi dan konversi ke tensor
inputs = tokenizer(kalimat, return_tensors="pt")

# Nonaktifkan gradient (tidak untuk training)
with torch.no_grad():
    outputs = model(**inputs)

# Ambil token embedding dari layer terakhir
token_embeddings = outputs.last_hidden_state  # Shape: [1, sequence_length, hidden_dim]

# Rata-rata semua token untuk jadi vektor kalimat
sentence_embedding = token_embeddings.mean(dim=1)  # Shape: [1, hidden_dim]

print(sentence_embedding.shape)  # Misal: torch.Size([1, 768])
```

```
torch.Size([1, 768])
```

```
tensor([[ 4.8018e-01,  8.2636e-01,  8.8317e-01, -3.5935e-01,  7.1930e-01,
          4.6947e-01, -5.3926e-01,  5.4497e-02,  9.9209e-01, -5.3490e-01,
          1.8890e-01, -2.6955e-02, -1.0131e+00,  2.4429e-02, -8.9917e-02,
         -1.1298e+00,  5.0868e-01, -5.3319e-02, -1.5839e-01,  4.8461e-01,
         -8.4600e-02, -3.4631e-01, -1.1506e+00, -5.0955e-01,  7.6826e-02,
         -4.5711e-01,  1.6453e-01,  1.4845e-01, -9.0497e-02, -1.4449e+00,
          4.3803e-02, -1.4646e-01,  5.5173e-01,  2.4741e-01,  2.4122e-01,
          2.2613e+00, -7.3430e-01,  7.0207e-01, -6.2872e-01, -8.5002e-01,
         -9.9678e-01, -2.8730e-01, -8.1574e-03, -6.0290e-01,  3.9700e-01,
          1.6693e-01, -3.3470e-01,  4.6702e-03, -2.5689e-01,  3.1600e-01,
          5.1449e-02, -1.1492e+00,  1.0218e+00,  5.9906e-01,  2.4342e-01,
         -7.5113e-02, -5.1411e-02, -4.2787e-01, -3.7678e-01,  2.1781e-01,
         -7.8053e-01,  7.8095e-01,  3.8546e-01,  4.3693e-01,  6.7797e-01,
          1.6995e-01,  1.6459e-01,  5.5182e-01,  3.5776e-01, -5.6186e-01,
```

Perbedaan

Aspek	TF-IDF	Static Word Embedding	BERT (Contextual Embedding)
Sifat Representasi	Sparse (vektor besar & banyak nol)	Dense (vektor kecil & padat)	Dense & Dinamis (kontekstual)
Menangkap Makna?	❌ Tidak	✅ Ya (mirip kata, mirip vektor)	✅✅ Ya, dan tergantung konteks
Ukuran vektor	Jumlah fitur unik (besar)	100–300 dimensi per kata	768 / 1024 dimensi per kalimat/kata
Urutan Kata?	❌ Diabaikan	❌ Diabaikan	✅ Diperhitungkan
Kata sama = vektor sama?	✅ Ya	✅ Ya	❌ Tidak (tergantung konteks kalimat)
Contoh Output	“bagus” selalu sama	“bagus” selalu sama	“bagus” bisa beda tergantung kalimat
Pre-trained tersedia?	❌ Tidak	✅ Word2Vec, GloVe, FastText	✅ IndoBERT, BERT multilingual, dll
Komputasi	Ringan	Sedang	Berat (butuh GPU idealnya)
Digunakan di	ML klasik (NB, SVM, LogReg)	Deep Learning (LSTM, CNN, dsb)	Fine-tuning BERT atau encoder DL

Modelling & Evaluation

Traditional ML

Naive Bayes

SVM

Random Forrest

SVM

```python
from sklearn.svm import SVC
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline

# Pipeline: ekstraksi fitur + model
svm_pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(min_df=2, max_df=0.8)),
    ('classifier', SVC(kernel='linear', C=1.0))
])

# Latih model
svm_pipeline.fit(X_train, y_train)

# Prediksi
svm_predictions = svm_pipeline.predict(X_test)
print("Prediksi SVM:", svm_predictions)
```

NB

```python
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split

# Data contoh
texts = [
    "I love this movie", "Great film, highly recommended",
    "Terrible movie, waste of time", "I hated this film",
    "Okay movie, nothing special", "Average film experience"
]
labels = [1, 1, 0, 0, 0.5, 0.5]  # 1=positif, 0=negatif, 0.5=netral

# Split data
X_train, X_test, y_train, y_test = train_test_split(texts, labels, test_size=0.3, random_state=42)

# Buat pipeline: ekstraksi fitur + model
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(min_df=2, max_df=0.8)),
    ('classifier', MultinomialNB())
])

# Latih model
pipeline.fit(X_train, y_train)

# Prediksi
predictions = pipeline.predict(X_test)
print("Prediksi:", predictions)
```

Random Forest

```python
from sklearn.ensemble import RandomForestClassifier

# Pipeline: ekstraksi fitur + model
rf_pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(min_df=2, max_df=0.8)),
    ('classifier', RandomForestClassifier(n_estimators=100, random_state=42))
])

# Latih model
rf_pipeline.fit(X_train, y_train)

# Prediksi
rf_predictions = rf_pipeline.predict(X_test)
print("Prediksi Random Forest:", rf_predictions)
```

Deep Learning

Recurrent Neural Networks (RNN) | Vanilla RNN, LSTM, dan GRU

Convolutional Neural Networks (CNN)

Transformer Models | BERT, RoBERTa, GPT, XLNet, etc

BERT

```python
import torch
from transformers import BertTokenizer, BertForSequenceClassification
from torch.utils.data import TensorDataset, DataLoader, RandomSampler
import numpy as np

# Data contoh
texts = [
    "I love this movie", "Great film, highly recommended",
    "Terrible movie, waste of time", "I hated this film",
    "Okay movie, nothing special", "Average film experience"
]
labels = np.array([1, 1, 0, 0, 0.5, 0.5])  # 1=positif, 0=negatif, 0.5=netral

# Tokenisasi untuk BERT
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Tokenize dan encode sequences
input_ids = []
attention_masks = []

for text in texts:
    encoded_dict = tokenizer.encode_plus(
        text,
        add_special_tokens=True,
        max_length=64,
        pad_to_max_length=True,
        return_attention_mask=True,
        return_tensors='pt',
    )

    input_ids.append(encoded_dict['input_ids'])
    attention_masks.append(encoded_dict['attention_mask'])

# Convert lists ke tensors
input_ids = torch.cat(input_ids, dim=0)
attention_masks = torch.cat(attention_masks, dim=0)
labels = torch.tensor(labels)

# Split data
dataset = TensorDataset(input_ids, attention_masks, labels)
train_dataset, test_dataset = torch.utils.data.random_split(dataset, [4, 2])

batch_size = 2
train_dataloader = DataLoader(
    train_dataset,
    sampler=RandomSampler(train_dataset),
    batch_size=batch_size
)
test_dataloader = DataLoader(
    test_dataset,
    batch_size=batch_size
)

# Load BERT model pre-trained
model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased",
    num_labels=1,
    output_attentions=False,
    output_hidden_states=False,
)

# Setup parameter untuk fine-tuning
optimizer = torch.optim.AdamW(model.parameters(), lr=2e-5)

# Latih model (kode latihan lengkap memerlukan loop epoch)
print("BERT model loaded and ready for fine-tuning")
```

Hands-On!

Notebook Link:

Traditional ML Approach

Deep Learning (BERT) Approach

BARCODE

ABSENSI

EDM Laboratory

Presensi GWE #7

https://tel-u.ac.id/presensigwe

Enterprise
Data
Management
Laboratory

Thank You

GWE #7