Summary :
Intro to Text Mining NLP

by : Dhea Fajriati Anas

Data Source: https://raw.githubusercontent.com/hendrip8/Data-Fellowship-IYKRA/main/NLP/tweet_covid_dataset.csv

iykra

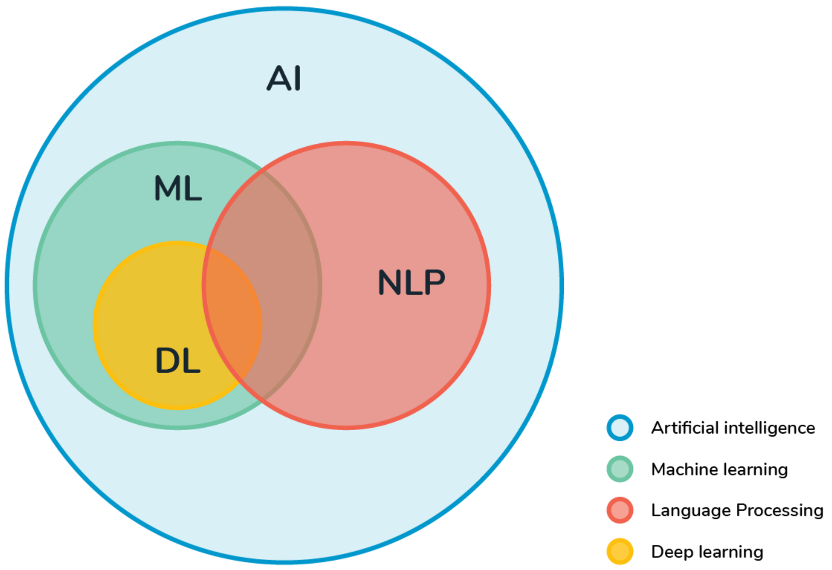
Table of Content

01

What is Text Mining and NLP ?

★★★★★

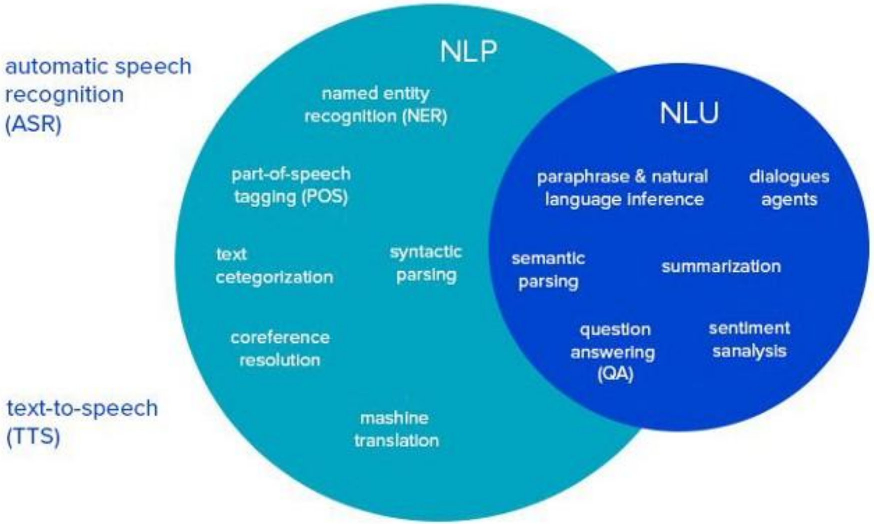
DEFINITION

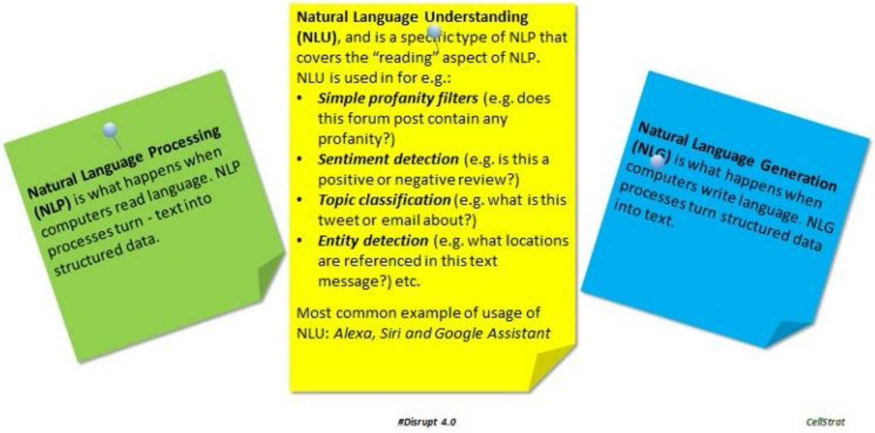


- **Text Analysis (a.k.a Text Mining)** : it's the process of understanding and sorting text, making it easier to manage. Text analysis could possibly be the last piece of the puzzle of growth every business is trying to solve. After all, in the information-saturated era we live in, what can be of more value than the organising of this information in a structured and meaningful way that we humans can understand.
- **Natural language processing (NLP)** : it's a subfield of AI concerned with the interactions between computers and human (natural) languages, in particular how to program computers to understand, interpret and manipulate human language.

Source: https://www.sentisum.com/library/nlp-and-text-mining

NLP VS NLU VS NLG





NLP = NLU + NLG

NLP — Natural Language "**Processing"**
NLU — Natural Language "**Understanding"**
NLG — Natural Language "**Generation"**

Source: https://towardsdatascience.com/nlp-vs-nlu-vs-nlg-know-what-you-are-trying-to-achieve-nlp-engine-part-1-1487a2c8b696

02

Techniques on Text Processing

TEXT PRE-PROCESSING

In NLP, text pre-processing is the first step in the process of building a model. The various text preprocessing steps are:

- **Case folding/ lower case**
  Converting a word to lower case.
- **Remove number**
  Delete numbers if they are not relevant to what you are going to analyze, such as house numbers, telephone numbers, etc.
- **Remove whitespace**
- **Remove punctuation**
  Remove punctuation like [!”#$%&’()*+,-./:;<=>?@[\]^_`{|}~].
- **Tokenizing**
  Splitting the sentence into words.
- **Stopword removal**
  Stopwords are common words that usually appear in large numbers and are considered meaningless. Examples of stopwords in Indonesian are "yang", “dan", "di", “dari", etc.
- **Stemming**
  Stemming is the process of removing the inflection of a word to its basic form, but the basic form does not mean the same as the root word. For example the words “mendengarkan", “dengarkan", “didengarkan" will be transformed into the word “dengar".

Regular Expression (Regex)

Common Regular Expression Syntax

Character	Description	Example
[]	A set of characters	"[a-m]"
\	Signals a special sequence (can also be used to escape special characters)	"\d"
.	Any character (except newline character)	"he..o"
^	Starts with	"^hello"
$	Ends with	"world$"
*	Zero or more occurrences	"aix*"
+	One or more occurrences	"aix+"
{}	Exactly the specified number of occurrences	"al{2}"
|	Either or	"falls|stays"
()	Capture and group	

Set	Description
[arn]	Returns a match where one of the specified characters (a , r , or n) are present
[a-n]	Returns a match for any lower case character, alphabetically between a and n
[^arn]	Returns a match for any character EXCEPT a , r , and n
[0123]	Returns a match where any of the specified digits (0 , 1 , 2 , or 3) are present
[0-9]	Returns a match for any digit between 0 and 9
[0-5][0-9]	Returns a match for any two-digit numbers from 00 and 59
[a-zA-Z]	Returns a match for any character alphabetically between a and z , lower case OR upper case
[+]	In sets, + , * , . , | , () , $, {} has no special meaning, so [+] means: return a match for any + character in the string

EXAMPLE : Regex

Syntax

```python
def clean_data(text):
    text = re.sub('@([a-zA-Z0-9_]+)', '', text) #remove @mention
    text = re.sub('#[\s]+', '', text) #remove hashtag
    text = re.sub('RT[\s]+', '', text) #remove RT
    text = re.sub('https?:\/\/\S+', '', text) #remove hyperlink
    text = re.sub('\d+', '', text) #remove number
    text = re.sub('[^\w\s]', '', text) #remove punctuations
    text = re.sub(r'\b[a-zA-Z]\b', '', text) #remove single character
    text = re.sub('\n', '', text) #remove \n
    text = re.sub('\r', '', text) #remove \r
    text = text.lower() #lowercase
    text = text.strip() #remove whitespace
    return text
```

```python
df['Tweet'] = df['Tweet'].apply(clean_data)
```

Before

```
0      @jokowi Kami sekeluarga dari awal covid sdh pr...
1      Pemerintah 'Nunggak' Bayar Klaim Covid-19 ke R...
2      @CTNurza @DoktorSamhan Masalahnya manusia yg t...
3      Ketawa saja bung @Dennysiregar7 , mereka itu o...
4      Aku iki cuma overthinking ae. Gumun juga klo n...
                              ...
995    @CNNIndonesia Dalam keadaan darurat, prosesnya...
996    @zouloutchaaaing rohi diri lvaccin hari denya ...
997    TNI-Polri bagikan Masker kepada masyarakat gun...
998    Hallo Sobat Polri... anak-anak sangat rentan t...
999    Bupati Karawang Cellica Nurrachadiana Kembali ...
Name: Tweet, Length: 1000, dtype: object
```

After

```
0      kami sekeluarga dari awal covid sdh prokes pak...
1          pemerintah nunggak bayar klaim covid ke rs rp
2      masalahnya manusia yg tak berakal tidak berota...
3      ketawa saja bung   mereka itu orangorang yg ku...
4      aku iki cuma overthinking ae gumun juga klo nd...
                              ...
995    dalam keadaan darurat prosesnya jangan lama bu...
996           rohi diri lvaccin hari denya kaml bl covid
997    tnipolri bagikan masker kepada masyarakat guna...
998    hallo sobat polri anakanak sangat rentan terha...
999    bupati karawang cellica nurrachadiana kembali ...
```

EXAMPLE : Tokenizing and Stopword

TOKENIZING

Library : nltk

Syntax

```
from nltk.tokenize import word_tokenize
raw = df['Tweet']
dff = [word_tokenize(paragraf) for paragraf in raw]
```

Output

```
[['kami',
  'sekeluarga',
  'dari',
  'awal',
  'covid',
  'sdh',
  'prokes',
  'pak',
  'tapi',
  'usaha',
```

STOPWORD

Library : nltk

Syntax

```
from nltk.corpus import stopwords
indo = stopwords.words('indonesian')
indo.extend(['yg','nya', 'dgn', 'dg', 'dr', 'ya', 'yaa',
             'aja', 'utk', 'ni', 'tp', 'amp', 'dah', 'krn'
             ,'udah'])
indo.extend(slang_)
indo.extend(formal_)
hasil_stopword = []
for i in range(len(dff)):
    data = [word for word in dff[i][:] if word not in indo]
    hasil_stopword.append(data)

hasil_join = []
for join in range(len(hasil_stopword)):
    hasil_join.append(' '.join(hasil_stopword[join]))
clean_data = hasil_join
```

Number of word before and after using stopword in row 0

```
print('before :'+str(len(dff[0])))
print('after :'+str(len(hasil_stopword[0])))
```

```
before :39
after :18
```

	slang	formal
0	woww	wow
1	aminn	amin
2	met	selamat
3	netaas	menetas
4	keberpa	keberapa
...	...	...
15001	gataunya	enggak taunya
15002	gtau	enggak tau
15003	gatau	enggak tau
15004	fans2	fan-fan
15005	gaharus	enggak harus

Source:
https://github.com/nasalsabila/kamus-alay/blob/master/colloquial-indonesian-lexicon.csv

03

Feature Engineering Techiques on Text

★★★★★

FEATURE ENGINEERING TECHIQUES

Feature engineering techiques :

- **Bag of Word (BoW)**
- **Term Frequency-Inverse Document Frequency (TF-IDF)**
- **Part of Speech (POS) – Tagging**
- **Named Entity Relation (NER)**

BAG OF WORDS (BoW) : Definition

The bag-of-words (BOW) model is a representation that turns arbitrary text into fixed-length vectors by counting how many times each word appears. This process is often referred to as vectorization. For example we have this sentence, and all we have to do is count how many times each word appears :

Document	the	cat	sat	in	hat	with
the cat sat	1	1	1	0	0	0
the cat sat in the hat	2	1	1	1	1	0
the cat with the hat	2	1	0	0	1	1

Notice that we lose contextual information, e.g. where in the document the word appeared, when we use BOW. It's like a literal bag-of-words: it only tells you *what* words occur in the document, not *where* they occurred.

Source: https://machinelearningmastery.com/gentle-introduction-bag-words-model/

BAG OF WORDS (BoW) : Hands On

Library : sklearn

Syntax

```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
x_sentence = cv.fit_transform(clean_data)
```

Output

	aa	aamiintambahimun	aamiintruestory	abah	abai	abaikan	abamaze	abang	abdurachman	abiszzzz	...	yustisi	yusuf	yuuuuu	zaman	zayed	zodiak
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
996	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
997	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
998	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
999	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0

1000 rows × 4782 columns

TF-IDF : Definition

TF-IDF was invented for document search and information retrieval. It works by increasing proportionally to the number of times a word appears in a document, but is offset by the number of documents that contain the word. So, words that are common in every document, such as this, what, and if, rank low even though they may appear many times, since they don't mean much to that document in particular.:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j

df_i = number of documents containing i

N = total number of documents

Source: https://monkeylearn.com/blog/what-is-tf-idf/

TF-IDF : Hands On

Library : sklearn
Syntax

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer()
x2_sentence = tfidf.fit_transform(clean_data)
```

Output

	aa	aamiintambahimun	aamiintruestory	abah	abai	abaikan	abamaze	abang	abdurachman	abiszzzz	...	yustisi	yusuf	yuuuuu	zaman	zayed	zodiak
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
996	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
997	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
998	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0
999	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0

1000 rows × 4782 columns

POS : Definition

POS Tag is a way of categorizing word classes, such as nouns, verbs, adjectives, etc. POS Tagger is an application that is able to automatically annotate part-of-speech tags for each word in a document.

Keterangan label /tagger:

ADJ : kata sifat
ADP : preposisi
ADV : keterangan
AUX : kata bantu
CCONJ : kata penghubung
INTJ : kata seru
NOUN : kata benda
NUM : angka
PART : partikel
PRON : kata ganti
PUNCT : tanda baca
SYM : simbol
VERB : kata kerja
X : lainnya

Source: https://monkeylearn.com/blog/what-is-tf-idf/

POS : Hands On

To do POS-tagging, we need to create a POS-Tagger consisting of word embedding and a dictionary. Simply put, word embedding is a representation of words into a vector. The library in this tagger is built from a corpus (a collection of words) that has been marked.

1. **Load Corpus**

```
from flair.data_fetcher import NLPTaskDataFetcher, NLPTask
corpus = NLPTaskDataFetcher.load_corpus(NLPTask.UD_INDONESIAN)
```

2. **Make Library from Corpus**

   Since the purpose of tagging here is to determine part-of-speech, the tag_type selected is 'upos'.

```
tag_type = 'upos'
tag_dictionary = corpus.make_tag_dictionary(tag_type=tag_type)
```

POS : Hands On

3. **Choose Word Embedding**
   Actually there are many types of embedding, but for Indonesian, it is better to use special Indonesian embedding, namely 'id-crawl' and 'id'. Here I combine two embedding (stacking), generally one embedding is enough.

```
from flair.embeddings import TokenEmbeddings, WordEmbeddings, StackedEmbeddings, BertEmbeddings
from typing import List

embedding_types: List[TokenEmbeddings] = [
 WordEmbeddings('id-crawl'),
 WordEmbeddings('id'),
]
embeddings: StackedEmbeddings = StackedEmbeddings(embeddings=embedding_types)
```

4. **Combining embedding and libraries**

```
from flair.models import SequenceTagger
tagger: SequenceTagger = SequenceTagger(hidden_size=256,
                                        embeddings=embeddings,
                                       tag_dictionary=tag_dictionary,
                                        tag_type=tag_type,
                                        use_crf=True)
```

POS : Hands On

5. **Conducting model training for taggers**
   This training process is a computational process that is quite heavy and long if it is run on a normal computer, therefore it is recommended to do training in the cloud (eg. using Google Colab).

```
from flair.trainers import ModelTrainer

trainer: ModelTrainer = ModelTrainer(tagger, corpus)
trainer.train('resources/taggers/example-universal-pos',learning_rate=0.1,
              mini_batch_size=32,max_epochs=5)
```

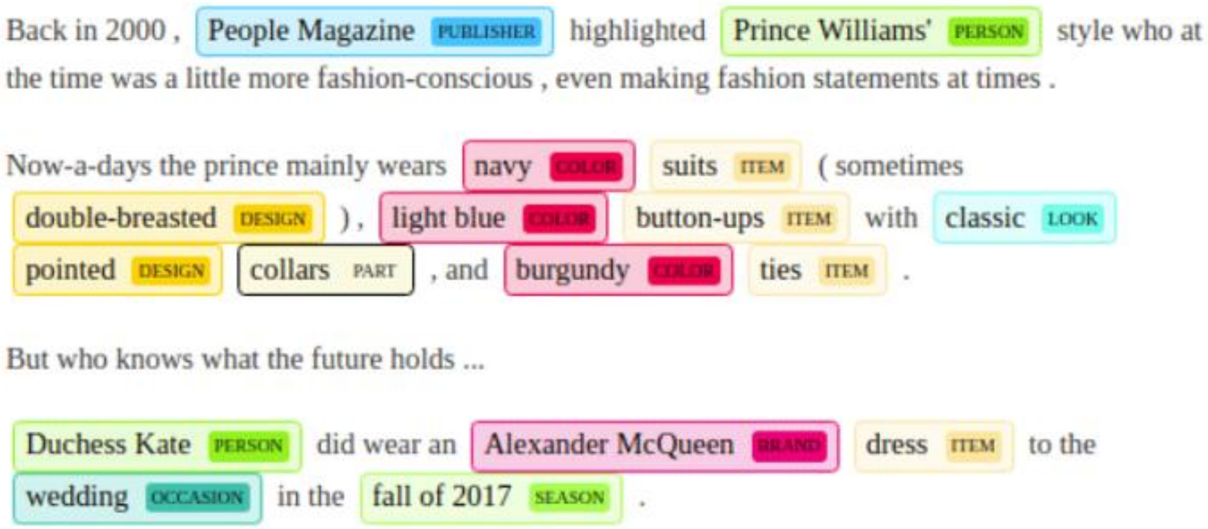
6. **Result**

```
from flair.data import Sentence
sentence = Sentence('arahan menteri desa dalam penangan covid')
tag_pos = SequenceTagger.load('resources/taggers/example-universal-pos/best-model.pt')
tag_pos.predict(sentence)

Output:
arahan <NOUN> menteri <NOUN> desa <NOUN> dalam <ADP> penangan <NOUN> covid <PUNCT>
```

NER : Definition

NER is technique of information extraction of the text which classify word into its predefined categories, such as name of person, or organizations, locations, expressions of times, quantities, monetary values, percentages, etc.



Source: https://yudanta.github.io/posts/train-an-indonesian-ner-from-a-blank-spacy-model/

NER : Hands On

1. **Import Library**

```
import pickle
import spacy
import random
from spacy.util import minibatch, compounding
from spacy import load, displacy
```

2. **Open Data Train**

```
with open('/content/sample_data/ner_spacy_fmt_datasets.pickle', 'rb') as f:
    ner_spacy_fmt_datasets = pickle.load(f)
```

3. **Next we will create a model from “empty model” or “blank-model” with the following command**

```
nlp = spacy.blank("id")
nlp.add_pipe(nlp.create_pipe('ner'))
nlp.begin_training()
```

NER : Hands On

4. **We take the data-module for NER and set the others aside**

```
ner = nlp.get_pipe("ner")
pipe_exceptions = ["ner", "trf_wordpiecer", "trf_tok2vec"]
unaffected_pipes = [pipe for pipe in nlp.pipe_names if pipe not in pipe_exceptions]
```

5. **We process or retrieve entity type labels from training data**

```
for _, annotations in ner_spacy_fmt_datasets:
    for ent in annotations.get("entities"):
        ner.add_label(ent[2])
        break
```

NER : Hands On

6. **We will randomize the train data first and then enter it into the training iteration, in this example we try to train the model 5 times**

```
# TRAINING THE MODEL
with nlp.disable_pipes(*unaffected_pipes):
  # Training for 5 iterations
  for iteration in range(5):
    # shuufling examples  before every iteration
    random.shuffle(ner_spacy_fmt_datasets)
    losses = {}
    # batch up the examples using spaCy's minibatch
    batches = minibatch(ner_spacy_fmt_datasets, size=compounding(4.0, 32.0, 1.001))
    for batch in batches:
        texts, annotations = zip(*batch)
        nlp.update(
                    texts,  # batch of texts
                    annotations,  # batch of annotations
                    drop=0.5,  # dropout - make it harder to memorise data
                    losses=losses,
                )
    print("Losses at iteration {}".format(iteration), losses)
```

NER : Hands On

7. **After the model learning process is complete, to try the model that we have trained, we can use the following code**

```
# test
doc = nlp("Arahan menteri Desa dalam penangan Covid")
print(doc.ents)
print("Entities", [(ent.text, ent.label_) for ent in doc.ents])
```

Output

```
Desa,)
Entities [('Desa', 'ORGANIZATION')]
```

THANK YOU

CREDITS: Diese Präsentationsvorlage wurde von **Slidesgo** erstellt, inklusive Icons von **Flaticon**, Infografiken & Bilder von **Freepik**

Bitte lösche diese Folie nicht, es sei denn du bist ein Premium Nutzer