Silahkan cari tutorial lain yang membahas tutorial teks mining, lalu buat ringkasannya dan tulis dalam format ms word dan dikumpulkan sebelum batas waktu pengumpulan berakhir.

Nama : I Gede Ery Santika

NIM : 202420012

Text Mining adalah adalah proses ekstraksi pola berupa informasi dan pengetahuan yang berguna dari sejumlah besar sumber data teks, seperti dokumen Word, PDF, kutipanteks, dll.

Text mining merupakan penerapan konsep dan teknik data mining untuk mencari pola dalam teks, yaitu proses penganalisisan teks guna menyarikan informasi yang bermanfaat untuk tujuan tertentu.

Tujuan dari text mining adalah untuk mendapatkan informasi yang berguna dari sekumpulan dokumen. Jadi, sumber data yang digunakan pada text mining adalah kumpulan teks yang memiliki format yang tidak terstruktur atau minimal semi terstruktur. Adapun tugas khusus dari text mining antara lain yaitu pengkategorisasian teks (text categorization) dan pengelompokan teks text clustering).

1. Case folding

Case folding adalah salah satu bentuk text preprocessing yang paling sederhana dan efektif meskipun sering diabaikan. Tujuan dari case folding untuk mengubah semua huruf dalam dokumen menjadi huruf kecil. Hanya huruf ‘a’ sampai ‘z’ yang diterima. Karakter selain huruf dihilangkan dan dianggap delimiter. Pada tahap ini tidak menggunakan external library apapun, kita bisa memanfaatkan modul yang tersedia di python.

Ada beberapa cara yang dapat digunakan dalam tahap case folding, anda dapat menggunakan beberapa atau menggunakan semuanya, tergantung pada tugas yang diberikan.

- Mengubah text menjadi lowercase

  Salah satu contoh pentingnya penggunaan lower case adalah untuk mesin pencarian. Bayangkan anda sedang mencari dokumen yang mengandung “indonesia” namun tidak ada hasil yang muncul karena “indonesia” di indeks sebagai “INDONESIA”. Siapa yang harus kita salahkan? U.I. designer yang mengatur user interface atau developer yang mengatur sistem dalam indeks pencarian?

Contoh dibawah menunjukan bagaimana python mengubah teks menjadi lowercase :

```
kalimat = "Berikut ini adalah 5 negara dengan pendidikan terbaik di
dunia adalah Korea Selatan, Jepang, Singapura, Hong Kong, dan
Finlandia."lower_case = kalimat.lower()
print(lower_case)# output
# berikut ini adalah 5 negara dengan pendidikan terbaik di dunia
adalah korea selatan, jepang, singapura, hong kong, dan finlandia.
```

- Menghapus angka

  Hapuslah angka jika tidak relevan dengan apa yang akan anda analisa, contohnya seperti nomor rumah, nomor telepon, dll. Regular expression (regex) dapat digunakan untuk

menghapus karakter angka. Python memiliki modulre untuk melakukan hal – hal yang berkaitan dengan regex.

Contoh dibawah menunjukan bagaimana python menghapus angka dalam sebuah kalimat :

```
import re # impor modul regular expressionkalimat = "Berikut ini
adalah 5 negara dengan pendidikan terbaik di dunia adalah Korea
Selatan, Jepang, Singapura, Hong Kong, dan Finlandia."
hasil = re.sub(r"\d+", "", kalimat)
print(hasil)# ouput
# Berikut ini adalah negara dengan pendidikan terbaik di dunia
adalah Korea Selatan, Jepang, Singapura, Hong Kong, dan Finlandia.
```

- Menghapus tanda baca

  Sama halnya dengan angka, tanda baca dalam kalimat tidak memiliki pengaruh pada text preprocessing. Menghapus tanda baca seperti [!”#$%&’()*+,-./:;<=>?@[\]^_`{|}~] dapat dilakukan di pyhton seperti dibawah ini :

```
kalimat = "Ini &adalah [contoh] kalimat? {dengan} tanda. baca?!!"
hasil = kalimat.translate(str.maketrans("","",string.punctuation))
print(hasil)# output
# Ini adalah contoh kalimat dengan tanda baca
```

- Menghapus whitepace (karakter kosong)

  Untuk menghapus spasi di awal dan akhir, anda dapat menggunakan fungsi strip()pada pyhton. Perhatikan kode dibawah ini :

```
kalimat = " \t ini kalimat contoh\t "
hasil = kalimat.strip()
print(hasil)# output
# ini kalimat contoh
```

2. Tokenizing

   Tokenizing adalah proses pemisahan teks menjadi potongan-potongan yang disebut sebagai token untuk kemudian di analisa. Kata, angka, simbol, tanda baca dan entitas penting lainnya dapat dianggap sebagai token. Didalam NLP, token diartikan sebagai “kata” meskipun tokenize juga dapat dilakukan pada paragraf maupun kalimat.

   Fungsi split()pada pyhton dapat digunakan untuk memisahkan teks. Perhatikan contoh dibawah ini :

```
kalimat = "rumah idaman adalah rumah yang bersih"
pisah = kalimat.split()
print(pisah)# output
# ['rumah', 'idaman', 'adalah', 'rumah', 'yang', 'bersih']
```

- Tokenizing kata

Sebuah kalimat atau data dapat dipisah menjadi kata-kata dengan kelas word_tokenize() pada modul NLTK.

```
# impor word_tokenize dari modul nltk
from nltk.tokenize import word_tokenize

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau
online."

tokens = nltk.tokenize.word_tokenize(kalimat)
print(tokens)# ouput
# ['Andi', 'kerap', 'melakukan', 'transaksi', 'rutin', 'secara',
'daring', 'atau', 'online', '.']
```

Dari output kode diatas terdapat kemunculan tanda baca titik(.) dan koma (,) serta token "Andi" yang masih menggunakan huruf besar pada awal kata. Hal tersebut nantinya dapat menggangu proses perhitungan dalam penerapan algoritma. Jadi, sebaiknya teks telah melewati tahap case folding sebelum di tokenize agar menghasilkan hasil yang lebih konsisten.

Kita dapat menambahkan fungsi case folding untuk menghilangkan tanda baca dan mengubah teks ke dalam bentuk lowercase.

```
kalimat =
kalimat.translate(str.maketrans('','',string.punctuation)).lower()#
output
# ['andi', 'kerap', 'melakukan', 'transaksi', 'rutin', 'secara',
'daring', 'atau', 'online']
```

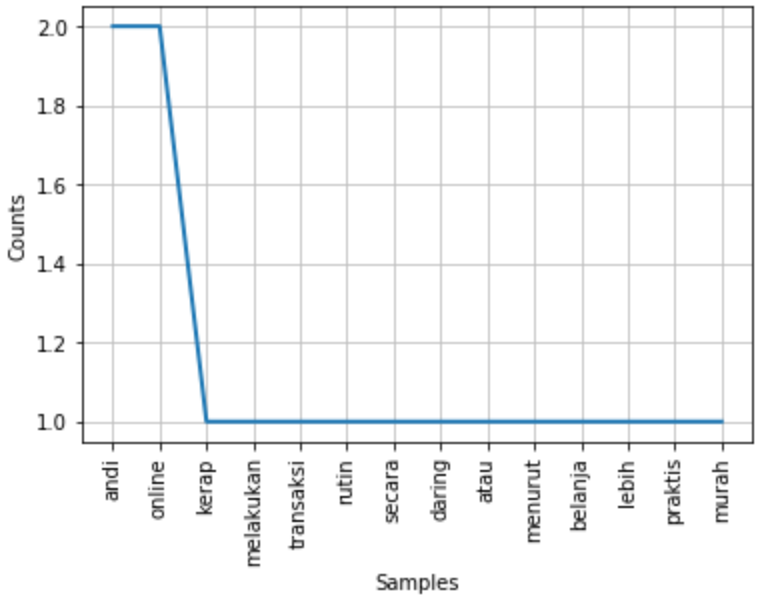
Kita juga bisa mendapatkan informasi frekuensi kemunculan setiap token dengan kelas FreqDist() yang sudah tersedia pada modul NLTK.

```
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDistkalimat = "Andi kerap
melakukan transaksi rutin secara daring atau online. Menurut Andi
belanja online lebih praktis & murah."
kalimat =
kalimat.translate(str.maketrans('','',string.punctuation)).lower()

tokens = nltk.tokenize.word_tokenize(kalimat)
kemunculan = nltk.FreqDist(tokens)
print(kemunculan.most_common())# output
# [('andi', 2), ('online', 2), ('kerap', 1), ('melakukan', 1),
('transaksi', 1), ('rutin', 1), ('secara', 1), ('daring', 1),
```

```
('atau', 1), ('menurut', 1), ('belanja', 1), ('lebih', 1),
('praktis', 1), ('murah', 1)]
```

Untuk menggambarkan ke dalam bentuk grafik, kita perlu menginstall library Matplotlib.

```
import matplotlib.pyplot as pltkemunculan.plot(30,cumulative=False)
plt.show()
```



- Tokenizing kalimat

Prinsip yang sama dapat diterapkan untuk memisahkan kalimat pada paragraf. Anda dapat menggunkan kelas sent_tokenize() pada modul NLTK. Saya telah menambahkan kalimat pada contoh seperti dibawah ini :

```
# impor sent_tokenize dari modul nltk
from nltk.tokenize import sent_tokenizekalimat = "Andi kerap
melakukan transaksi rutin secara daring atau online. Menurut Andi
belanja online lebih praktis & murah."

tokens = nltk.tokenize.sent_tokenize(kalimat)
print(tokens)# ouput
# ['Andi kerap melakukan transaksi rutin secara daring atau
online.', 'Menurut Andi belanja online lebih praktis & murah.']
```

NIM : 202420040 ISTIARSO

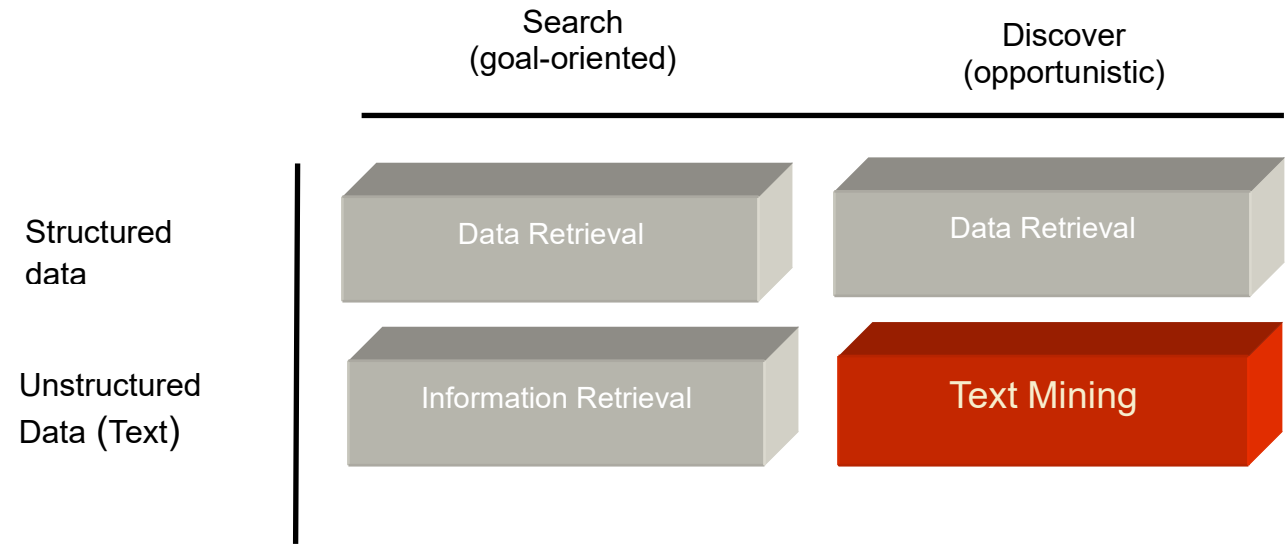
TEXT Mining

Pengertian Text Mining

Text mining merupakan penerapan konsep dan teknik data mining untuk mencari pola dalam teks

- proses penganalisisan teks guna menyarikan informasi yang bermanfaat untuk tujuan tertentu.

Proses data mining untuk data dokumen atau teks memerlukan lebih banyak tahapan, mengingat data teks memiliki karakteristik yang lebih kompleks daripada data biasa.



Text Mining dapat didefinisikan sebagai “penemuan informasi baru dan tidak diketahui sebelumnya oleh computer, dengan secara otomatis mengekstrak informasi dari sumber – sumber teks tak terstruktur yang berbeda” (Tan, 1999). Definisi singkatnya adalah suatu proses menganilisa teks untuk mengekstrak informasi yang berguna untuk tujuan tertentu. Perbedaan mendasar dari text mining dan data mining terletak pada sumber data yang digunakan. Pada data mining data yang diekstrak berasal dari pola-pola tertentu dan terstruktur, sedangkam text mining sumber data yang digunakan berasal dari teks yang relatif tidak testruktur karena menggunakan tata Bahasa manusia atau biasa disebut (natural language). Secara umum basis data didesain untuk program dengan tujuan melakukan pemrosesan secara otomatis, sedangkan teks ditulis untuk dibaca langsung oleh manusia (Herast, 2003). Dalam text mining terdapat beberapa tahapan untuk memproses data teks tersebut

1. Case Folding dan Tokenizing Case Folding biasa disebut penyeragaman kata dengan cara mengubah seluruh kata menjadi huruf kecil (lowercase). Hanya huruf a sampai z yang dapat diterima karaktek selain huruf dihilangkan. Terdapat juga kata-kata tertentu yang harus sesuai dengan kaidah yang tidak bisa dilakukan penyeragaman kata sperti kata lembaga atau institusi yang selalui diawali huruf kapital dan juga nama gelar seperti halnya ST, M.Psi dan lain sebagainya. Tergantung dari

sumber data yang digunakan untuk diproses. Tokenizing adalah suatu tahapan pemotongan string kata berdasrakan penyusunan kata tersebut.
2. Filtering Filtering adalah pengambilan kata-kata penting dari hasil Tokenizing atau biasa disebut pengeliminasi sebuah kata-kata sesuai dengan kaidahnya. 7 Algoritma stop-word removal adalah salah satu yang digunakan untuk melakukan tahapan filtering
3. Stemming Stemming adalah suatu proses untuk memecah sutau varian-varian kata menjadi kata dasar sesuai dengan kata yang sedang diproses. Jika kata yang diproses adalah Bahasa Indonesia untuk memecah varian kata menjadi kata dasar harus sesuai dengan aturan Bahasa Indonesia salah satu algoritma yang digunakan adalah Nazief & Adriani.
4. Analyzing Analyzing adalah suatu tahapan menganlisa data teks yang sedang diproses untuk menentukan kemiripan antar dokumen teks salah satu metode yang digunakan adalah cosine similarity.

Karakteristik Dokumen Teks

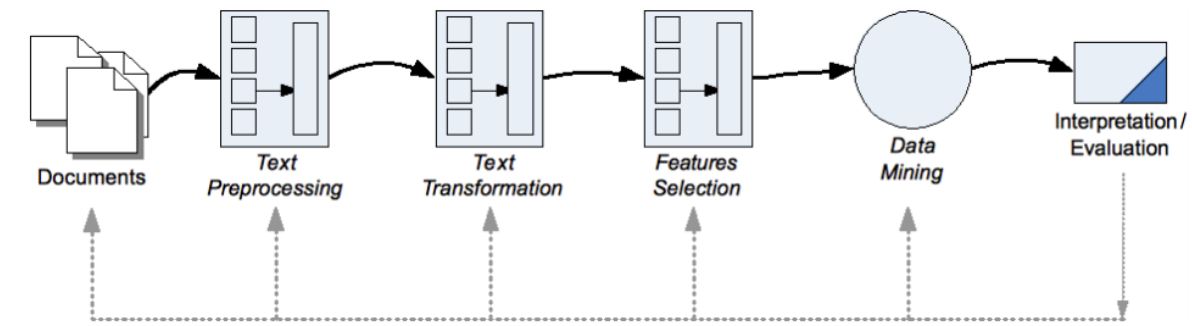
Menurut Loreta Auvil dan Duane Searsmith dari University of Illinois, karakteristik dokumen teks:

- ✓ database teks yang berukuran besar,
- ✓ memiliki dimensi yang tinggi, yakni satu kata merupakan satu dimensi,
- ✓ mengandung kumpulan kata yang saling terkait (frase) dan antara kumpulan kata satu dengan lain dapat memiliki arti yang berbeda,
- ✓ banyak mengandung kata ataupun arti yang bias (ambiguity),
- ✓ dokumen email merupakan dokumen yang tidak memiliki struktur bahasa yang baku, karena di dalamnya terkadang muncul istilah slank seperti ”r u there?”, ”hellllooo bosss, whatzzzzzzz up?”, dan sebagainya.

Proses Text Mining

Berdasarkan ketidakteraturan struktur data teks, maka proses text mining memerlukan beberapa tahap awal yang pada intinya adalah mempersiapkan agar teks dapat diubah menjadi lebih terstruktur.
Bentuk perubahan yang dilakukan adalah ke dalam spreadsheet, kolom menunjuk dokumen dan baris menunjuk kata, sedangkan selnya menunjuk frekuensi kata dalam dokumen.

NIM : 202420040 ISTIARSO



Proses text Mining

Yang di maksud Dokumen

- ✓ Plain text
- ✓ Format Elemen
  - XML, HTML, RTF, ODT, email, dsb.
- ✓ Format Biner
  - PDF, DOC, dsb.

Tokenisasi

Tokenisasi secara garis besar memecah sekumpulan karakter dalam suatu teks ke dalam satuan kata.

- ➢ bagaimana membedakan karakter-karakter tertentu yang dapat diperlakukan sebagai pemisah kata atau bukan.
- ➢ Sebagai contoh karakter whitespace, seperti enter, tabulasi, spasi dianggap sebagai pemisah kata.

Namun untuk karakter petik tunggal ('), titik (.), semikolon (;), titk dua (:) atau lainnya, dapat memiliki peran yang cukup banyak sebagai pemisah kata.

- Sebagai contoh antara “tahu, tempet dan sambal” dengan “100,56”.

Dalam memperlakukan karakter-karakter dalam teks sangat tergantung sekali pada kontek aplikasi yang dikembangkan.

Pekerjaan tokenisasi ini akan semakin sulit jika juga harus memperhatikan struktur bahasa (grammatikal).

Bagaimana tokenisasi menangani keadaan sbb:

a. Karakter Nonalphanumeric

   contoh: Yahoo!, AT&T, dsb.

b. Sebuah titik (.) biasanya untuk tanda akhir kalimat, tapi dapat juga muncul dalam singkatan, inisial orang, alamat internet
   Contoh: Sdr., S.Kom., 192.168.1.1, ukdw.ac.id
c. Tanda hypen (-) biasanya muncul untuk menggabungkan dua token yang berbeda untuk membentuk token tunggal. Tapi dapat pula ditemukan untuk menyatakan rentang nilai, kata berulang, dsb.
   Contoh: x-ray, 32-120, lari-lari.
d. Karakter slash (/) sebagai pemisah fle atau direktori atau url ataupun untuk menyatakan "dan atau"
   Contoh: /opt/rapidminer, www.google.com/search? num=100&q=text+mining, Ibu/Bapak.
e. URL.
f. Format nomor telepon.
g. Emoticon
h. Format angka

Lemmatization

Setelah deretan karakter telah disegmentasi ke dalam kata-kata (token), langkah berikut yang mungkin dilakukan adalah mengubah setiap token ke bentuk standard. Proses ini disebut menerapkan stemming dan atau lemmatization.

– Tujuan: untuk mendapatkan bentuk dasar umum dari suatu kata.

Contoh:

– Am, are, is => be

– Car, cars, car's, cars' => car

Stemming

Proses heuristic yang memotong akhir kata, dan sering juga membuang imbuhan.

Lemmatization

Serupa dengan stemming, hanya lebih baik hasilnya, karena :

– Memperhatikan kamus dan analisis morfologi.

– Menghasilkan kata dasar (lemma)

Porter Stemming

NIM : 202420040 ISTIARSO

Algoritma stemming Porter didasarkan pada ide bahwa akhiran dalam bahasa Inggris sebagian besar terbentuk dari kombinasi akhiran yang lebih kecil dan sederhana. Proses penanggalan dikerjakan pada serentetan langkah, yang mensimulasikan perubahan bentuk dan penurunan dari sebuah kata. Pada setiap langkah, sebuah akhiran tertentu dibuang berdasar aturan substitusi. Sebuah aturan substitusi diterapkan ketika sekumpulan kondisi/batasan untuk aturan tersebut terpenuhi. Salah satu contoh kondisi adalah jumlah minimal dari hasil stem (disebut juga ukuran (measure)). Kondisi sederhana lain dapat berupa apakah akhir dari stem konsonan atau vokal.

Porter Stemming

(F)	Rule			Example		
	SSES	→	SS	caresses	→	caress
	IES	→	I	ponies	→	poni
	SS	→	SS	caress	→	caress
	S	→		cats	→	cat

Porter Stemming Terdapat banyak aturan lain. Kunjungi:
htp://snowball.tartarus.org/algorithms/porter/stemmer.html

Contoh Perbandingan Stemmer

Contoh Perbandingan Stemmer

Sample text: Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Lovins stemmer: such an analys can reve featur that ar not eas vis from th vari in th individu gen and can lead to a pictur of expres that is mor biolog transpar and acces to interpres

Porter stemmer: such an analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to a pictur of express that is more biolog transpar and access to interpret

Paice stemmer: such an analys can rev feat that are not easy vis from the vary in the individ gen and can lead to a pict of express that is mor biolog transp and access to interpret

NIM : 202420040 ISTIARSO

http://www.cs.waikato.ac.nz/~eibe/stemmers/

http://www.comp.lancs.ac.uk/computing/research/stemming/

ZIPF's LAW

Kita menggunakan sedikit kata lebih sering dan jarang untuk sebagian besar kata lain. Rata-rata 20% kata-kata berperan sebagai mayoritas kata dalam suatu teks. Kita dapat memilih kata-kat sehingga kita mengkomunikasikan pesan dengan jumlah kata yang lebih sedikit.

the product of the frequency of a word and its rank will be approximately the same as the product of the frequency and rank of another word.

Membangun Vektor untuk Prediksi

Karakteristik ciri/sifat sebuah dokumen dinyatakan oleh token atau kata-kata di dalamnya.

```
Input:
  ts, all the tokens in the document collection
  k, the number of features desired
Output:
  fs, a set of k features
Initialize:
  hs := empty hashtable

for each tok in ts do
  If hs contains tok then
    i := value of tok in hs
    increment i by 1
  else
    i := 1
  endif
  store i as value of tok in hs
endfor
sk := keys in hs sorted by decreasing value
fs := top k keys in sk
output fs
```

Himpunan ciri-ciri yang terkumpul disebut sebagai kamus (dictionary). Token-token atau kata-kata dalam kamus membentuk dasar untuk membuat sebuah matrik angka yang sangat berkaitan dengan kumpulan dokumen yang di analisis. Sehingga, setiap sel berisi ukuran dari sebuah ciri/sifat (kolom)

untuk sebuah dokumen (baris). Dimensi kamus yang dihasilkan tentu saja akan berukuran sangat besar, sehingga perlu dilakukan proses transformasi untuk mengurangi ukuran dimensinya. Beberapa proses transformasi yang dapat diterapkan antara lain, Stopwords, Frequent Words, dan pengurangan token (Stemming atau Sinonim).

Mengubah Dokumen ke sebuah matrix

```
Input:
  fs, a set of k features
  dc, a collection of n documents
Output: ss, a spreadsheet with n rows and k columns
Initialize: i := 1

for each document d in dc, do
  j := 1
  for each feature f in fs, do
    m := number of occurrences of f in d
    if (m > 0) then ss(row=i, col=j) := 1;
    else ss(row=i, col=j) := 0 ;
    endif
    increment j by 1
  endfor
  increment i by 1
endfor
output ss
```

Untuk memberikan ketepatan prediksi, perlu dilakukan transformasi tambahan dengan :

- ✓ Menghitung tingkat peran kata dalam corpus.
- ✓ tf-idf (term frequency-inverse document frequency).

Metode tf-idf merupakan salah satu metode untuk menghitung bobot setiap kata yang digunakan. Pada penelitian sebelumnya dilakukan perhitungan dengan metode binary term sedangkan pada Metode tf-idf terkenal mudah, efisien dalam hal melakukan perhitungan bobot kemunculan kata pada sebuah dokumen dengan rumus sebagi berikut:

NIM : 202420040 ISTIARSO

$$tf\_wt_{t,d} = \begin{cases} 1 + \log_{10}(tf_{t,d}) \;\; Jika \;\; tf_{t,d} > 0 \\ 0 \qquad\qquad\qquad jika\, tf_{t,d} \leq 0 \end{cases}$$

$$idf_t = \log_{10}\left(\frac{N}{df_t}\right)$$

$$w_{t,d} = \left(1 + \log_{10}(tf_{t,d})\right) \times \log_{10}\left(\frac{N}{df_t}\right)$$

$$tf\_wt_{t,d} = \begin{cases} 1 + \log_{10}(tf_{t,d}) \;\; Jika \;\; tf_{t,d} > 0 \\ 0 \qquad\qquad\qquad jika\, tf_{t,d} \leq 0 \end{cases}$$

$$idf_t = \log_{10}\left(\frac{N}{df_t}\right)$$

$$w_{t,d} = \left(1 + \log_{10}(tf_{t,d})\right) \times \log_{10}\left(\frac{N}{df_t}\right)$$

$$w_{ij} = TFIDF\,(t_i, e_j) = TF_{i,j} \log \frac{N}{DF_i}$$

$$w_{ij} = \frac{TFIDF(t_i, e_j)}{\sqrt{\sum_{s=1}^{|V|} (TFIDF(t_s, e_j))^2}}$$

NIM : 202420040 ISTIARSO

Pengukuran Lain

Information Gain:

$$IG(t_i, c_j) = P(t_i, c_j) \log \frac{P(t_i, c_j)}{P(c_j)P(t_i)} + P(\bar{t}_i, c_i) \log \frac{P(\bar{t}_i, c_j)}{P(c_j)P(\bar{t}_i)}$$

Chi-Squared Measure:

$$x^2(t_i, c_j) = \frac{N[P(t_i, c_j)P(\bar{t}_i, \bar{c}_j) - P(t_i, \bar{c}_j)P(\bar{t}_i, c_j)]^2}{P(t_i)P(\bar{t}_i)P(c_j)P(\bar{c}_j)}$$

Contoh

```
D1      Human machine interface for computer applications
D2      A survey of user opinion of computer system response time
D3      The EPS user interface management system
D4      System and human system engineering testing of EPS
D5      The generation of random, binary and ordered trees
D6      The intersection graph of paths in trees
D7      Graph minors: A survey
```

Matrik Dokumen

```
=== Raw Term Frequencies ===
                    D1      D2      D3      D4      D5      D6      D7
          binary  0.0000  0.0000  0.0000  0.0000  1.0000  0.0000  0.0000
        computer  1.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
computer system   0.0000  1.0000  0.0000  0.0000  0.0000  0.0000  0.0000
     engineering  0.0000  0.0000  0.0000  1.0000  0.0000  0.0000  0.0000
             eps  0.0000  0.0000  1.0000  1.0000  0.0000  0.0000  0.0000
      generation  0.0000  0.0000  0.0000  0.0000  1.0000  0.0000  0.0000
           graph  0.0000  0.0000  0.0000  0.0000  0.0000  1.0000  1.0000
           human  1.0000  0.0000  0.0000  1.0000  0.0000  0.0000  0.0000
       interface  1.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    intersection  0.0000  0.0000  0.0000  0.0000  0.0000  1.0000  0.0000
         machine  1.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
      management  0.0000  0.0000  1.0000  0.0000  0.0000  0.0000  0.0000
           minors  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  1.0000
         opinion  0.0000  1.0000  0.0000  0.0000  0.0000  0.0000  0.0000
         ordered  0.0000  0.0000  0.0000  0.0000  1.0000  0.0000  0.0000
          random  0.0000  0.0000  0.0000  0.0000  1.0000  0.0000  0.0000
        response  0.0000  1.0000  0.0000  0.0000  0.0000  0.0000  0.0000
          survey  0.0000  1.0000  0.0000  0.0000  0.0000  0.0000  1.0000
          system  0.0000  0.0000  1.0000  2.0000  0.0000  0.0000  0.0000
         testing  0.0000  0.0000  0.0000  1.0000  0.0000  0.0000  0.0000
            time  0.0000  1.0000  0.0000  0.0000  0.0000  0.0000  0.0000
            user  0.0000  1.0000  0.0000  0.0000  0.0000  0.0000  0.0000
 user interface   0.0000  0.0000  1.0000  0.0000  0.0000  0.0000  0.0000
```

Matrik Dokumen TF-IDF

NIM : 202420040 ISTIARSO

```
=== Inverse Document Frequency ===
                     D1      D2      D3      D4      D5      D6      D7
          binary  0.0000  0.0000  0.0000  0.0000  0.2500  0.0000  0.0000
        computer  0.2656  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
computer system  0.0000  0.1735  0.0000  0.0000  0.0000  0.0000  0.0000
     engineering  0.0000  0.0000  0.0000  0.1977  0.0000  0.0000  0.0000
             eps  0.0000  0.0000  0.2167  0.1512  0.0000  0.0000  0.0000
      generation  0.0000  0.0000  0.0000  0.0000  0.2500  0.0000  0.0000
           graph  0.0000  0.0000  0.0000  0.0000  0.0000  0.4333  0.3023
           human  0.2031  0.0000  0.0000  0.1512  0.0000  0.0000  0.0000
       interface  0.2656  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
    intersection  0.0000  0.0000  0.0000  0.0000  0.0000  0.5667  0.0000
         machine  0.2656  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
      management  0.0000  0.0000  0.2833  0.0000  0.0000  0.0000  0.0000
          minors  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.3953
         opinion  0.0000  0.1735  0.0000  0.0000  0.0000  0.0000  0.0000
         ordered  0.0000  0.0000  0.0000  0.0000  0.2500  0.0000  0.0000
          random  0.0000  0.0000  0.0000  0.0000  0.2500  0.0000  0.0000
        response  0.0000  0.1735  0.0000  0.0000  0.0000  0.0000  0.0000
          survey  0.0000  0.1327  0.0000  0.0000  0.0000  0.0000  0.3023
          system  0.0000  0.0000  0.2167  0.3023  0.0000  0.0000  0.0000
         testing  0.0000  0.0000  0.0000  0.1977  0.0000  0.0000  0.0000
            time  0.0000  0.1735  0.0000  0.0000  0.0000  0.0000  0.0000
            user  0.0000  0.1735  0.0000  0.0000  0.0000  0.0000  0.0000
 user interface  0.0000  0.0000  0.2833  0.0000  0.0000  0.0000  0.0000
```

Feature Selection adalah Teknik pemilihan sebuah subset feature yang relevan untuk membentuk model yang baik.

http://222.124.22.22/budsus/pdf/textwebmining/TextMining_Kuliah.pdf

http://eprints.umg.ac.id/603/3/BAB%20II.pdf

Text Mining

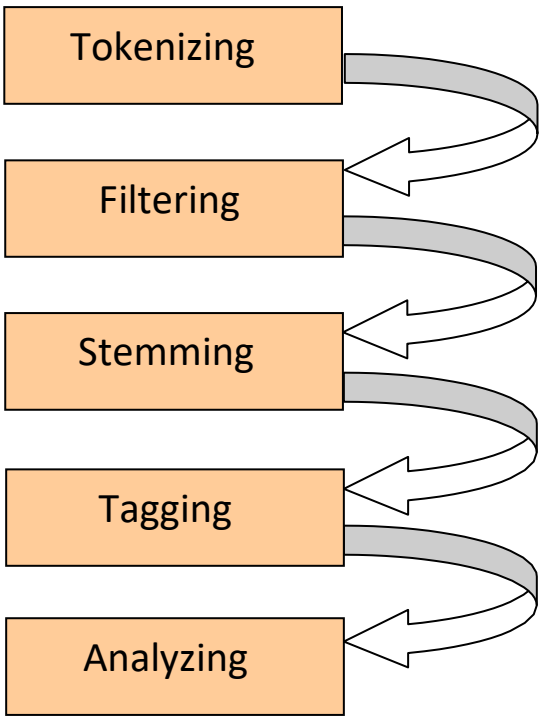
Text mining memiliki definisi menambang data yang berupa teks dimana sumber data biasanya didapatkan dari dokumen, dan tujuannya adalah mencari kata-kata yang dapat mewakili isi dari dokumen sehingga dapat dilakukan analisa keterhubungan antar dokumen

Berdasarkan ketidakteraturan struktur data teks, maka proses text mining memerlukan beberapa tahap awal yang pada intinya adalah mempersiapkan agar teks dapat diubah menjadi lebih terstruktur.

Tujuan dari text mining adalah untuk mendapatkan informasi yang berguna dari sekumpulan dokumen. Jadi, sumber data yang digunakan pada text mining adalah kumpulan teks yang memiliki format yang tidak terstruktur atau minimal semi terstruktur. Adapun tugas khusus dari text mining antara lain yaitu pengkategorisasian teks (text categorization) dan pengelompokan teks text clustering).

Tahapan dalam Text Mining

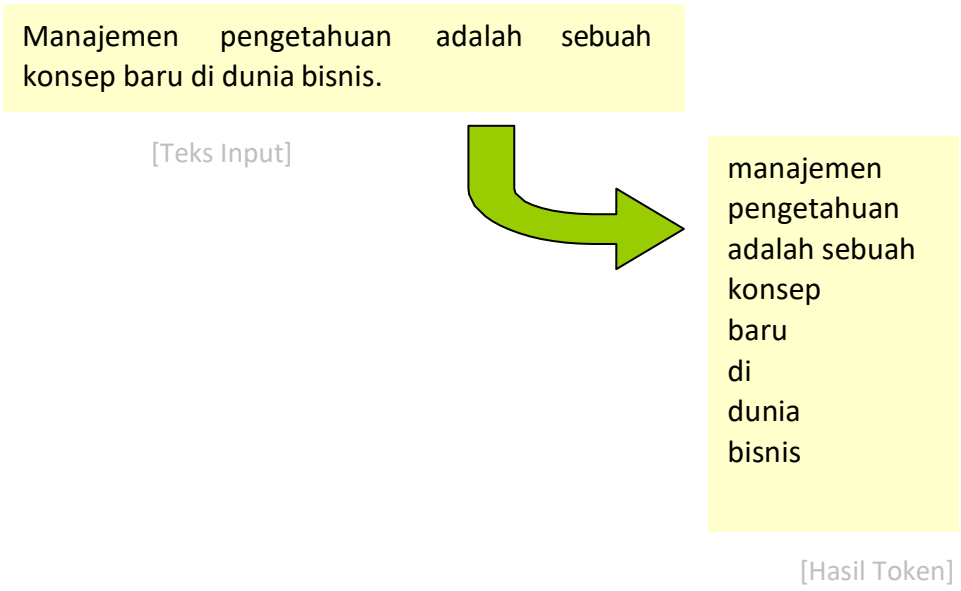
Tahapan yang dilakukan secara umum adalah:



1.Tokenizing

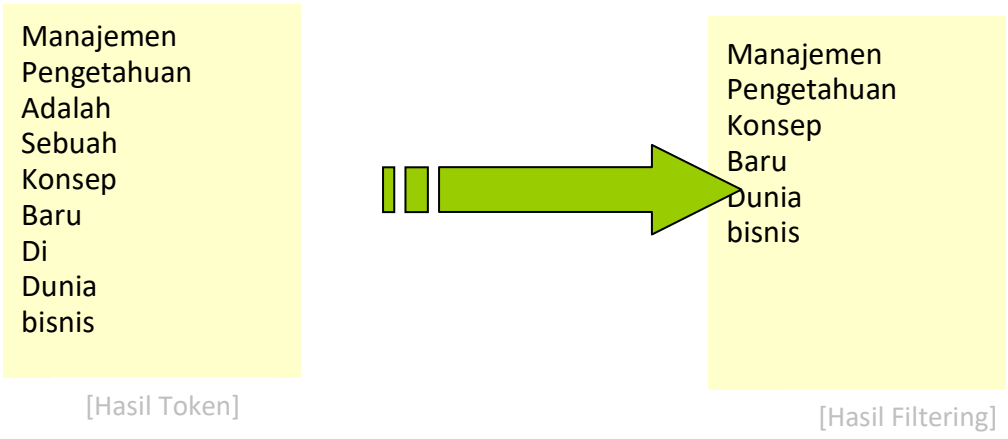
Tahap tokenizing adalah tahap pemotongan string input berdasarkantiap kata yang menyusunnya.

Contoh dari tahap ini adalahsebagai berikut



2. Filtering

- Tahap filtering adalah tahap mengambil kata-kata penting dari hasil token.
- Bisa menggunakan algoritma stop list (membuang kata yang kurang penting) atau word list (menyimpan kata penting).
- Contoh dari tahap ini adalah sebagai berikut:



3. Stemming

- Tahap stemming adalah tahap mencari root kata dari tiap kata hasil filtering.
- Contoh dari tahap ini adalah sebagai berikut:

learning
using
text
mining

learn
use
text
mine

[Hasil Filter]

[Hasil Stemming]

4. Tagging

- Tahap tagging adalah tahap mencari bentuk awal/root dari tiap kata lampau atau kata hasil stemming.
- Contoh dari tahap ini adalah sebagai berikut:

was
used
stori

be
use
story

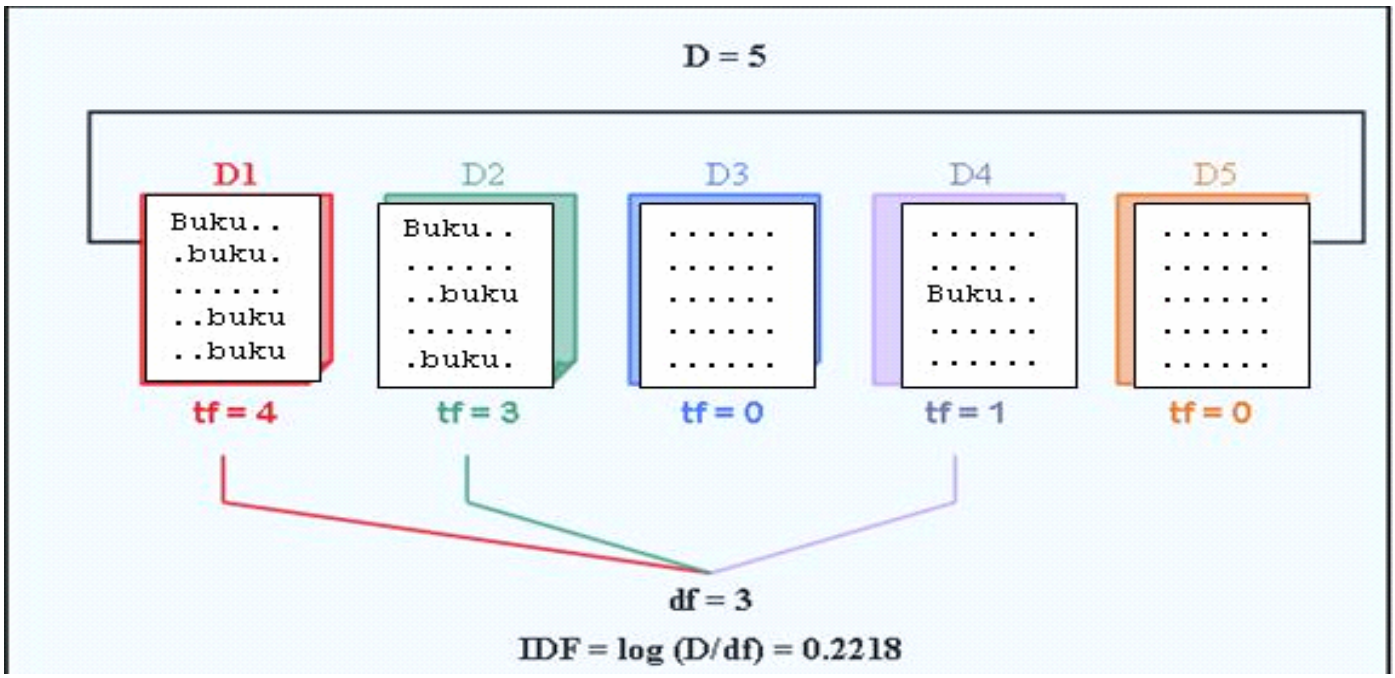
[Hasil Stemming]

[Hasil tangging]

5. Analyzing

- Tahap Analyzing merupakan tahap penentuan seberapa jauh keterhubungan antar kata-kata antar dokumen yangada.

Ilustrasi Algoritma Text Mining



D1, D2, D3, D4, D5 = dokumen

Tf = banyak kata yang dicari pada sebuah dokumen

D = total dokumen

Df = banyak dokumen yang mengandung kata yang dicari

Algoritma TF/IDF

Formula yang digunakan untuk menghitung bobot (w) masing-masing dokumen terhadap kata kunci adalah

$$W_{d,t} = tf_{d,t} * IDF_t$$

Dimana:

d = dokumen ke-d

t = kata ke-t dari kata kunci

W = bobot dokumen ke-d terhadap kata ke-t

Setelah bobot (w) masing-masing dokumen diketahui, maka dilakukan proses sorting/pengurutan dimana semakin besar nilai w, semakin besar tingkat similaritas dokumen tersebut terhadap kata yang dicari, demikian sebaliknya.

Ilustrasi TF/IDF

Kata kunci (kk) = pengetahuan logistik Dokumen 1
(D1) = Manajemen transaksi logistik Dokumen 2
(D2) = Pengetahuan antar individu
Dokumen 3 (D3) = Dalam manajemen pengetahuan terdapat transfer pengetahuan logistik
Jadi jumlah dokumen (D) = 3

Setelah melalui proses filtering, maka kata antar pada dokumen 2 serta kata dalam dan terdapat pada dokumen 3 dihapus

Tabel perhitungan TF-IDF

	tf				df	D/df	IDF	W			
token	kk	D1	D2	D3			og(D/ df)	kk	D1	D2	D3
manajemen	0	1	0	1	2	1.5	0.176	0	0.176	0	0.176
transaksi	0	1	0	0	1	3	0.477	0	0.477	0	0
logistik	1	1	0	1	2	1.5	0.176	0.176	0.176	0	0.176
transfer	0	0	0	1	1	3	0.477	0	0	0	0.477
pengetahuan	1	0	1	2	2	1.5	0.176	0.176	0	0.176	0.352
individu	0	0	1	0	1	3	0.477	0	0	0.477	0
Total		0.352	0.829	0.653	1.181						

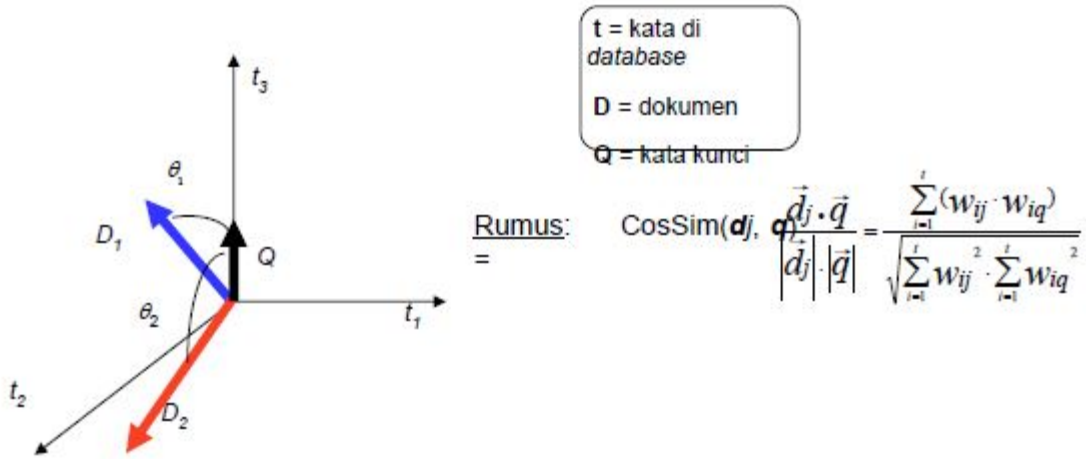
bobot (w) untuk D1 = 0.176 + 0 = 0.176
bobot (w) untuk D2 = 0 + 0.176 = 0.176
bobot (w) untuk D3 = 0.176 + 0.352 = 0.528

Analisa TFIIDF

- Dari contoh studi kasus di atas, dapat diketahui bahwa nilai bobot (w) dari D1 dan D2 adalah sama.
- Apabila diurutkan maka proses sorting juga tidak akan dapat mengurutkan secara tepat, karena nilai w keduanya sama.
- Untuk mengatasi hal ini, digunakan algoritma dari vector-space model

Vector Space Model

- Ide dari metode ini adalah dengan menghitung nilai cosinus sudut dari dua vektor, yaitu W dari tiap dokumen dan W dari kata kunci



Tabel perhitungan Vector-Space Model

token	Kk	D1	D2	D3	Kk*D1	Kk*D2	kk*D3
manajemen	0	0.031	0	0.031	0	0	0
Transaksi	0	0.228	0	0	0	0	0
Logistik	0.031	0.031	0	0.031	0.031	0	0.031
Transfer	0	0	0	0.228	0	0	0
pengetahuan	0.031	0	0.031	0.124	0	0.031	0.062
Individu	0	0	0.228	0	0	0	0
	Sqrt(kk)	**Sqrt(Di)**			**Sum(kk dot Di)**		
	0.249	**0.539**	**0.509**	**0.643**	**0.031**	**0.031**	

Perhitungan

- **Sqrt(kk)** = Sqrt($\sum_{j=1}^{n} kk_j^2$)
  dimana j = kata di database
- Misalnya untuk Sqrt(kk) = Sqrt($\sum_{j=1}^{n} kk_j^2$)
  = $\sqrt{(0+0+0.031+0+0.031+0)}$
  = $\sqrt{0.062}$ = 0.249
- **Sqrt(D_i)** = Sqrt($\sum_{j=1}^{n} D_{i,j}^2$)
  dimana j = kata di database
- Misalnya untuk Sqrt(D_2) = Sqrt($\sum_{j=1}^{n} D_{2,j}^2$)
  = $\sqrt{(0+0+0+0+0.031+0.228)}$
  = $\sqrt{0.259}$ = 0.509

- **Sum(kk dot D_i)** = $\sum_{j=1}^{n} kk_j D_{i,j}$

  dimana j = kata di database
- Misalnya untuk Sum(kk dot D_3) = $\sum_{j=1}^{n} kk_j D_{3,j}$

  = 0 + 0 + 0.031 + 0 + 0.062 + 0
  = 0.093

Selanjutnya menghitung nilai Cosinus sudut antara vektor kata kunci dengan tiap dokumen dengan rumus:

Cosine (D_i) = sum(kk dot D_i) / [sqrt(kk) * sqrt(D_i))]

- Misalnya untuk D_3 maka:
- Cosine (D_3)= sum(kk dot D_3) / [sqrt(kk) * sqrt(D_3))]
  = 0.093 / [0.249 * 0.643]
  = 0.581

Analisa Vector Space Model

- Demikian juga untuk Cosine dari D_1 dan D_2. Sehingga hasil yang diperoleh untuk ketiga dokumen di atas adalah seperti berikut ini.
- Tabel 2.3 Tabel hasil Vector-Space Model

	D1	D2	D3
Cosine	0.231	0.245	0.581
	Rank 3	Rank 2	Rank 1

- Dari hasil akhir (Cosine) maka dapat diketahui bahwa document 3 (D3) memiliki tingkat similaritas tertinggi kemudian disusul dengan D2 lalu D1.

Tugas 8.

Nama : Juminovario
NIM : 202420018
Kelas : MTI 23 Reg-A
MK : Advanced Database

Salah satu keunggulan python adalah mendukung banyak *open-source library*. Ada banyak *library* python yang dapat digunakan untuk melakukan dan mengimplementasikan masalah dalam NLP.

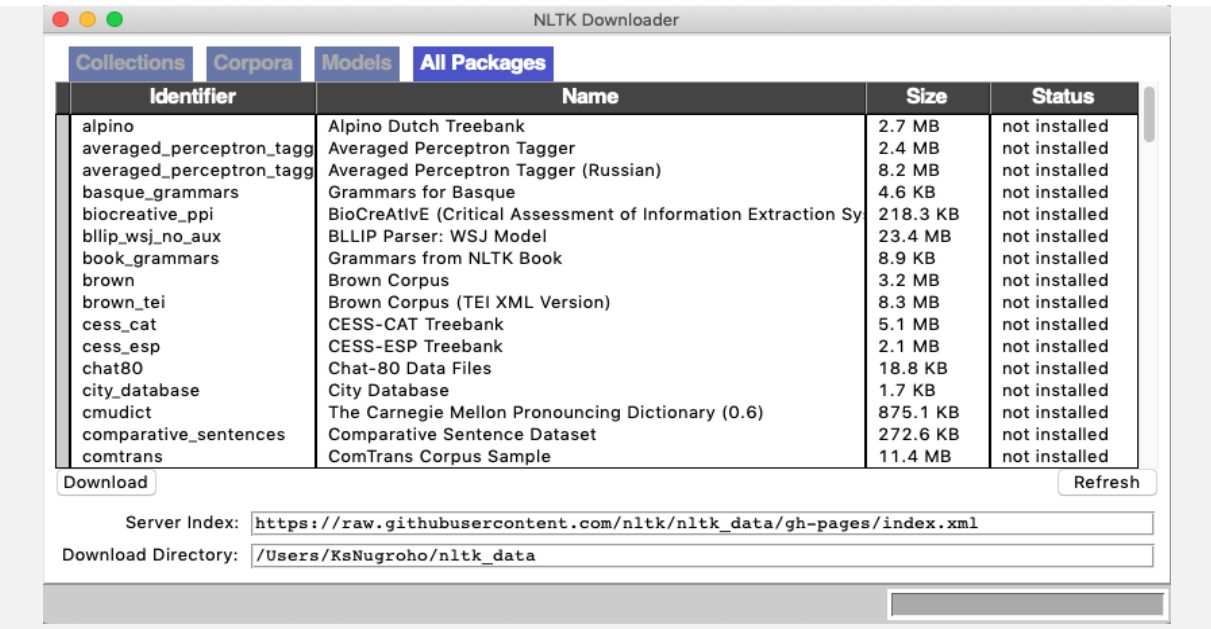
Natural Language Toolkit (NLTK)

Natural Language Toolkit atau disingkat NLTK, adalah *libray* python untuk bekerja dengan permodelan teks. NLTK menyediakan alat yang baik mempersiapkan teks sebelum digunakan pada *machine learning* atau algoritma *deep learning*. Cara termudah untuk menginstall NLTK adalah menggunakan "pip" pada *command line/terminal*.

```
pip install nltk
```

Langkah pertama yang perlu anda lakukan setelah menginstall NLTK adalah mengunduh paket NLTK.

```
Import nltk
nltk.download()
```



NLTK

Python Sastrawi

Python Sastrawi adalah pengembangan dari proyek PHP Sastrawi. Python Sastrawi merupakan library sederhana yang dapat mengubah kata berimbuhan bahasa Indonesia menjadi bentuk dasarnya. Sastrawi juga dapat diinstal melalui "pip".

```
pip install Sastrawi
```

Penggunaan Python Sastrawi sangat sederhana seperti baris kode dibawah ini :

```
# import StemmerFactory class
from Sastrawi.Stemmer.StemmerFactory import
StemmerFactory

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

# stemming process
sentence = 'Perekonomian Indonesia sedang dalam pertumbuhan yang membanggakan'

output   = stemmer.stem(sentence)

print(output)
# ekonomi indonesia sedang dalam tumbuh yang bangga

print(stemmer.stem('Mereka meniru-nirukannya'))
# mereka tiru
```

1. Case folding

Case folding adalah salah satu bentuk *text preprocessing* yang paling sederhana dan efektif meskipun sering diabaikan. Tujuan dari *case folding* untuk mengubah semua huruf dalam dokumen menjadi huruf kecil. Hanya huruf 'a' sampai 'z' yang diterima. Karakter selain huruf dihilangkan dan dianggap *delimiter*. Pada tahap ini tidak menggunakan *external library* apapun, kita bisa memanfaatkan modul yang tersedia di python.

Ada beberapa cara yang dapat digunakan dalam tahap *case folding*, anda dapat menggunakan beberapa atau menggunakan semuanya, tergantung pada tugas yang diberikan.

Mengubah text menjadi lowercase

Salah satu contoh pentingnya penggunaan *lower case* adalah untuk mesin pencarian. Bayangkan anda sedang mencari dokumen yang mengandung "indonesia" namun tidak ada hasil yang muncul karena "indonesia" di indeks sebagai "INDONESIA". Siapa yang harus kita salahkan? U.I. *designer* yang mengatur *user interface* atau *developer* yang mengatur sistem dalam indeks pencarian?

Contoh dibawah menunjukan bagaimana python mengubah teks menjadi *lowercase* :

```
kalimat = "Berikut ini adalah 5 negara dengan pendidikan terbaik di dunia adalah Korea Selatan, Jepang, Singapura, Hong Kong, dan Finlandia."
lower_case = kalimat.lower()
print(lower_case)
# output
# berikut ini adalah 5 negara dengan pendidikan terbaik di dunia adalah korea selatan, jepang, singapura, hong kong, dan finlandia.
```

Menghapus angka

Hapuslah angka jika tidak relevan dengan apa yang akan anda analisa, contohnya seperti nomor rumah, nomor telepon, dll. *Regular expression* (*regex*) dapat digunakan untuk menghapus karakter angka. Python memiliki modulre untuk melakukan hal – hal yang berkaitan dengan *regex*.

Contoh dibawah menunjukan bagaimana python menghapus angka dalam sebuah kalimat :

```
import re # impor modul regular expression
kalimat = "Berikut ini adalah 5 negara dengan pendidikan terbaik di dunia adalah Korea Selatan, Jepang, Singapura, Hong Kong, dan Finlandia."
hasil = re.sub(r"\d+", "", kalimat)
print(hasil)

# ouput
# Berikut ini adalah negara dengan pendidikan terbaik di dunia adalah Korea Selatan, Jepang, Singapura, Hong Kong, dan Finlandia.
```

Menghapus tanda baca

Sama halnya dengan angka, tanda baca dalam kalimat tidak memiliki pengaruh pada *text preprocessing*. Menghapus tanda baca seperti [!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~] dapat dilakukan di pyhton seperti dibawah ini :

```
kalimat = "Ini &adalah [contoh] kalimat? {dengan} tanda. baca?!!"
hasil =
 kalimat.translate(str.maketrans("","",string.punctuation))
print(hasil)

# output
# Ini adalah contoh kalimat dengan tanda baca
```

Menghapus whitepace (karakter kosong)

Untuk menghapus spasi di awal dan akhir, anda dapat menggunakan fungsi strip() pada pyhton. Perhatikan kode dibawah ini :

```
kalimat = " \t ini kalimat contoh\t "
hasil = kalimat.strip()
print(hasil)

# output
# ini kalimat contoh
```

2. Tokenizing

Tokenizing adalah proses pemisahan teks menjadi potongan-potongan yang disebut sebagai token untuk kemudian di analisa. Kata, angka, simbol, tanda baca dan entitas penting lainnya dapat dianggap sebagai token. Didalam NLP, token diartikan sebagai "kata" meskipun *tokenize* juga dapat dilakukan pada paragraf maupun kalimat.

Fungsi split() pada pyhton dapat digunakan untuk memisahkan teks. Perhatikan contoh dibawah ini :

```
kalimat = "rumah idaman adalah rumah yang bersih"
pisah = kalimat.split()
print(pisah)

# output
# ['rumah', 'idaman', 'adalah', 'rumah', 'yang', 'bersih']
```

Sayangnya, tahap *tokenizing* tidak sesederhana itu. Dari *output* kode diatas kita akan mengolah kata "rumah" dan "rumah" sebagai 2 entitas yang berbeda. Permasalahan ini tentunya akan mempengaruhi hasil dari analisa teks itu sendiri.

Untuk mengatasi masalah ini kita dapat menggunakan modul dari NLTK yang sudah diinstal sebelumnya.

Tokenizing kata

Sebuah kalimat atau data dapat dipisah menjadi kata-kata dengan kelas word_tokenize() pada modul NLTK.

```
# impor word_tokenize dari modul nltk
from nltk.tokenize import word_tokenize

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online."

tokens = nltk.tokenize.word_tokenize(kalimat)
print(tokens)

# ouput
# ['Andi', 'kerap', 'melakukan', 'transaksi', 'rutin', 'secara', 'daring', 'atau', 'online', '.']
```

Dari *output* kode diatas terdapat kemunculan tanda baca titik(.) dan koma (,) serta token "Andi" yang masih menggunakan huruf besar pada awal kata. Hal tersebut nantinya dapat menggangu proses perhitungan dalam penerapan algoritma. Jadi, sebaiknya teks telah melewati tahap *case folding* sebelum di *tokenize* agar menghasilkan hasil yang lebih konsisten.

Kita dapat menambahkan fungsi *case folding* untuk menghilangkan tanda baca dan mengubah *teks* ke dalam bentuk *lowercase*.

```
kalimat = kalimat.translate(str.maketrans("",string.punctuation)).lower()

# output
# ['andi', 'kerap', 'melakukan', 'transaksi', 'rutin', 'secara', 'daring', 'atau', 'online']
```

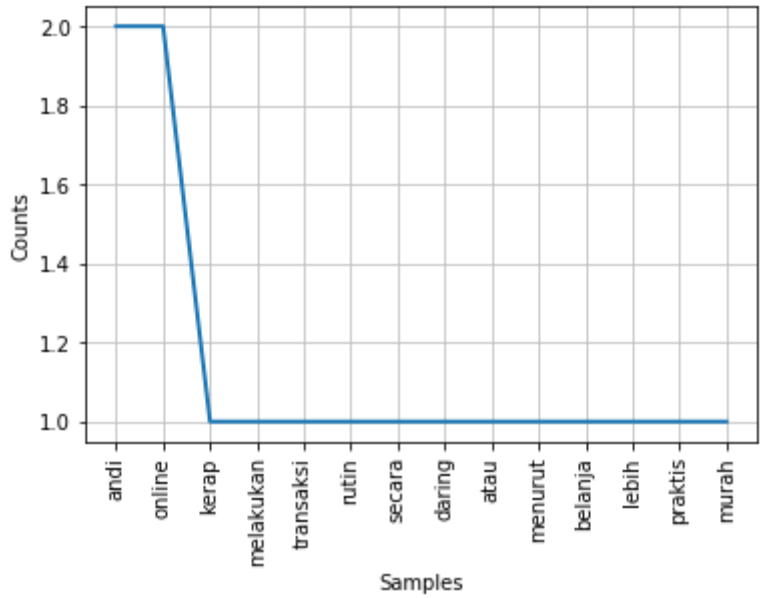
Kita juga bisa mendapatkan informasi frekuensi kemunculan setiap token dengan kelas FreqDist() yang sudah tersedia pada modul NLTK.

```
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDist
kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online. Menurut Andi belanja online lebih praktis & murah."
kalimat = kalimat.translate(str.maketrans("",string.punctuation)).lower()
tokens = nltk.tokenize.word_tokenize(kalimat)
kemunculan = nltk.FreqDist(tokens)
print(kemunculan.most_common())
```

```
# output
# [('andi', 2), ('online', 2), ('kerap', 1), ('melakukan', 1), ('transaksi', 1), ('rutin', 1), ('secara', 1), ('daring', 1), ('atau', 1), ('menurut', 1), ('belanja', 1), ('lebih', 1), ('praktis', 1), ('murah', 1)]
```

Untuk menggambarkan ke dalam bentuk grafik, kita perlu menginstall library Matplotlib.

```
import matplotlib.pyplot as plt

kemunculan.plot(30,cumulative=False)
plt.show()
```



Grafik frekuensi kemunculan kata pada dokumen

Tokenizing kalimat

Prinsip yang sama dapat diterapkan untuk memisahkan kalimat pada paragraf. Anda dapat menggunkan kelas sent_tokenize() pada modul NLTK. Saya telah menambahkan kalimat pada contoh seperti dibawah ini :

```
# impor sent_tokenize dari modul nltk
from nltk.tokenize import sent_tokenize

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online. Menurut Andi belanja online lebih praktis & murah."

tokens = nltk.tokenize.sent_tokenize(kalimat)
print(tokens)
```

```
# ouput
# ['Andi kerap melakukan transaksi rutin secara daring atau online.', 'Menurut Andi belanja online lebih praktis & murah.']
```

3. Filtering (Stopword Removal)

Filtering adalah tahap mengambil kata-kata penting dari hasil token dengan menggunakan algoritma *stoplist* (membuang kata kurang penting) atau *wordlist* (menyimpan kata penting).

Stopword adalah kata umum yang biasanya muncul dalam jumlah besar dan dianggap tidak memiliki makna. Contoh *stopword* dalam bahasa Indonesia adalah “yang”, “dan”, “di”, “dari”, dll. Makna di balik penggunaan *stopword* yaitu dengan menghapus kata-kata yang memiliki informasi rendah dari sebuah teks, kita dapat fokus pada kata-kata penting sebagai gantinya.

Contoh penggunaan *filtering* dapat kita temukan pada konteks mesin pencarian. Jika permintaan pencarian anda adalah “apa itu pengertian manajemen?” tentunya anda ingin sistem pencarian fokus pada memunculkan dokumen dengan topik tentang “pengertian manajemen” di atas dokumen dengan topik “apa itu”. Hal ini dapat dilakukan dengan mencegah kata dari daftar *stopword* dianalisa.

Filtering dengan NLTK

```
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online. Menurut Andi belanja online lebih praktis & murah."
kalimat = kalimat.translate(str.maketrans("",string.punctuation)).lower()

tokens = word_tokenize(kalimat)
listStopword =  set(stopwords.words('indonesian'))

removed = []
for t in tokens:
    if t not in listStopword:
        removed.append(t)

print(removed)

# ouput
# ['andi', 'kerap', 'transaksi', 'rutin', 'daring', 'online', 'andi', 'belanja', 'online', 'praktis', 'murah']
```

Filtering dengan sastrawi

Selain untuk *stemming,* library Sastrawi juga mendukung proses *filtering*. Kita dapat menggunakan **stopWordRemoverFactory** dari modul sastrawi.

Untuk melihat daftar *stopword* yang telah didefinisikan dalam library Sastrawi dapat menggunakan kode berikut :

```
from Sastrawi.StopWordRemover.StopWordRemoverFactory
import StopWordRemoverFactory

factory = StopWordRemoverFactory()
stopwords = factory.get_stop_words()
print(stopwords)
```

Kode diatas akan menampikan *stopword* yang tersedia di library Sastrawi. Prosos *filtering* pada Sastrawi dapat dilihat pada baris kode dibawah :

```
from Sastrawi.StopWordRemover.StopWordRemoverFactory
import StopWordRemoverFactory
from nltk.tokenize import word_tokenize

factory = StopWordRemoverFactory()
stopword = factory.create_stop_word_remover()

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online. Menurut Andi belanja online lebih praktis & murah."
kalimat = kalimat.translate(str.maketrans("",string.punctuation)).lower()

stop = stopword.remove(kalimat)
tokens = nltk.tokenize.word_tokenize(stop)
print(tokens)

# output
# ['andi', 'kerap', 'transaksi', 'rutin', 'daring', 'online', 'andi', 'belanja', 'online', 'praktis', 'murah']
```

Kita dapat menambah atau mengurangi kata pada daftar *stopword* sesuai dengan kebutuhan analisa. Pada dasarnya daftar *stopword* pada library Sastrawi tersimpan di dalam list yang anda lihat disini. Jadi sebenarnya kita tinggal mengubah daftar pada list tersebut. Tetapi hal tersebut bisa menjadi permasalahan apabila pada suatu kasus kita diharuskan menambahkan *stopword* secara dinamis.

Library Sastrawi dapat mengatasi permasalahan tersebut, perhatikan kode dibawah ini :

```
from Sastrawi.StopWordRemover.StopWordRemoverFactory
import StopWordRemoverFactory, StopWordRemover, ArrayDictionary
from nltk.tokenize import word_tokenize

stop_factory = StopWordRemoverFactory().get_stop_words() #load defaul stopword
more_stopword = ['daring', 'online'] #menambahkan stopword

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online. Menurut Andi belanja online lebih praktis & murah."
kalimat = kalimat.translate(str.maketrans(",",string.punctuation)).lower()

data = stop_factory + more_stopword #menggabungkan stopword

dictionary = ArrayDictionary(data)
str = StopWordRemover(dictionary)
tokens = nltk.tokenize.word_tokenize(str.remove(kalimat))

print(tokens)

# output
# ['andi', 'kerap', 'transaksi', 'rutin', 'andi', 'belanja', 'praktis', 'murah']
```

Menurut Jim Geovedi pada artikelnya menjelaskan bahwa penyesuaian daftar *stopwords* perlu dilakukan setiap pertama kali melakukan proyek analisa. Memang bukan sesuatu yang melelahkan, tapi jika tidak dilakukan maka ini akan dapat mengakibatkan salah interpretasi terhadap data.

4. Stemming

Stemming adalah proses menghilangkan infleksi kata ke bentuk dasarnya, namun bentuk dasar tersebut tidak berarti sama dengan akar kata (*root word*). Misalnya kata “mendengarkan”, “dengarkan”, “didengarkan” akan ditransformasi menjadi kata “dengar”.

Idenya adalah ketika anda mencari dokumen “cara membuka lemari”, anda juga ingin melihat dokumen yang menyebutkan “cara terbuka lemari” atau “cara dibuka lemari” meskipun terdengar tidak enak. Tentunya anda ingin mencocokan semua variasi kata untuk memunculkan dokumen yang paling relevan.

Stemming dengan NLTK (bahasa inggris)

Ada banyak algoritma yang digunakan untuk *stemming*. Salah satu algoritma yang tertua dan paling populer adalah algoritma Porter. Algoritma ini tersedia dalam modul NLTK melalui kelas PorterStemmer().

```
from nltk.stem import PorterStemmer
ps = PorterStemmer()
kata = ["program", "programs", "programer", "programing", "programers"]

for k in kata:
    print (k, " : ", ps.stem(k))

# ouput
# program  :  program
programs  :  program
programer  :  program
programing  :  program
programers  :  program
```

Selain Porter, NLTK juga mendukung algoritma Lancester, WordNet Lemmatizer, dan SnowBall. Sayangnya proses *stemming* Bahasa Indonesia pada modul NLTK belum didukung.

Stemming bahasa indonesia menggunakan Python Sastrawi

Proses *stemming* antara satu bahasa dengan bahasa yang lain tentu berbeda. Contohnya pada teks berbahasa inggris, proses yang diperlukan hanya proses menghilangkan sufiks. Sedangkan pada teks berbahasa Indonesia semua kata imbuhan baik itu sufiks dan prefiks juga dihilangkan.

Untuk melakukan *stemming* bahasa Indonesia kita dapat menggunakan library Python Sastrawi yang sudah kita siapkan di awal. *Library* Sastrawi menerapkan Algoritma Nazief dan Adriani dalam melakukan *stemming* bahasa Indonesia.

```
from Sastrawi.Stemmer.StemmerFactory
import StemmerFactory

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

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online. Menurut Andi belanja online lebih praktis & murah."

hasil = stemmer.stem(**kalimat**)

print(**hasil**)

ouput
andi kerap laku transaksi rutin cara daring atau online turut andi belanja online lebih praktis murah

Nama : Nanda Tri Haryati
NIM / Kelas : 202420016 / MTI23- REG-A

Tugas 08

Tutorial Text Mining

Pada *natural language processing* (NLP), informasi yang akan digali berisi data-data yang strukturnya "sembarang" atau tidak terstruktur. Oleh karena itu, diperlukan proses pengubahan bentuk menjadi data yang terstruktur untuk kebutuhan lebih lanjut (*sentiment analysis*, *topic modelling*, dll).

Persiapan : Library yang dibutuhkan

Salah satu keunggulan python adalah mendukung banyak *open-source library*. Ada banyak *library* python yang dapat digunakan untuk melakukan dan mengimplementasikan masalah dalam NLP.

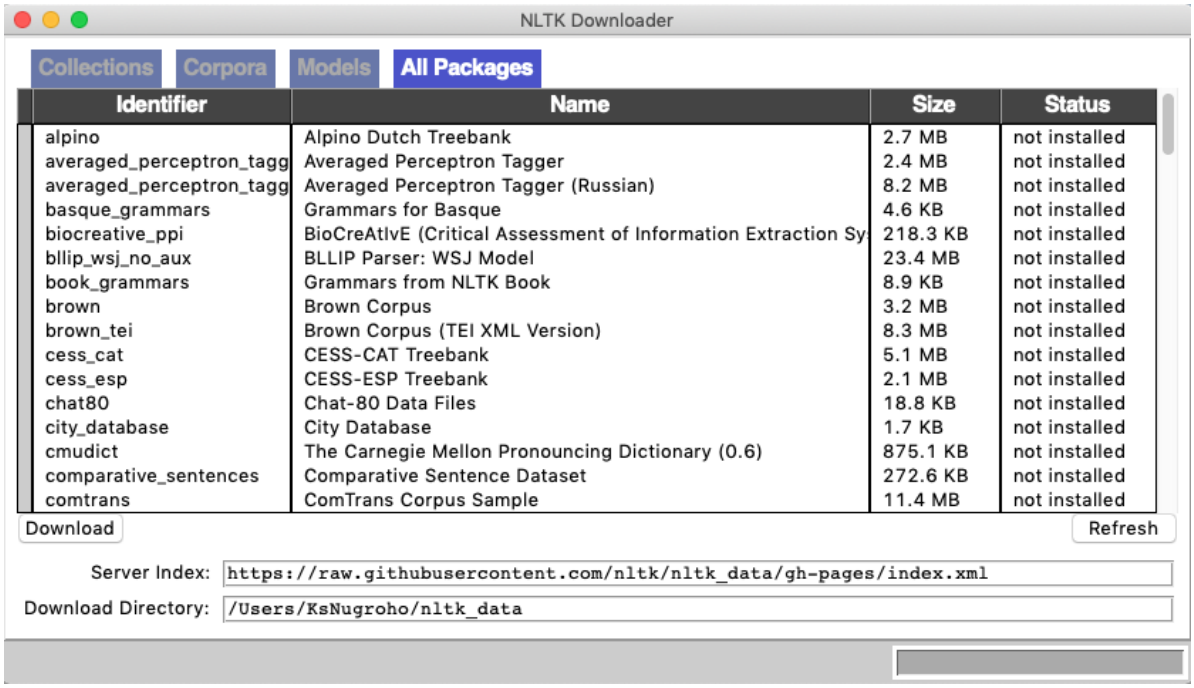
Natural Language Toolkit (NLTK)

Natural Language Toolkit atau disingkat NLTK, adalah *libray* python untuk bekerja dengan permodelan teks. NLTK menyediakan alat yang baik mempersiapkan teks sebelum digunakan pada *machine learning* atau algoritma *deep learnin*g. Cara termudah untuk menginstall NLTK adalah menggunakan "pip" pada *command line*/*terminal*.

```
pip install nltk
```

Langkah pertama yang perlu anda lakukan setelah menginstall NLTK adalah mengunduh paket NLTK.

```
import nltk
nltk.download()
```



NLTK

Nama : Nanda Tri Haryati
NIM / Kelas : 202420016 / MTI23- REG-A

Python Sastrawi (Stemming Bahasa Indonesia)

Python Sastrawi adalah pengembangan dari proyek PHP Sastrawi. Python Sastrawi merupakan library sederhana yang dapat mengubah kata berimbuhan bahasa Indonesia menjadi bentuk dasarnya. Sastrawi juga dapat diinstal melalui "pip".

```
pip install Sastrawi
```

Penggunaan Python Sastrawi sangat sederhana seperti baris kode dibawah ini :

```
# import StemmerFactory class
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory# create stemmer
factory = StemmerFactory()
stemmer = factory.create_stemmer()# stemming process
sentence = 'Perekonomian Indonesia sedang dalam pertumbuhan yang
membanggakan'output   = stemmer.stem(sentence)print(output)
# ekonomi indonesia sedang dalam tumbuh yang
banggaprint(stemmer.stem('Mereka meniru-nirukannya'))
# mereka tiru
```

1. Case folding

Case folding adalah salah satu bentuk *text preprocessing* yang paling sederhana dan efektif meskipun sering diabaikan. Tujuan dari *case folding* untuk mengubah semua huruf dalam dokumen menjadi huruf kecil. Hanya huruf 'a' sampai 'z' yang diterima. Karakter selain huruf dihilangkan dan dianggap *delimiter*. Pada tahap ini tidak menggunakan *external library* apapun, kita bisa memanfaatkan modul yang tersedia di python.

Ada beberapa cara yang dapat digunakan dalam tahap *case folding*, anda dapat menggunakan beberapa atau menggunakan semuanya, tergantung pada tugas yang diberikan.

Mengubah text menjadi lowercase

Salah satu contoh pentingnya penggunaan *lower case* adalah untuk mesin pencarian. Bayangkan anda sedang mencari dokumen yang mengandung "indonesia" namun tidak ada hasil yang muncul karena "indonesia" di indeks sebagai "INDONESIA". Siapa yang harus kita salahkan? U.I. *designer* yang mengatur *user interface* atau *developer* yang mengatur sistem dalam indeks pencarian?

Contoh dibawah menunjukan bagaimana python mengubah teks menjadi *lowercase* :

```
kalimat = "Berikut ini adalah 5 negara dengan pendidikan terbaik di dunia
adalah Korea Selatan, Jepang, Singapura, Hong Kong, dan
Finlandia."lower_case = kalimat.lower()
print(lower_case)# output
# berikut ini adalah 5 negara dengan pendidikan terbaik di dunia adalah
korea selatan, jepang, singapura, hong kong, dan finlandia.
```

Menghapus angka

Hapuslah angka jika tidak relevan dengan apa yang akan anda analisa, contohnya seperti nomor rumah, nomor telepon, dll. *Regular expression* (*regex*) dapat digunakan untuk menghapus

karakter angka. Python memiliki modul`re` untuk melakukan hal – hal yang berkaitan dengan *regex*.

Contoh dibawah menunjukan bagaimana python menghapus angka dalam sebuah kalimat :

```
import re # impor modul regular expressionkalimat = "Berikut ini adalah 5
negara dengan pendidikan terbaik di dunia adalah Korea Selatan, Jepang,
Singapura, Hong Kong, dan Finlandia."
hasil = re.sub(r"\d+", "", kalimat)
print(hasil)# ouput
# Berikut ini adalah negara dengan pendidikan terbaik di dunia adalah Korea
Selatan, Jepang, Singapura, Hong Kong, dan Finlandia.
```

Menghapus tanda baca

Sama halnya dengan angka, tanda baca dalam kalimat tidak memiliki pengaruh pada *text preprocessing*. Menghapus tanda baca seperti [!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~] dapat dilakukan di pyhton seperti dibawah ini :

```
kalimat = "Ini &adalah [contoh] kalimat? {dengan} tanda. baca?!!"
hasil = kalimat.translate(str.maketrans("","",string.punctuation))
print(hasil)# output
# Ini adalah contoh kalimat dengan tanda baca
```

Menghapus whitepace (karakter kosong)

Untuk menghapus spasi di awal dan akhir, anda dapat menggunakan fungsi `strip()` pada pyhton. Perhatikan kode dibawah ini :

```
kalimat = " \t ini kalimat contoh\t "
hasil = kalimat.strip()
print(hasil)# output
# ini kalimat contoh
```

2. Tokenizing

Tokenizing adalah proses pemisahan teks menjadi potongan-potongan yang disebut sebagai token untuk kemudian di analisa. Kata, angka, simbol, tanda baca dan entitas penting lainnya dapat dianggap sebagai token. Didalam NLP, token diartikan sebagai "kata" meskipun *tokenize* juga dapat dilakukan pada paragraf maupun kalimat.

Fungsi `split()` pada pyhton dapat digunakan untuk memisahkan teks. Perhatikan contoh dibawah ini :

```
kalimat = "rumah idaman adalah rumah yang bersih"
pisah = kalimat.split()
print(pisah)# output
# ['rumah', 'idaman', 'adalah', 'rumah', 'yang', 'bersih']
```

Sayangnya, tahap *tokenizing* tidak sesederhana itu. Dari *output* kode diatas kita akan mengolah kata "rumah" dan "rumah" sebagai 2 entitas yang berbeda. Permasalahan ini tentunya akan mempengaruhi hasil dari analisa teks itu sendiri.

Nama : Nanda Tri Haryati
NIM / Kelas : 202420016 / MTI23- REG-A

Untuk mengatasi masalah ini kita dapat menggunakan modul dari NLTK yang sudah diinstal sebelumnya.

Tokenizing kata

Sebuah kalimat atau data dapat dipisah menjadi kata-kata dengan kelas `word_tokenize()` pada modul NLTK.

```
# impor word_tokenize dari modul nltk
from nltk.tokenize import word_tokenize

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online."

tokens = nltk.tokenize.word_tokenize(kalimat)
print(tokens)# ouput
# ['Andi', 'kerap', 'melakukan', 'transaksi', 'rutin', 'secara', 'daring',
'atau', 'online', '.']
```

Dari *output* kode diatas terdapat kemunculan tanda baca titik(.) dan koma (,) serta token "Andi" yang masih menggunakan huruf besar pada awal kata. Hal tersebut nantinya dapat menggangu proses perhitungan dalam penerapan algoritma. Jadi, sebaiknya teks telah melewati tahap *case folding* sebelum di *tokenize* agar menghasilkan hasil yang lebih konsisten.

Kita dapat menambahkan fungsi *case folding* untuk menghilangkan tanda baca dan mengubah *teks* ke dalam bentuk *lowercase*.

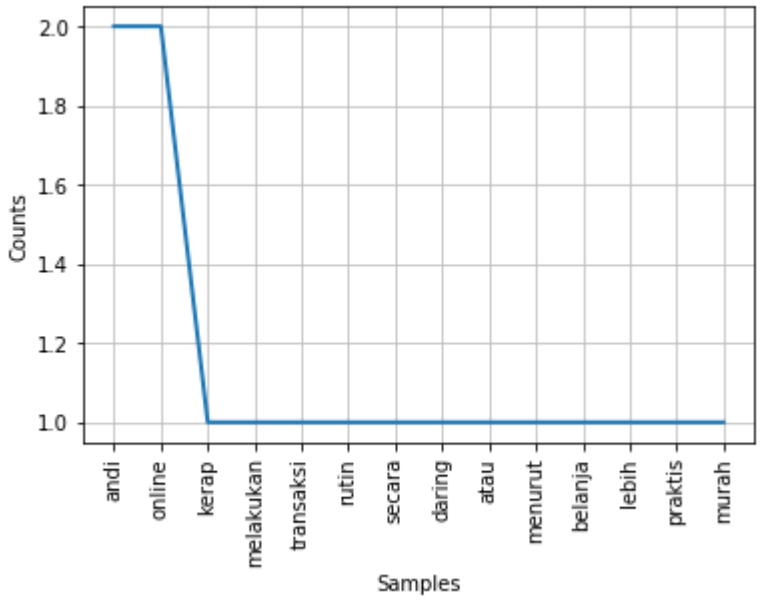
```
kalimat =
kalimat.translate(str.maketrans('','',string.punctuation)).lower()# output
# ['andi', 'kerap', 'melakukan', 'transaksi', 'rutin', 'secara', 'daring',
'atau', 'online']
```

Kita juga bisa mendapatkan informasi frekuensi kemunculan setiap token dengan kelas `FreqDist()` yang sudah tersedia pada modul NLTK.

```
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDistkalimat = "Andi kerap melakukan
transaksi rutin secara daring atau online. Menurut Andi belanja online
lebih praktis & murah."
kalimat =
kalimat.translate(str.maketrans('','',string.punctuation)).lower()

tokens = nltk.tokenize.word_tokenize(kalimat)
kemunculan = nltk.FreqDist(tokens)
print(kemunculan.most_common())# output
# [('andi', 2), ('online', 2), ('kerap', 1), ('melakukan', 1),
('transaksi', 1), ('rutin', 1), ('secara', 1), ('daring', 1), ('atau', 1),
('menurut', 1), ('belanja', 1), ('lebih', 1), ('praktis', 1), ('murah', 1)]
```

Untuk menggambarkan ke dalam bentuk grafik, kita perlu menginstall library Matplotlib.

```
import matplotlib.pyplot as pltkemunculan.plot(30,cumulative=False)
plt.show()
```



Grafik frekuensi kemunculan kata pada dokumen

Tokenizing kalimat

Prinsip yang sama dapat diterapkan untuk memisahkan kalimat pada paragraf. Anda dapat menggunkan kelas `sent_tokenize()` pada modul NLTK. Saya telah menambahkan kalimat pada contoh seperti dibawah ini :

```
# impor sent_tokenize dari modul nltk
from nltk.tokenize import sent_tokenizekalimat = "Andi kerap melakukan
transaksi rutin secara daring atau online. Menurut Andi belanja online
lebih praktis & murah."

tokens = nltk.tokenize.sent_tokenize(kalimat)
print(tokens)# ouput
# ['Andi kerap melakukan transaksi rutin secara daring atau online.',
'Menurut Andi belanja online lebih praktis & murah.']
```

3. Filtering (Stopword Removal)

Filtering adalah tahap mengambil kata-kata penting dari hasil token dengan menggunakan algoritma *stoplist* (membuang kata kurang penting) atau *wordlist* (menyimpan kata penting).

Stopword adalah kata umum yang biasanya muncul dalam jumlah besar dan dianggap tidak memiliki makna. Contoh *stopword* dalam bahasa Indonesia adalah “yang”, “dan”, “di”, “dari”, dll. Makna di balik penggunaan *stopword* yaitu dengan menghapus kata-kata yang memiliki informasi rendah dari sebuah teks, kita dapat fokus pada kata-kata penting sebagai gantinya.

Contoh penggunaan *filtering* dapat kita temukan pada konteks mesin pencarian. Jika permintaan pencarian anda adalah “apa itu pengertian manajemen?” tentunya anda ingin sistem pencarian fokus pada memunculkan dokumen dengan topik tentang “pengertian manajemen” di atas dokumen dengan topik “apa itu”. Hal ini dapat dilakukan dengan mencegah kata dari daftar *stopword* dianalisa.

Nama : Nanda Tri Haryati
NIM / Kelas : 202420016 / MTI23- REG-A

Filtering dengan NLTK

```
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online.
Menurut Andi belanja online lebih praktis & murah."
kalimat =
kalimat.translate(str.maketrans('','',string.punctuation)).lower()

tokens = word_tokenize(kalimat)
listStopword =  set(stopwords.words('indonesian'))

removed = []
for t in tokens:
    if t not in listStopword:
        removed.append(t)

print(removed)# ouput
# ['andi', 'kerap', 'transaksi', 'rutin', 'daring', 'online', 'andi',
'belanja', 'online', 'praktis', 'murah']
```

Filtering dengan sastrawi

Selain untuk *stemming,* library Sastrawi juga mendukung proses *filtering*. Kita dapat menggunakan `stopWordRemoverFactory` dari modul sastrawi.

Untuk melihat daftar *stopword* yang telah didefinisikan dalam library Sastrawi dapat menggunakan kode berikut :

```
from Sastrawi.StopWordRemover.StopWordRemoverFactory import
StopWordRemoverFactoryfactory = StopWordRemoverFactory()
stopwords = factory.get_stop_words()
print(stopwords)
```

Kode diatas akan menampikan *stopword* yang tersedia di library Sastrawi. Prosos *filtering* pada Sastrawi dapat dilihat pada baris kode dibawah :

```
from Sastrawi.StopWordRemover.StopWordRemoverFactory import
StopWordRemoverFactory
from nltk.tokenize import word_tokenizefactory = StopWordRemoverFactory()
stopword = factory.create_stop_word_remover()kalimat = "Andi kerap
melakukan transaksi rutin secara daring atau online. Menurut Andi belanja
online lebih praktis & murah."
kalimat =
kalimat.translate(str.maketrans('','',string.punctuation)).lower()stop =
stopword.remove(kalimat)
tokens = nltk.tokenize.word_tokenize(stop)
print(tokens)# output
# ['andi', 'kerap', 'transaksi', 'rutin', 'daring', 'online', 'andi',
'belanja', 'online', 'praktis', 'murah']
```

Kita dapat menambah atau mengurangi kata pada daftar *stopword* sesuai dengan kebutuhan analisa. Pada dasarnya daftar *stopword* pada library Sastrawi tersimpan di dalam list yang anda lihat disini. Jadi sebenarnya kita tinggal mengubah daftar pada list tersebut. Tetapi hal tersebut

bisa menjadi permasalahan apabila pada suatu kasus kita diharuskan menambahkan *stopword* secara dinamis.

Library Sastrawi dapat mengatasi permasalahan tersebut, perhatikan kode dibawah ini :

```
from Sastrawi.StopWordRemover.StopWordRemoverFactory import
StopWordRemoverFactory, StopWordRemover, ArrayDictionary
from nltk.tokenize import word_tokenize


stop_factory = StopWordRemoverFactory().get_stop_words() #load defaul
stopword
more_stopword = ['daring', 'online'] #menambahkan stopwordkalimat = "Andi
kerap melakukan transaksi rutin secara daring atau online. Menurut Andi
belanja online lebih praktis & murah."
kalimat =
kalimat.translate(str.maketrans('','',string.punctuation)).lower()data =
stop_factory + more_stopword #menggabungkan stopword

dictionary = ArrayDictionary(data)
str = StopWordRemover(dictionary)
tokens = nltk.tokenize.word_tokenize(str.remove(kalimat))

print(tokens)# output
# ['andi', 'kerap', 'transaksi', 'rutin', 'andi', 'belanja', 'praktis',
'murah']
```

Menurut Jim Geovedi pada artikelnya menjelaskan bahwa penyesuaian daftar *stopwords* perlu dilakukan setiap pertama kali melakukan proyek analisa. Memang bukan sesuatu yang melelahkan, tapi jika tidak dilakukan maka ini akan dapat mengakibatkan salah interpretasi terhadap data.

4. Stemming

Stemming adalah proses menghilangkan infleksi kata ke bentuk dasarnya, namun bentuk dasar tersebut tidak berarti sama dengan akar kata (*root word*). Misalnya kata “mendengarkan”, “dengarkan”, “didengarkan” akan ditransformasi menjadi kata “dengar”.

Idenya adalah ketika anda mencari dokumen “cara membuka lemari”, anda juga ingin melihat dokumen yang menyebutkan “cara terbuka lemari” atau “cara dibuka lemari” meskipun terdengar tidak enak. Tentunya anda ingin mencocokan semua variasi kata untuk memunculkan dokumen yang paling relevan.

Stemming dengan NLTK (bahasa inggris)

Ada banyak algoritma yang digunakan untuk *stemming*. Salah satu algoritma yang tertua dan paling populer adalah algoritma Porter. Algoritma ini tersedia dalam modul NLTK melalui kelas`PorterStemmer()`.

```
from nltk.stem import PorterStemmer

ps = PorterStemmer()

kata = ["program", "programs", "programer", "programing", "programers"]
```

Nama : Nanda Tri Haryati
NIM / Kelas : 202420016 / MTI23- REG-A

```
for k in kata:
    print(k, " : ", ps.stem(k))# ouput
# program  :  program
programs  :  program
programer  :  program
programing  :  program
programers  :  program
```

Selain Porter, NLTK juga mendukung algoritma Lancester, WordNet Lemmatizer, dan SnowBall. Sayangnya proses *stemming* Bahasa Indonesia pada modul NLTK belum didukung.

Stemming bahasa indonesia menggunakan Python Sastrawi

Proses *stemming* antara satu bahasa dengan bahasa yang lain tentu berbeda. Contohnya pada teks berbahasa inggris, proses yang diperlukan hanya proses menghilangkan sufiks. Sedangkan pada teks berbahasa Indonesia semua kata imbuhan baik itu sufiks dan prefiks juga dihilangkan.

Untuk melakukan *stemming* bahasa Indonesia kita dapat menggunakan library Python Sastrawi yang sudah kita siapkan di awal. *Library* Sastrawi menerapkan Algoritma Nazief dan Adriani dalam melakukan *stemming* bahasa Indonesia.

```
from Sastrawi.Stemmer.StemmerFactory import StemmerFactoryfactory =
StemmerFactory()
stemmer = factory.create_stemmer()

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online.
Menurut Andi belanja online lebih praktis & murah."hasil =
stemmer.stem(kalimat)print(hasil)# ouput
# andi kerap laku transaksi rutin cara daring atau online turut andi
belanja online lebih praktis murah
```

Apakah kita memerlukan semua tahapan pada text preprocessing?

Tidak ada aturan pasti yang membahas setiap tahapan pada *text preprocessing*. Tentu saja untuk memastikan hasil yang lebih baik dan konsisten semua tahapan harus dilakukan. Untuk memberi gambaran tentang apa yang minimal seharusnya dilakukan, saya telah menguraikan tahapan menjadi **harus dilakukan**, **sebaiknya dilakukan**, dan **tergantung tugas**. Perlu diingat, *less is more*, anda ingin menjaga pendekatan dengan seindah mungkin. Semakin banyak fitur atau tahapan yang anda tambahkan, semakin banyak pula lapisan yang anda harus kupas.

- **Harus dilakukan** meliputi *case folding* (dapat tergantung tugas dalam beberapa kasus)
- **Sebaiknya dilakukan** meliputi normalisasi sederhana — misalnya menstandarkan kata yang hampir sama
- **Tergantung tugas** meliputi normalisasi tingkat lanjut — misalnya mengatasi kata yang tidak biasa, s*topword removal* dan *stemming*

Jadi, untuk mengatasi tugas apapun minimal anda harus melakukan *case folding*. Selanjutnya dapat ditambahkan beberapa normalisasi dasar untuk mendapatkan hasil yang lebih konsisten dan secara bertahap menambahkan tahapan yang lainnya sesuai yang anda inginkan.

Nama : Nanda Tri Haryati
NIM / Kelas : 202420016 / MTI23- REG-A

Sumber :

https://medium.com/@ksnugroho/dasar-text-preprocessing-dengan-python-a4fa52608ffe

Nama : Nanda Tri Haryati
NIM / Kelas : 202420016/ MTI23-REG-A

Tugas 08

Tutorial Text Mining

Dalam overview kali ini, kita akan melakukan beberapa proses dan metode pada pembuatan pola mining untuk mendapatkan sejumlah fitur dalam text. dengan beberapa proses linguistic yang digunkan untuk dilakukan modeling text terkait beberapa kasus.Berikut ini penjelasan, bagaimana pengaplikasinnya.

Text Mining merupakan pengembangan baru dalam pengelolaan teks yang digunakan untuk dilakukan, dalam beberapa kasusu seperti menghapus kata-kata yang tidak terlalu signifikan diperlukan dan masih banyak lagi, berikut penjelasannya:

- **Scale** : Komputer secara keseluruhan sangat kurang kemampuannya untuk menginpretasikan sebuah makna atau arti dari sebuah kata. namun komputer dapat digunakan dalam skala yang tanpa batas. jika membaca banyak buku, banyak ribuan website, jutaan tweet maka kita kesusahan untuk membaca maupun mengambil bagian penting secara keseluruhan, maka dari itu perlu komputer mampu melakukan skala besar seperti itu.
- **Re-contextualization** : dengan text mining, kita dapat menggunakan beberapa text dan menggunakannya secara bersamaan & keseluruhan. Dengan begini kita bisa menggunakannya untuk memahami informasi yang didapat dari text maupun buku yang kita baca. maka dari itu, perlu kita breakdown untuk menciptakan komparasi maupun critical tools. yang dapat digunakan untuk berbagai pengetahuan seputar atribut apa saja yang digunakan untuk authorship secara anonimus atau pseudonymous writing.
- **Summarization** : dalam bentuk agregasi, ekstraksi dan visualisasi maupun pola yang diciptkana dari sebuah model berbagai fitur yang digunakan untuk dapat mengekstrak point-point yang terkandung dalam sebuah literatur maupun sebuah penulisan dari karya ilmiah.

Requirement apa saja yang diperlukan

dalam berbagai macam sumber. anda bisa membaca berbagai macam literatur yang diambil dari berbagai macam website maupun buku. Salah satunya, anda bisa membaca dari iteratur

- Art of Literary Text Analysis — Stefan Sinclair, 2015
- Introduction to Information Retrieval — Manning & Schutz, 2008
- Speech and Language Processing — 3rd edition, Dan Jurafsky & James H. Martin 2017
- Search Engines: Information Retrieval in Practice — Croft, Metzler & Strohman, 2009

Text Summarization

Text Summarization ini dengan Brown Corpus, secara garis besar Automatic Summarization adalah sebuah proses memampatkan sebuah document text dengan software. Bagaimana menyusun sebuah kesimpulan dengan mengambil point-point penting dari sebuah dokumen asli.

Nama : Nanda Tri Haryati
NIM / Kelas : 202420016/ MTI23-REG-A

Teknologi ini dapat membuat sebuah kesimpulan yang koheren untuk mendapatkan variable akun untuk dibuat sebuah writing style dan & syntax.

Dengan proses machine learning didalamnya, kita dapat mencari sebuah subset data yang berisi informasi dari segala set yang ada. seperti teknik kebanyakan di industri sekarang, seperti search engine contohnya, atau summarization untuk dokumen, koleksi gambar dan video.

Dokumen summarization akan menciptkana sebuah representative summary atau abstrak di segala dokumen yang terkait, dengan mencari kalimat paling informatif, sedangkan untuk gambar mencari paling representatif dan paling penting. Untuk video mungkin bisa melihat dari fitur ekstrasi dari event-event yang relevan sekitar kita.

Ada 2 pendekatan yang menjadi bahasan summarization kita yaitu extraction dan abstraction. Extractive method digunakan untuk menseleksi sebuah subset dari kata yang sering muncul, frase atau kalimat di dalam sebuah original text.

lalu digunakan abstraksi metode untuk membuat internal semantic representation & digunakan pula natural language generation technique untuk mendapatkan kesimpulan yang mendekati sesuai dengan yang kita ekspetasikan.

Seperti layaknya sebuah inovasi dari sebuah kata-kata verbal. Riset ini mengacu pada extractive methods, dimana bisa diambil dari koleksi gambar maupun hasil video dari gambar.

di Jupyter Notebook ini, saya menggunakan algoritma textrank untuk mendapatkan extractive text summarization untuk diaplikasikan sesuai dengan algoritma pencarian.

Source Code

Untuk source code bisa dilihat pada gist dibawah ini:

library yang digunakan yaitu numpy, matplotlib serta beberapa komponen NLTK.

Nama : Nanda Tri Haryati
NIM / Kelas : 202420016/ MTI23-REG-A

```
import re
import string
import numpy as np
import matplotlib.pyplot as plt
from nltk import pos_tag
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus.reader.wordnet import NOUN, VERB, ADJ, ADV
from nltk.corpus import brown, stopwords
from nltk.cluster.util import cosine_distance
from operator import itemgetter
%matplotlib
```

Library Text Summarization hosted with ❤ by GitHub view raw

Gambar 1. Library Text Summarization

Selanjutnya menetukan sentence mana yang kita ambil dari corpus brown tersebut.

```
# untuk mendapatkan sentence dari brown
sentences = brown.sents('ca01')

# menghitung panjang sentence
len(sentences)

# lalu kita gabungkan seluruh kalimat tersebut menjadi beberapa dataset
[' '.join(sent) for sent in sentences]
```

brown corpus hosted with ❤ by GitHub view raw

Gambar 2. Memntukan sentence mana yang mau kita gunakan

lalu kita gunakan fungsi ini, untuk cleaning text, memfilter sentence yang diperlukan saja.

Nama : Nanda Tri Haryati
NIM / Kelas : 202420016/ MTI23-REG-A

```
# fungsi2 yang digunakan untuk text cleaner
class TextCleaner():

    # inisialisasi
    def __init__(self):
        self.stop_words = set(stopwords.words("english"))
        self.punctuations = set(string.punctuation)
        self.pos_tags = {
                NOUN: ['NN', 'NNS', 'NNP', 'NNPS', 'PRP', 'PRP$', 'WP', 'WP$'],
                VERB: ['VB', 'VBD', 'VBG', 'VBN', 'VBP', 'VBZ'],
                ADJ: ['JJ', 'JJR', 'JJS'],
                ADV: ['RB', 'RBR', 'RBS', 'WRB']
        }

    # menghapus bagian stop words pada setiap kalimat
    def _remove_stop_words(self, words):
        return [w for w in words if w not in self.stop_words]

    # emgnhapus regex
    def _remove_regex(self):
        self.input_sent = " ".join([w.lower() for w in self.input_sent])
        self.input_sent = re.sub(r"i'm", "i am", self.input_sent)
        self.input_sent = re.sub(r"he's", "he is", self.input_sent)
        self.input_sent = re.sub(r"she's", "she is", self.input_sent)
        self.input_sent = re.sub(r"that's", "that is", self.input_sent)
        self.input_sent = re.sub(r"what's", "what is", self.input_sent)
        self.input_sent = re.sub(r"where's", "where is", self.input_sent)
        self.input_sent = re.sub(r"\'ll", " will", self.input_sent)
        self.input_sent = re.sub(r"\'ve", " have", self.input_sent)
```

Nama : Nanda Tri Haryati
NIM / Kelas : 202420016/ MTI23-REG-A

```
30          self.input_sent = re.sub(r"\'re", " are", self.input_sent)
31          self.input_sent = re.sub(r"\'d", " would", self.input_sent)
32          self.input_sent = re.sub(r"won't", "will not", self.input_sent)
33          self.input_sent = re.sub(r"can't", "cannot", self.input_sent)
34          self.input_sent = re.sub(r"don't", "do not", self.input_sent)
35          patterns = re.finditer("#[\w]*", self.input_sent)
36          for pattern in patterns:
37              self.input_sent = re.sub(pattern.group().strip(), "", self.input_sent)
38          self.input_sent = "".join(ch for ch in self.input_sent if ch not in self.punctuatic
39
40      # tokenisasi
41      def _tokenize(self):
42          return word_tokenize(self.input_sent)
43
44      # melakukan proses POS tag
45      def _process_content_for_pos(self, words):
46          tagged_words = pos_tag(words)
47          pos_words = []
48          for word in tagged_words:
49              flag = False
50              for key, value in self.pos_tags.items():
51                  if word[1] in value:
52                      pos_words.append((word[0], key))
53                      flag = True
54                      break
55              if not flag:
56                  pos_words.append((word[0], NOUN))
57          return pos_words
```

Nama : Nanda Tri Haryati
NIM / Kelas : 202420016/ MTI23-REG-A

```
58
59      # menghapus noise yang tidak diperlukan
60      def _remove_noise(self):
61          self._remove_regex()
62          words = self._tokenize()
63          noise_free_words = self._remove_stop_words(words)
64          return noise_free_words
65
66      # normalisasi text
67      def _normalize_text(self, words):
68          lem = WordNetLemmatizer()
69          pos_words = self._process_content_for_pos(words)
70          normalized_words = [lem.lemmatize(w, pos=p) for w, p in pos_words]
71          return normalized_words
72
73      # cleaning data
74      def clean_up(self, input_sent):
75          self.input_sent = input_sent
76          cleaned_words = self._remove_noise()
77          cleaned_words = self._normalize_text(cleaned_words)
78          return cleaned_words
```

text cleaner hosted with ❤ by GitHub view raw

Gambar 3. Cleaning text

melakukan proses algoritma ranking untuk menentukan sentence mana yang paling utama sesuai urutannya.

```
1   # page rank algorithm
2   def pagerank(M, eps=1.0e-8, d=0.85):
3       N = M.shape[1]
4       v = np.random.rand(N, 1)
5       v = v / np.linalg.norm(v, 1)
6       last_v = np.ones((N, 1), dtype=np.float32) * np.inf
7       M_hat = (d * M) + (((1 - d) / N) * np.ones((N, N), dtype=np.float32))
8
9       while np.linalg.norm(v - last_v, 2) > eps:
10          last_v = v
11          v = np.matmul(M_hat, v)
12      return v
```

gistfile1.txt hosted with ❤ by GitHub view raw

Gambar 4. Page Rank Algorithm

Selanjutnya penggunaan fungsi cosine similarity untuk menentukan dari setiap sentence tersebut.

Nama : Nanda Tri Haryati
NIM / Kelas : 202420016/ MTI23-REG-A

```python
# sentence similarity
def sentence_similarity(sent1, sent2):
    text_cleaner = TextCleaner()

    sent1 = text_cleaner.clean_up(sent1)
    sent2 = text_cleaner.clean_up(sent2)

    all_words = list(set(sent1 + sent2))

    vector1 = [0] * len(all_words)
    vector2 = [0] * len(all_words)

    for w in sent1:
        vector1[all_words.index(w)] += 1

    for w in sent2:
        vector2[all_words.index(w)] += 1

    return 1 - cosine_distance(vector1, vector2)
```

gistfile1.txt hosted with ❤ by GitHub view raw

Gambar 5. Sentence Similarity

lalu lakukan proses adjency matrix berdasarkan kesamaan dari setiap sentence tersebut.

```python
# similaritas berdasarkan adjency matrix
def build_similarity_matrix(sentences):
    S = np.zeros((len(sentences), len(sentences)))
    for i in range(len(sentences)):
        for j in range(len(sentences)):
            if i == j:
                continue
            else:
                S[i][j] = sentence_similarity(sentences[i], sentences[j])

    for i in range(len(S)):
        S[i] /= S[i].sum()
    return S
```

gistfile1.txt hosted with ❤ by GitHub view raw

Gambar 6. adjency matrix similarity

Nama : Nanda Tri Haryati
NIM / Kelas : 202420016/ MTI23-REG-A

Debug proses diatas untuk menampilkan matriks array tersebut

```
# building similiarity
S = build_similarity_matrix(sentences)
S
```

gistfile1.txt hosted with ❤ by GitHub view raw

Gambar 7. Building Similarity

Lalu kita proses sesuai dengan diranking sesuai dengan kalimat penting di letakkan di paling atas.

```
# sentence ranking sesuai prioritasnya
sentence_ranks = pagerank(S)
sentence_ranks
```

gistfile1.txt hosted with ❤ by GitHub view raw

Gambar 8. Sentence Ranking

Dilanjutkan dengan ranking sesuai dengan index

```
# ranking sesuai dengan index yang dimuat
ranked_sentence_indexes = [item[0] for item in sorted(enumerate(sentence_ranks), key=lambda
ranked_sentence_indexes
```

gistfile1.txt hosted with ❤ by GitHub view raw

Gambar 9. Index Ranking

Kemudian petakan sesuai dengan ranking-ranking yang didapatkan.

```
# pemetakan sesuai dengan ranking
plt.bar([item[0] for item in sorted(enumerate(sentence_ranks))], sentence_ranks.T[0])
plt.xlabel("Sentence No.")
plt.ylabel("Importance")
plt.show()
```

gistfile1.txt hosted with ❤ by GitHub view raw

Gambar 10. Plot Rank 1

Nama : Nanda Tri Haryati
NIM / Kelas : 202420016/ MTI23-REG-A

Lalu kita buat plot rank untuk proses smooting

```
# kita buat plot lebih smooth unutk mendapatkan nilai ambangnya
plt.plot([item[0] for item in sorted(enumerate(sentence_ranks))], sentence_ranks)
plt.xlabel("Sentence No.")
plt.ylabel("Importance")
plt.show()
```

gistfile1.txt hosted with ❤ by GitHub view raw

Gambar 11. Plot Rank 2

Ambil nilai summary yang paling penting dari setiap sentence

```
# ambil 5 paling penting
SUMMARY_SIZE = 5
selected_sentences = sorted(ranked_sentence_indexes[:SUMMARY_SIZE])
selected_sentences
```

gistfile1.txt hosted with ❤ by GitHub view raw

Gambar 12. Summary 5 paling penting

Kita dapatkan hasil summary kita.

```
# Generate Summary
summary = itemgetter(*selected_sentences)(sentences)
for sent in summary:
    print(' '.join(sent))
```

gistfile1.txt hosted with ❤ by GitHub view raw

Gambar 13. Generasi hasil kesimpulannya

Begitulah, kira-kira salah satu contoh untuk menentukan summary dari sebuah dasar penggnaan text mining, terima kasih..

Sumber :

- https://towardsdatascience.com/text-mining-for-dummies-text-classification-with-python-98e47c3a9deb
- Industry 4.0 : Emerging themes & future research avenue using a text mining approach https://doi.org/10.1016/j.compind.2019.04.018
- Chapter 7 — Using Natural Language Tools in Forencis https://doi.org/10.1016/B978-0-12-418676-7.00007-4

Nama : Nanda Tri Haryati
NIM / Kelas : 202420016/ MTI23-REG-A

- A method for determining the number of ducuments needed for a gold standard corpus
  https://doi.org/10.1016/j.jbi.2011.12.010

Tugas 8

Nama : Oman Arrohman
NIM : 202420042
Kelas : MTI Reguler A
MK : Advanced Database

Dalam text mining dikenal istilah *Text Preprocessing*. *Text Preprocessing* adalah tahapan dimana kita melakukan seleksi data agar data yang akan kita olah menjadi lebih terstruktur. Disini dijelaskan bagiamana melakukan proses *Text Preprocessing* menggunakan *Python* dengan *Library NLTK.*

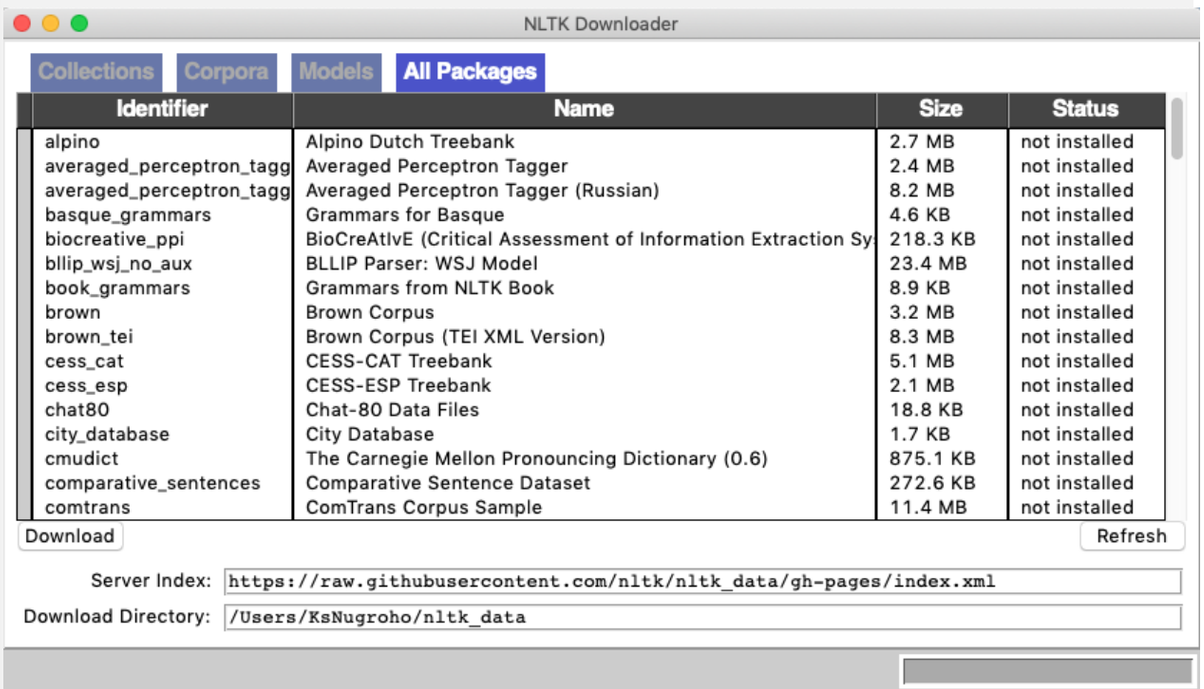
Natural Language Toolkit (NLTK)

Natural Language Toolkit atau disingkat NLTK, adalah *libray* python untuk bekerja dengan permodelan teks. NLTK menyediakan alat yang baik mempersiapkan teks sebelum digunakan pada *machine learning* atau algoritma *deep learning*. Cara termudah untuk menginstall NLTK adalah menggunakan "pip" pada *command line/terminal*.

```
pip install nltk
```

Langkah pertama yang perlu anda lakukan setelah menginstall *NLTK* adalah mengunduh paket *NLTK.*

```
import nltk
nltk.download()
```



NLTK

Tidak ada aturan pasti tentang setiap tahapan didalam proses Text Preprocessing semua tergantung dari jenis data (dokumen teks) dan hasil yang diinginkan. Namun pada umumnya tahapan proses text preprocessing adalah Case Folding, Tokenization dan Filtering, Stopword Removal, Stemming. Untuk memberi gambaran tentang apa yang minimal seharusnya dilakukan, adalah harus dilakukan, sebaiknya dilakukan, dan tergantung tugas.

- Harus dilakukan meliputi *case folding* (dapat tergantung tugas dalam beberapa kasus)
- Sebaiknya dilakukan meliputi normalisasi sederhana — misalnya menstandarkan kata yang hampir sama
- Tergantung tugas meliputi normalisasi tingkat lanjut — misalnya mengatasi kata yang tidak biasa, s*topword removal* dan *stemming*

Jadi, untuk mengatasi tugas apapun minimal anda harus melakukan *case folding*. Selanjutnya dapat ditambahkan beberapa normalisasi dasar untuk mendapatkan hasil yang lebih konsisten dan secara bertahap menambahkan tahapan yang lainnya sesuai yang anda inginkan.

Contoh teks yang akan diproses :

"[MOJOK.co] Manfaat jogging setiap pagi yang pertama adalah meredakan stres. Olahraga itu seperti kode bagi tubuh untuk memproduksi hormon endorfin, agen perangsang rasa bahagia. Dilakukan di pagi hari, ketika udara masih bersih, sejuk, jalanan lengang, gunung terlihat jelas di sebelah utara, manfaat jogging bisa kamu rasakan secara maksimal."

- Case Folding

Dalam sebuah dokumen penggunan huruf kapital atau sejenisnya terkadang tidak mempunyai kesamaan, hal ini bisa dikarenakan kesalahan penulisan. Dalam text preprocessing proses case folding bertujuan untuk mengubah semua huruf dalam sebuah dokumen teks menjadi huruf kecil (lowercase). Untuk proses ini tidak perlu menggunakan library NLTK, kita bisa menggunakan fungsi lower() yang merupakaan bawaah dari python.

```
kalimat = "[MOJOK.co] Manfaat jogging setiap pagi yang pertama adalah meredakan stres. Olahraga itu seperti kode bagi tubuh untuk memproduksi hormon endorfin, agen perangsang rasa bahagia."
case_folding = kalimat.lower()
print(case_folding)
```

- Tokenization

Dokumen teks terdiri dari sekumpulan kalimat, proses tokenization memecah dokumen tersebut menjadi bagian-bagian kata yang disebut token. Melakukan tokenization bisa menggunakan library NLTK

```
import nltk
```

```
kalimat = "[MOJOK.co] Manfaat jogging setiap pagi yang pertama adalah meredakan stres. Olahraga itu seperti kode bagi tubuh untuk memproduksi hormon endorfin, agen perangsang rasa bahagia. Dilakukan di pagi hari, ketika udara masih bersih, sejuk, jalanan lengang, gunung terlihat jelas di sebelah utara, manfaat jogging bisa kamu rasakan secara maksimal."

tokens = nltk.tokenize.word_tokenize(kalimat)
print(tokens)

# Hasil
# ['[', 'MOJOK.co', ']', 'Manfaat', 'jogging', 'setiap', 'pagi', 'yang', 'pertama', 'adalah', 'meredakan', 'stres', '.', 'Olahraga', 'itu', 'seperti', 'kode', 'bagi', 'tubuh', 'untuk', 'memproduksi', 'hormon', 'endorfin', ',', 'agen', 'perangsang', 'rasa', 'bahagia', '.', 'Dilakukan', 'di', 'pagi', 'hari', ',', 'ketika', 'udara', 'masih', 'bersih', ',', 'sejuk', ',', 'jalanan', 'lengang', ',', 'gunung', 'terlihat', 'jelas', 'di', 'sebelah', 'utara', ',', 'manfaat', 'jogging', 'bisa', 'kamu', 'rasakan', 'secara', 'maksimal', '.']
```

Diatas diketahui jumlah kemunculan kata pada sebuah dokumen. Terdapat 6 kemunculan **tanda koma**, 3 kemunculan **tanda titik**, 2 kemunculan kata **jogging** dan sterusnya. Namun kemunculan tanda baca sebaiknya dihindari karena dapat nantinya menggangu proses perhitungan dalam penerapan algoritma Text Mining, kita bisa melakukan filtering terhadap kalimat tersebut untuk menghilangkan tanda baca dan karakter non-alfabet.

```
import nltk
import string

kalimat = "[MOJOK.co] Manfaat jogging setiap pagi yang pertama adalah meredakan stres. Olahraga itu seperti kode bagi tubuh untuk memproduksi hormon endorfin, agen perangsang rasa bahagia. Dilakukan di pagi hari, ketika udara masih bersih, sejuk, jalanan lengang, gunung terlihat jelas di sebelah utara, manfaat jogging bisa kamu rasakan secara maksimal."
kalimat = kalimat.translate(str.maketrans("",string.punctuation)).lower()

tokens = nltk.tokenize.word_tokenize(kalimat)
kemunculan = nltk.FreqDist(tokens)
print(kemunculan.most_common())

# Hasil
# [('jogging', 2), ('pagi', 2), ('di', 2), ('MOJOKco', 1), ('Manfaat', 1), ('setiap', 1), ('yang', 1), ('pertama', 1), ('adalah', 1), ('meredakan', 1), ('stres', 1), ('Olahraga', 1), ('itu', 1), ('seperti', 1), ('kode', 1), ('bagi', 1), ('tubuh', 1), ('untuk', 1), ('memproduksi', 1), ('hormon', 1), ('endorfin', 1), ('agen', 1), ('perangsang', 1), ('rasa', 1), ('bahagia', 1), ('Dilakukan', 1), ('hari', 1), ('ketika', 1), ('udara', 1), ('masih', 1), ('bersih', 1), ('sejuk', 1), ('jalanan', 1), ('lengang', 1), ('gunung', 1), ('terlihat', 1), ('jelas', 1), ('sebelah', 1), ('utara', 1), ('manfaat', 1), ('bisa', 1), ('kamu', 1), ('rasakan', 1), ('secara', 1), ('maksimal', 1)]
```

- Stopword Removal

Tahapan ini akan mengambil kata-kata yang dianggap penting dari hasil tokenization atau membuang kata-kata yang dianggap tidak terlalu mempunyai arti penting dalam proses text mining. Jika menggunakan NLTK dengan cara:

```
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
import string

kalimat = "[MOJOK.co] Manfaat jogging setiap pagi yang pertama adalah meredakan stres. Olahraga itu seperti kode bagi tubuh untuk memproduksi hormon endorfin, agen perangsang rasa bahagia. Dilakukan di pagi hari, ketika udara masih bersih, sejuk, jalanan lengang, gunung terlihat jelas di sebelah utara, manfaat jogging bisa kamu rasakan secara maksimal."
kalimat = kalimat.translate(str.maketrans("",string.punctuation)).lower()

tokens = word_tokenize(kalimat)
listStopword =  set(stopwords.words('indonesian'))

removed = []
for t in tokens:
    if t not in listStopword:
        removed.append(t)

print(removed)
# Hasil
# ['mojokco', 'manfaat', 'jogging', 'pagi', 'meredakan', 'stres', 'olahraga', 'kode', 'tubuh', 'memproduksi', 'hormon', 'endorfin', 'agen', 'perangsang', 'bahagia', 'pagi', 'udara', 'bersih', 'sejuk', 'jalanan', 'lengang', 'gunung', 'sebelah', 'utara', 'manfaat', 'jogging', 'rasakan', 'maksimal']
```

- Stemming

Steeming bertujuan untuk mentransformasikan kata menjadi kata dasarnya (root word) dengan menghilangkan semua imbuhan kata. Sayangnya proses stemming Bahasa Indonesia menggunakan NLTK belum didukung.

```
from nltk.stem import PorterStemmer

st = PorterStemmer()
print(st.stem('monitoring'))
# monitor
print(st.stem('stemmed'))
#stem
```

Selanjutnya Untuk steeming bahasa indonesia bisa menggunakan python bisa menggunakan library Python Sastrawi.

- Python Sastrawi (Stemming Bahasa Indonesia)

Python Sastrawi adalah pengembangan dari proyek PHP Sastrawi Python Sastrawi merupakan library sederhana yang dapat mengubah kata berimbuhan bahasa Indonesia menjadi bentuk dasarnya. Sastrawi juga dapat diinstal melalui “pip”.

```
pip install Sastrawi
```

Penggunaan Python Sastrawi sebagai berikut :

```
# import StemmerFactory class
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory# create stemmer
factory = StemmerFactory()
stemmer = factory.create_stemmer()# stemming process
sentence = 'Perekonomian Indonesia sedang dalam pertumbuhan yang membanggakan'output
= stemmer.stem(sentence)print(output)
# ekonomi indonesia sedang dalam tumbuh yang banggaprint(stemmer.stem('Mereka meniru-
nirukannya'))
# mereka tiru
```

- Stemming dengan NLTK (bahasa inggris)

Ada banyak algoritma yang digunakan untuk *stemming*. Salah satu algoritma yang tertua dan paling populer adalah algoritma porter. Algoritma ini tersedia dalam modul NLTK melalui kelas`PorterStemmer()`.

```
from nltk.stem import PorterStemmer

ps = PorterStemmer()

kata = ["program", "programs", "programer", "programing", "programers"]

for k in kata:
    print(k, " : ", ps.stem(k))# ouput
# program  :  program
programs  :  program
programer  :  program
programing  :  program
programers  :  program
```

Nama : Puspita Dewi Setyadi
NIM : 202420011
Kelas : MTI 23 A

Salah satu keunggulan python adalah mendukung banyak *open-source library*. Ada banyak *library* python yang dapat digunakan untuk melakukan dan mengimplementasikan masalah dalam NLP.

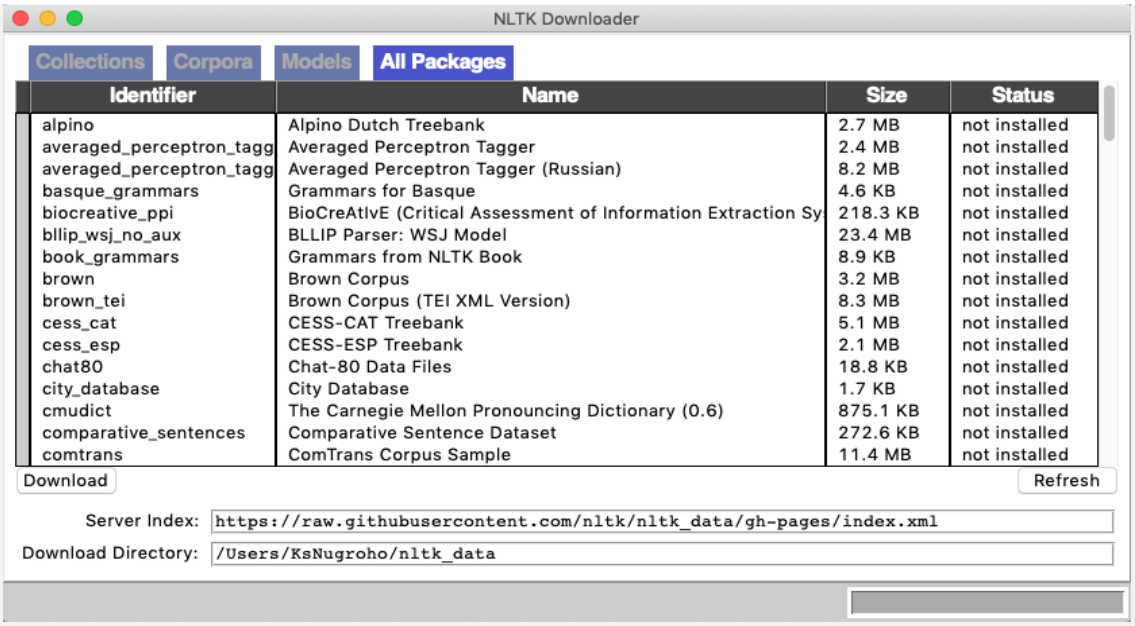
Natural Language Toolkit (NLTK)

Natural Language Toolkit atau disingkat NLTK, adalah *libray* python untuk bekerja dengan permodelan teks. NLTK menyediakan alat yang baik mempersiapkan teks sebelum digunakan pada *machine learning* atau algoritma *deep learnin*g. Cara termudah untuk menginstall NLTK adalah menggunakan "pip" pada *command line*/*terminal*.

```
pip install nltk
```

Langkah pertama yang perlu anda lakukan setelah menginstall NLTK adalah mengunduh paket NLTK.

```
Import nltk
nltk.download()
```



NLTK

Python Sastrawi

Python Sastrawi adalah pengembangan dari proyek PHP Sastrawi. Python Sastrawi merupakan library sederhana yang dapat mengubah kata berimbuhan bahasa Indonesia menjadi bentuk dasarnya. Sastrawi juga dapat diinstal melalui “pip”.

```
pip install Sastrawi
```

Penggunaan Python Sastrawi sangat sederhana seperti baris kode dibawah ini :

```
# import StemmerFactory class
from Sastrawi.Stemmer.StemmerFactory import
StemmerFactory

# create stemmer
factory = StemmerFactory()
stemmer = factory.create_stemmer()
```

```
# stemming process
sentence = 'Perekonomian Indonesia sedang dalam pertumbuhan yang membanggakan'

output   = stemmer.stem(sentence)

print(output)
# ekonomi indonesia sedang dalam tumbuh yang bangga

print(stemmer.stem('Mereka meniru-nirukannya'))
# mereka tiru
```

1. Case folding

Case folding adalah salah satu bentuk *text preprocessing* yang paling sederhana dan efektif meskipun sering diabaikan. Tujuan dari *case folding* untuk mengubah semua huruf dalam dokumen menjadi huruf kecil. Hanya huruf 'a' sampai 'z' yang diterima. Karakter selain huruf dihilangkan dan dianggap *delimiter*. Pada tahap ini tidak menggunakan *external library* apapun, kita bisa memanfaatkan modul yang tersedia di python.

Ada beberapa cara yang dapat digunakan dalam tahap *case folding*, anda dapat menggunakan beberapa atau menggunakan semuanya, tergantung pada tugas yang diberikan.

Mengubah text menjadi lowercase

Salah satu contoh pentingnya penggunaan *lower case* adalah untuk mesin pencarian. Bayangkan anda sedang mencari dokumen yang mengandung "indonesia" namun tidak ada hasil yang muncul karena "indonesia" di indeks sebagai "INDONESIA". Siapa yang harus kita salahkan? U.I. *designer* yang mengatur *user interface* atau *developer* yang mengatur

sistem dalam indeks pencarian? Contoh dibawah menunjukan bagaimana python mengubah teks menjadi *lowercase* :

```
kalimat = "Berikut ini adalah 5 negara dengan pendidikan terbaik di dunia adalah Korea Selatan, Jepang, Singapura, Hong Kong, dan Finlandia."
lower_case = kalimat.lower()
print(lower_case)
# output
# berikut ini adalah 5 negara dengan pendidikan terbaik di dunia adalah korea selatan, jepang, singapura, hong kong, dan finlandia.
```

Menghapus angka

Hapuslah angka jika tidak relevan dengan apa yang akan anda analisa, contohnya seperti nomor rumah, nomor telepon, dll. *Regular expression* (*regex*) dapat digunakan untuk menghapus karakter angka. Python memiliki modulre untuk melakukan hal – hal yang berkaitan dengan *regex*.

Contoh dibawah menunjukan bagaimana python menghapus angka dalam sebuah kalimat :

```
import re # impor modul regular expression
kalimat = "Berikut ini adalah 5 negara dengan pendidikan terbaik di dunia adalah Korea Selatan, Jepang, Singapura, Hong Kong, dan Finlandia."
hasil = re.sub(r"\d+", "", kalimat)
print(hasil)
```

```
# ouput
# Berikut ini adalah negara dengan pendidikan terbaik di dunia adalah Korea Selatan, Jepang, Singapura, Hong Kong, dan Finlandia.
```

Menghapus tanda baca

Sama halnya dengan angka, tanda baca dalam kalimat tidak memiliki pengaruh pada *text preprocessing*. Menghapus tanda baca seperti [!”#$%&’()*+,-./:;<=>?@[\]^_`{|}~] dapat dilakukan di pyhton seperti dibawah ini :

```
kalimat = "Ini &adalah [contoh] kalimat? {dengan} tanda. baca?!!"
hasil =
 kalimat.translate(str.maketrans("","",string.punctuation))
print(hasil)


# output
# Ini adalah contoh kalimat dengan tanda baca
```

Menghapus whitepace (karakter kosong)

Untuk menghapus spasi di awal dan akhir, anda dapat menggunakan fungsi strip() pada pyhton. Perhatikan kode dibawah ini :

```
kalimat = " \t ini kalimat contoh\t "
hasil = kalimat.strip()
print(hasil)


# output
# ini kalimat contoh
```

2. Tokenizing

Tokenizing adalah proses pemisahan teks menjadi potongan-potongan yang disebut sebagai token untuk kemudian di analisa. Kata, angka, simbol, tanda baca dan entitas penting lainnya dapat dianggap sebagai token. Didalam NLP, token diartikan sebagai "kata" meskipun *tokenize* juga dapat dilakukan pada paragraf maupun kalimat.

Fungsi split() pada pyhton dapat digunakan untuk memisahkan teks. Perhatikan contoh dibawah ini :

```
kalimat = "rumah idaman adalah rumah yang bersih"
pisah = kalimat.split()
print(pisah)


# output
# ['rumah', 'idaman', 'adalah', 'rumah', 'yang', 'bersih']
```

Sayangnya, tahap *tokenizing* tidak sesederhana itu. Dari *output* kode diatas kita akan mengolah kata "rumah" dan "rumah" sebagai 2 entitas yang berbeda. Permasalahan ini tentunya akan mempengaruhi hasil dari analisa teks itu sendiri.

Untuk mengatasi masalah ini kita dapat menggunakan modul dari NLTK yang sudah diinstal sebelumnya.

Tokenizing kata

Sebuah kalimat atau data dapat dipisah menjadi kata-kata dengan kelas word_tokenize() pada modul NLTK.

```
# impor word_tokenize dari modul nltk
from nltk.tokenize import word_tokenize

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online."

tokens = nltk.tokenize.word_tokenize(kalimat)
print(tokens)

# ouput
# ['Andi', 'kerap', 'melakukan', 'transaksi', 'rutin', 'secara', 'daring', 'atau', 'online', '.']
```

Dari *output* kode diatas terdapat kemunculan tanda baca titik(.) dan koma (,) serta token "Andi" yang masih menggunakan huruf besar pada awal kata. Hal tersebut nantinya dapat menggangu proses perhitungan dalam penerapan algoritma. Jadi, sebaiknya teks telah melewati tahap *case folding* sebelum di *tokenize* agar menghasilkan hasil yang lebih konsisten. Kita dapat menambahkan fungsi *case folding* untuk menghilangkan tanda baca dan mengubah *teks* ke dalam bentuk *lowercase*.

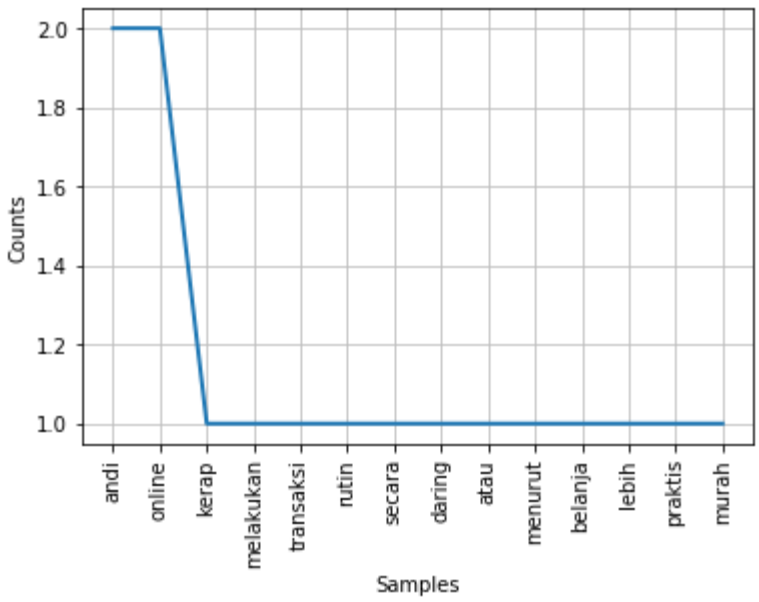
```
kalimat = kalimat.translate(str.maketrans("",string.punctuation)).lower()

# output
# ['andi', 'kerap', 'melakukan', 'transaksi', 'rutin', 'secara', 'daring', 'atau', 'online']
```

Kita juga bisa mendapatkan informasi frekuensi kemunculan setiap token dengan kelas FreqDist() yang sudah tersedia pada modul NLTK.

```
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDist
kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online. Menurut Andi belanja online lebih praktis & murah."
kalimat = kalimat.translate(str.maketrans("",string.punctuation)).lower()
tokens = nltk.tokenize.word_tokenize(kalimat)
kemunculan = nltk.FreqDist(tokens)
print(kemunculan.most_common())
# output
# [('andi', 2), ('online', 2), ('kerap', 1), ('melakukan', 1), ('transaksi', 1), ('rutin', 1), ('secara', 1), ('daring', 1), ('atau', 1), ('menurut', 1), ('belanja', 1), ('lebih', 1), ('praktis', 1), ('murah', 1)]
```

Untuk menggambarkan ke dalam bentuk grafik, kita perlu menginstall library Matplotlib.

```
import matplotlib.pyplot as plt
kemunculan.plot(30,cumulative=False)
plt.show()
```



Grafik frekuensi kemunculan kata pada dokumen

Tokenizing kalimat

Prinsip yang sama dapat diterapkan untuk memisahkan kalimat pada paragraf. Anda dapat menggunkan kelas sent_tokenize() pada modul NLTK. Saya telah menambahkan kalimat pada contoh seperti dibawah ini :

```
# impor sent_tokenize dari modul nltk
from nltk.tokenize import sent_tokenize

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online. Menurut Andi belanja online lebih praktis & murah."

tokens = nltk.tokenize.sent_tokenize(kalimat)
print(tokens)
```

```
# ouput
# ['Andi kerap melakukan transaksi rutin secara daring atau online.', 'Menurut Andi belanja online lebih praktis & murah.']
```

3. Filtering (Stopword Removal)

Filtering adalah tahap mengambil kata-kata penting dari hasil token dengan menggunakan algoritma *stoplist* (membuang kata kurang penting) atau *wordlist* (menyimpan kata penting).

Stopword adalah kata umum yang biasanya muncul dalam jumlah besar dan dianggap tidak memiliki makna. Contoh *stopword* dalam bahasa Indonesia adalah “yang”, “dan”, “di”, “dari”, dll. Makna di balik penggunaan *stopword* yaitu dengan menghapus kata-kata yang memiliki informasi rendah dari sebuah teks, kita dapat fokus pada kata-kata penting sebagai gantinya.

Contoh penggunaan *filtering* dapat kita temukan pada konteks mesin pencarian. Jika permintaan pencarian anda adalah “apa itu pengertian manajemen?” tentunya anda ingin sistem pencarian fokus pada memunculkan dokumen dengan topik tentang “pengertian manajemen” di atas dokumen dengan topik “apa itu”. Hal ini dapat dilakukan dengan mencegah kata dari daftar *stopword* dianalisa.

Filtering dengan NLTK

```
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online. Menurut Andi belanja online lebih praktis & murah."
```

```
kalimat = kalimat.translate(str.maketrans("",string.punctuation)).lower()


tokens = word_tokenize(kalimat)
listStopword =  set(stopwords.words('indonesian'))


removed = []
for t in tokens:
    if t not in listStopword:
        removed.append(t)


print(removed)


# ouput
# ['andi', 'kerap', 'transaksi', 'rutin', 'daring', 'online', 'andi', 'belanja', 'online', 'praktis', 'murah']
```

Filtering dengan sastrawi

Selain untuk *stemming,* library Sastrawi juga mendukung proses *filtering*. Kita dapat menggunakan **stopWordRemoverFactory** dari modul sastrawi.

Untuk melihat daftar *stopword* yang telah didefinisikan dalam library Sastrawi dapat menggunakan kode berikut :

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

```
factory = StopWordRemoverFactory()
stopwords = factory.get_stop_words()
print(stopwords)
```

Kode diatas akan menampikan *stopword* yang tersedia di library Sastrawi. Prosos *filtering* pada Sastrawi dapat dilihat pada baris kode dibawah :

```
from Sastrawi.StopWordRemover.StopWordRemoverFactory
import StopWordRemoverFactory
from nltk.tokenize import word_tokenize

factory = StopWordRemoverFactory()
stopword = factory.create_stop_word_remover()

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online. Menurut Andi belanja
online lebih praktis & murah."
kalimat = kalimat.translate(str.maketrans("",string.punctuation)).lower()

stop = stopword.remove(kalimat)
tokens = nltk.tokenize.word_tokenize(stop)
print(tokens)

# output
# ['andi', 'kerap', 'transaksi', 'rutin', 'daring', 'online', 'andi', 'belanja', 'online', 'praktis', 'murah']
```

Kita dapat menambah atau mengurangi kata pada daftar *stopword* sesuai dengan kebutuhan analisa. Pada dasarnya daftar *stopword* pada library Sastrawi tersimpan di dalam list yang anda lihat disini. Jadi sebenarnya kita tinggal mengubah daftar pada list tersebut. Tetapi hal tersebut bisa menjadi permasalahan apabila pada suatu kasus kita diharuskan menambahkan *stopword* secara dinamis.

Library Sastrawi dapat mengatasi permasalahan tersebut, perhatikan kode dibawah ini :

```
from Sastrawi.StopWordRemover.StopWordRemoverFactory
import StopWordRemoverFactory, StopWordRemover, ArrayDictionary
from nltk.tokenize import word_tokenize

stop_factory = StopWordRemoverFactory().get_stop_words() #load defaul stopword
more_stopword = ['daring', 'online'] #menambahkan stopword

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online. Menurut Andi belanja online lebih praktis & murah."
kalimat = kalimat.translate(str.maketrans("",string.punctuation)).lower()

data = stop_factory + more_stopword #menggabungkan stopword

dictionary = ArrayDictionary(data)
str = StopWordRemover(dictionary)
tokens = nltk.tokenize.word_tokenize(str.remove(kalimat))
```

```
print(tokens)


# output
# ['andi', 'kerap', 'transaksi', 'rutin', 'andi', 'belanja', 'praktis', 'murah']
```

Menurut Jim Geovedi pada artikelnya menjelaskan bahwa penyesuaian daftar *stopwords* perlu dilakukan setiap pertama kali melakukan proyek analisa. Memang bukan sesuatu yang melelahkan, tapi jika tidak dilakukan maka ini akan dapat mengakibatkan salah interpretasi terhadap data.

4. Stemming

Stemming adalah proses menghilangkan infleksi kata ke bentuk dasarnya, namun bentuk dasar tersebut tidak berarti sama dengan akar kata (*root word*). Misalnya kata “mendengarkan”, “dengarkan”, “didengarkan” akan ditransformasi menjadi kata “dengar”.

Idenya adalah ketika anda mencari dokumen “cara membuka lemari”, anda juga ingin melihat dokumen yang menyebutkan “cara terbuka lemari” atau “cara dibuka lemari” meskipun terdengar tidak enak. Tentunya anda ingin mencocokan semua variasi kata untuk memunculkan dokumen yang paling relevan.

Stemming dengan NLTK (bahasa inggris)

Ada banyak algoritma yang digunakan untuk *stemming*. Salah satu algoritma yang tertua dan paling populer adalah algoritma Porter. Algoritma ini tersedia dalam modul NLTK melalui kelas PorterStemmer().

```
from nltk.stem import PorterStemmer
ps = PorterStemmer()
```

```
kata = ["program", "programs", "programer", "programing", "programers"]
for k in kata:
    print (k, " : ", ps.stem(k))
# ouput
# program  :  program
programs  :  program
programer  :  program
programing  :  program
programers  :  program
```

Selain Porter, NLTK juga mendukung algoritma Lancester, WordNet Lemmatizer, dan SnowBall. Sayangnya proses *stemming* Bahasa Indonesia pada modul NLTK belum didukung.

Stemming bahasa indonesia menggunakan Python Sastrawi

Proses *stemming* antara satu bahasa dengan bahasa yang lain tentu berbeda. Contohnya pada teks berbahasa inggris, proses yang diperlukan hanya proses menghilangkan sufiks. Sedangkan pada teks berbahasa Indonesia semua kata imbuhan baik itu sufiks dan prefiks juga dihilangkan. Untuk melakukan *stemming* bahasa Indonesia kita dapat menggunakan library Python Sastrawi yang sudah kita siapkan di awal. *Library* Sastrawi menerapkan Algoritma Nazief dan Adriani dalam melakukan *stemming* bahasa Indonesia.

```
from Sastrawi.Stemmer.StemmerFactory
import StemmerFactory


factory = StemmerFactory()
```

```
stemmer = factory.create_stemmer()
kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online. Menurut Andi belanja online lebih praktis & murah."
hasil = stemmer.stem(kalimat)
print(hasil)
# ouput
# andi kerap laku transaksi rutin cara daring atau online turut andi belanja online lebih praktis murah
```

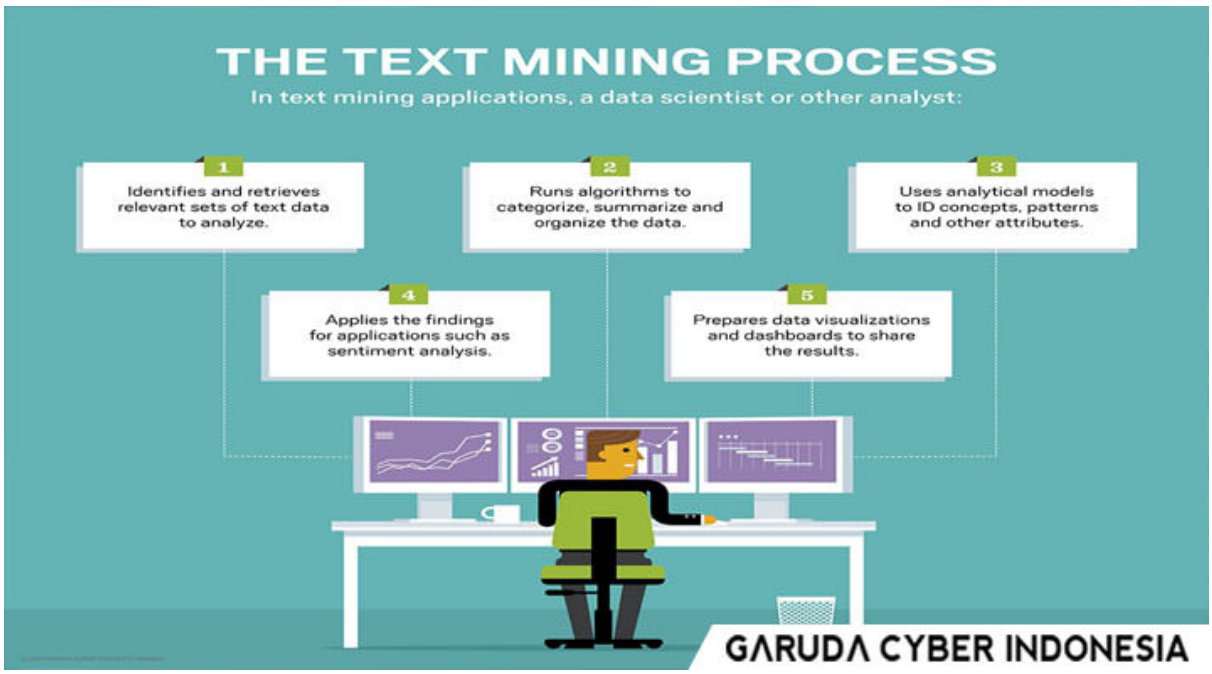
Tugas 08

Text Mining

Text mining adalah proses mengeksplorasi dan menganalisis sejumlah besar data teks tidak terstruktur yang dibantu oleh perangkat lunak yang dapat mengidentifikasi konsep, pola, topik, kata kunci, dan atribut lainnya dalam data. Ini juga dikenal sebagai analisis teks, meskipun beberapa orang menarik perbedaan antara dua istilah; dalam pandangan itu, analitik teks adalah aplikasi yang diaktifkan oleh penggunaan teknik text mining untuk memilah-milah set data.

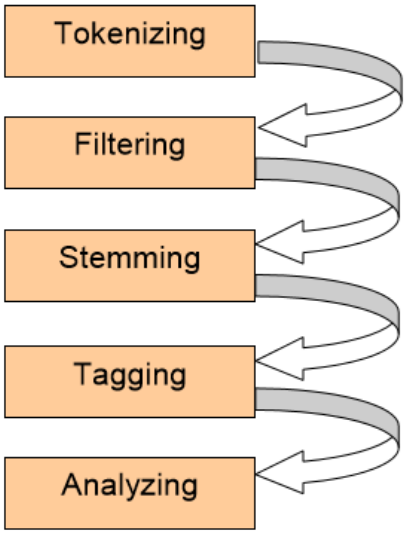
Bagaimana Cara Kerja Text Mining ??

Text mining memiliki sifat yang mirip dengan data mining, tetapi dengan fokus pada teks daripada bentuk data yang lebih terstruktur. Namun, salah satu langkah pertama dalam proses penambangan teks adalah mengatur dan menyusun data dengan cara tertentu sehingga dapat menjadi sasaran analisis kualitatif dan kuantitatif. Melakukannya secara khusus melibatkan penggunaan teknologi natural language processing (NLP), yang menerapkan prinsip-prinsip linguistik komputasional untuk menguraikan dan menginterpretasikan set data.



https://garudacyber.co.id/artikel/1254-apa-itu-text-mining

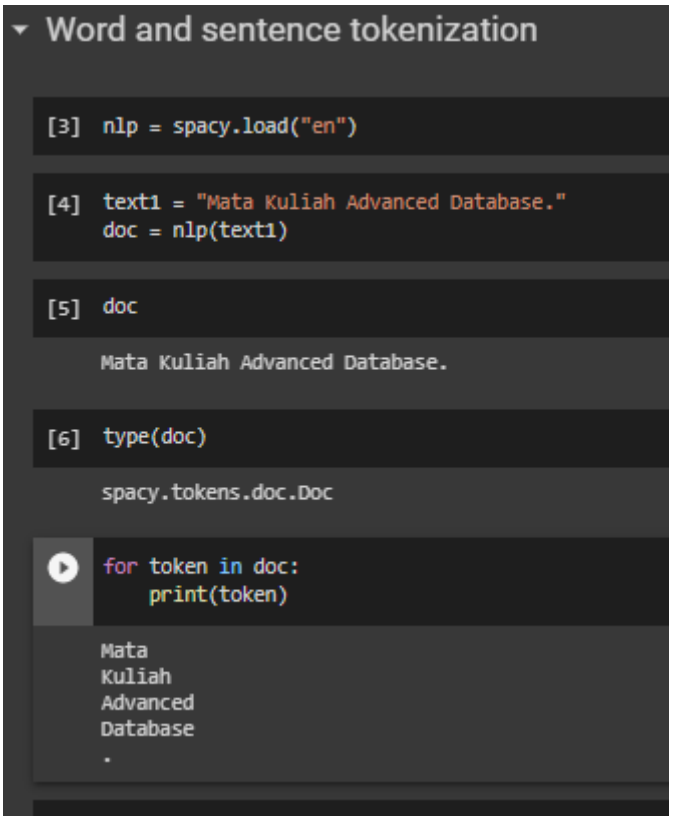
Tahapan Text Mining



Tahapan – tahapan Text Mining

Berikut saya lampirkan salah satu tutorial dari tahapan text minig :

Tokenizing.



https://colab.research.google.com/drive/1RDjz0VksuX2mEMuGUvWMBWf_U8g_xG_?usp=sharing

TUGAS 08

Tutorial Teks Mining

Dibuat Oleh

Robby Prabowo

202420001

MTIA1

Dosen Pengampu

TRI BASUKI KURNIAWAN, S.Kom, M.Eng, Ph.D

Program Pasca Sarjana

Universitas Binadarma Palembang

2020/2021

Nama : Robby Prabowo

NIM : 202420001

MTIA1

Dalam text mining dikenal istilah Text Preprocessing. Text Preprocessing adalah tahapan dimana kita melakukan seleksi data agar data yang akan kita olah menjadi lebih terstruktur. Disini dijelaskan bagiamana melakukan proses Text Preprocessing menggunakan Python dengan Library NLTK

Tidak ada aturan pasti tentang setiap tahapan didalam proses Text Preprocessing semua tergantung dari jenis data (dokumen teks) dan hasil yang diinginkan. Namun pada umumnya tahapan proses text preprocessing adalah Case Folding, Tokenization dan Filtering, Stopword Removal, Stemming.

Case Folding

Dalam sebuah dokumen penggunan huruf kapital atau sejenisnya terkadang tidak mempunyai kesamaan, hal ini bisa dikarenakan kesalahan penulisan. Dalam text preprocessing proses case folding bertujuan untuk mengubah semua huruf dalam sebuah dokumen teks menjadi huruf kecil (lowercase). Untuk proses ini tidak perlu menggunakan library NLTK, kita bisa menggunakan fungsi lower() yang merupakaan bawaah dari python.

```
kalimat = "[MOJOK.co] Manfaat jogging setiap pagi yang pertama adalah meredakan stres. Olahraga itu seperti kode bagi tubuh untuk memproduksi hormon endorfin, agen perangsang rasa bahagia."


case_folding = kalimat.lower()

print(case_folding)
```

Tokenization

Dokumen teks terdiri dari sekumpulan kalimat, proses tokenization memecah dokumen tersebut menjadi bagian-bagian kata yang disebut token. Melakukan tokenization bisa menggunakan library NLTK

```
import nltk
```

kalimat = "[MOJOK.co] Manfaat jogging setiap pagi yang pertama adalah meredakan stres. Olahraga itu seperti kode bagi tubuh untuk memproduksi hormon endorfin, agen perangsang rasa bahagia. Dilakukan di pagi hari, ketika udara masih bersih, sejuk, jalanan lengang, gunung terlihat jelas di sebelah utara, manfaat jogging bisa kamu rasakan secara maksimal."

```
tokens = nltk.tokenize.word_tokenize(kalimat)

print(tokens)


# Hasil

# ['[', 'MOJOK.co', ']', 'Manfaat', 'jogging', 'setiap', 'pagi', 'yang', 'pertama', 'adalah', 'meredakan', 'stres', '.', 'Olahraga', 'itu', 'seperti', 'kode', 'bagi', 'tubuh', 'untuk', 'memproduksi', 'hormon', 'endorfin', ',', 'agen', 'perangsang', 'rasa', 'bahagia', '.', 'Dilakukan', 'di', 'pagi', 'hari', ',', 'ketika', 'udara', 'masih', 'bersih', ',', 'sejuk', ',', 'jalanan', 'lengang', ',', 'gunung', 'terlihat', 'jelas', 'di', 'sebelah', 'utara', ',', 'manfaat', 'jogging', 'bisa', 'kamu', 'rasakan', 'secara', 'maksimal', '.']
```

Setelah melalui proses tokenization kita bisa mendapatkan jumlah kemunculan setiap token nya.

```
import nltk

kalimat = "[MOJOK.co] Manfaat jogging setiap pagi yang pertama adalah meredakan stres. Olahraga itu seperti kode bagi tubuh untuk memproduksi hormon endorfin, agen perangsang rasa bahagia. Dilakukan di pagi hari, ketika udara masih bersih, sejuk, jalanan lengang, gunung terlihat jelas di sebelah utara, manfaat jogging bisa kamu rasakan secara maksimal."

tokens = nltk.tokenize.word_tokenize(kalimat)

kemunculan = nltk.FreqDist(tokens)

print(kemunculan.most_common())


# Hasil

# [(',', 6), ('.', 3), ('jogging', 2), ('pagi', 2), ('di', 2), ('[', 1), ('MOJOK.co', 1), (']', 1), ('Manfaat', 1), ('setiap', 1), ('yang', 1), ('pertama', 1), ('adalah', 1), ('meredakan', 1), ('stres', 1), ('Olahraga', 1), ('itu', 1), ('seperti', 1),
```

('kode', 1), ('bagi', 1), ('tubuh', 1), ('untuk', 1), ('memproduksi', 1), ('hormon', 1), ('endorfin', 1), ('agen', 1), ('perangsang', 1), ('rasa', 1), ('bahagia', 1), ('Dilakukan', 1), ('hari', 1), ('ketika', 1), ('udara', 1), ('masih', 1), ('bersih', 1), ('sejuk', 1), ('jalanan', 1), ('lengang', 1), ('gunung', 1), ('terlihat', 1), ('jelas', 1), ('sebelah', 1), ('utara', 1), ('manfaat', 1), ('bisa', 1), ('kamu', 1), ('rasakan', 1), ('secara', 1), ('maksimal', 1)]

Diatas diketahui jumlah kemunculan kata pada sebuah dokumen. Terdapat 6 kemunculan tanda koma, 3 kemunculan tanda titik, 2 kemunculan kata jogging dst.

Namun kemunculan tanda baca sebaiknya dihindari karena dapat nantinya menggangu proses perhitungan dalam penerapan algoritma Text Mining, kita bisa melakukan filtering terhadap kalimat tersebut untuk menghilangkan tanda baca dan karakter non-alfabet.

```
import nltk

import string


kalimat = "[MOJOK.co] Manfaat jogging setiap pagi yang pertama adalah meredakan stres. Olahraga itu seperti kode bagi tubuh untuk memproduksi hormon endorfin, agen perangsang rasa bahagia. Dilakukan di pagi hari, ketika udara masih bersih, sejuk, jalanan lengang, gunung terlihat jelas di sebelah utara, manfaat jogging bisa kamu rasakan secara maksimal."

kalimat = kalimat.translate(str.maketrans('','',string.punctuation)).lower()


tokens = nltk.tokenize.word_tokenize(kalimat)

kemunculan = nltk.FreqDist(tokens)

print(kemunculan.most_common())


# Hasil

# [('jogging', 2), ('pagi', 2), ('di', 2), ('MOJOKco', 1), ('Manfaat', 1), ('setiap', 1), ('yang', 1), ('pertama', 1), ('adalah', 1), ('meredakan', 1), ('stres', 1), ('Olahraga', 1), ('itu', 1), ('seperti', 1), ('kode', 1), ('bagi', 1), ('tubuh', 1), ('untuk', 1), ('memproduksi', 1), ('hormon', 1), ('endorfin', 1), ('agen', 1), ('perangsang', 1), ('rasa', 1), ('bahagia', 1), ('Dilakukan', 1), ('hari', 1), ('ketika', 1), ('udara', 1), ('masih', 1), ('bersih', 1), ('sejuk', 1), ('jalanan', 1), ('lengang', 1), ('gunung', 1), ('terlihat', 1), ('jelas', 1), ('sebelah', 1), ('utara', 1), ('manfaat', 1), ('bisa', 1), ('kamu', 1), ('rasakan', 1), ('secara', 1), ('maksimal', 1)]
```

Stopword Removal

Tahapan ini akan mengambil kata-kata yang dianggap penting dari hasil tokenization atau membuang kata-kata yang dianggap tidak terlalu mempunyai arti penting dalam proses text mining

```
from nltk.tokenize import sent_tokenize, word_tokenize

from nltk.corpus import stopwords

import string


kalimat = "[MOJOK.co] Manfaat jogging setiap pagi yang pertama adalah meredakan stres. Olahraga itu seperti kode bagi tubuh untuk memproduksi hormon endorfin, agen perangsang rasa bahagia. Dilakukan di pagi hari, ketika udara masih bersih, sejuk, jalanan lengang, gunung terlihat jelas di sebelah utara, manfaat jogging bisa kamu rasakan secara maksimal."

kalimat = kalimat.translate(str.maketrans('','',string.punctuation)).lower()


tokens = word_tokenize(kalimat)

listStopword =  set(stopwords.words('indonesian'))


removed = []

for t in tokens:

   if t not in listStopword:

      removed.append(t)


print(removed)

# Hasil

# ['mojokco', 'manfaat', 'jogging', 'pagi', 'meredakan', 'stres', 'olahraga', 'kode', 'tubuh', 'memproduksi', 'hormon', 'endorfin', 'agen', 'perangsang', 'bahagia', 'pagi', 'udara', 'bersih', 'sejuk', 'jalanan', 'lengang', 'gunung', 'sebelah', 'utara', 'manfaat', 'jogging', 'rasakan', 'maksimal']
```

Stemming

Steeming bertujuan untuk mentransformasikan kata menjadi kata dasarnya (root word) dengan menghilangkan semua imbuhan kata. Sayangnya proses stemming Bahasa Indonesia menggunakan NLTK belum didukung

```
from nltk.stem import PorterStemmer


st = PorterStemmer()
print(st.stem('monitoring'))
# monitor
print(st.stem('stemmed'))
#stem
```

Nama : Shabila Fitri Aulia
Nim : 202420024
Kelas : MTI A 23
MK : Advanced Databased

Teks Mining

Silahkan cari tutorial lain yang membahas tutorial teks mining, lalu buat ringkasannya dan tulis dalam format ms word dan dikumpulkan sebelum batas waktu pengumpulan berakhir.

Jawaban,

Case

Folding

Case Folding adalah tahap untuk konversi text menjadi suatu bentuk yang standar. Pada tahap ini biasanya dipilih lowercase untuk membuat huruf kapital menjadi lowercase.

Pada kesempatan ini saya akan mengambil contoh beberapa kalimat secara acak masing – masing akan saya bedakan apakah itu huruf besar atau kecil, adapun samplenya sebagai berikut :

THE VIRUS THAT CAUSES COVID-19 is mainly TRANSMITTED THROUGH DROPLETS GENERATED WHEN AN INFECTED PERSON COUGHS, SNEEZES, OR EXHALES. These droplETS ARE TOO HEAVY TO
HANG IN THE air, and QUICKLY FALL ON FLOORS OR SURFACES. You can be infected by breathing in the virus if you are Within Close Proximity Of Someone Who Has COVID-19, Or By Touching A Contaminated surface and then your eyes, nose or mouth.

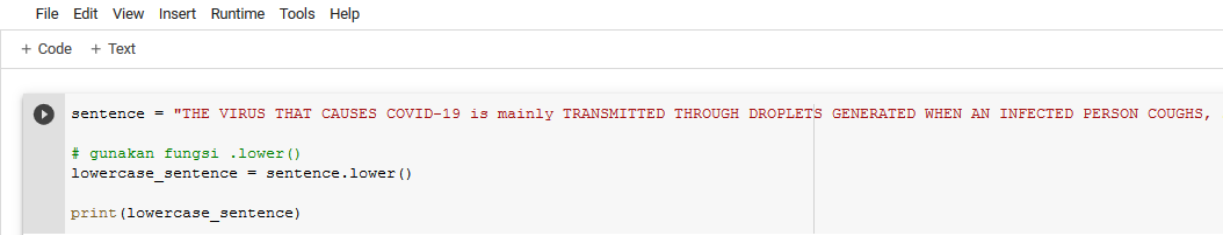
Dari data diatas kita gunakan *colab.research.google.com* untuk memproses sesuai dengan text mining case folding, adapun sintax codingnya sebagai berikut :

sentence = " *THE VIRUS THAT CAUSES COVID-19 is mainly TRANSMITTED THROUGH DROPLETS GENERATED WHEN AN INFECTED PERSON COUGHS, SNEEZES, OR EXHALES. These droplETS ARE TOO HEAVY TO HANG IN THE air, and QUICKLY FALL ON FLOORS OR SURFACES. You can be infected*
by breathing in the virus if you are Within Close Proximity Of Someone Who Has COVID-19, Or By Touching A Contaminated surface and then your eyes, nose or mouth."

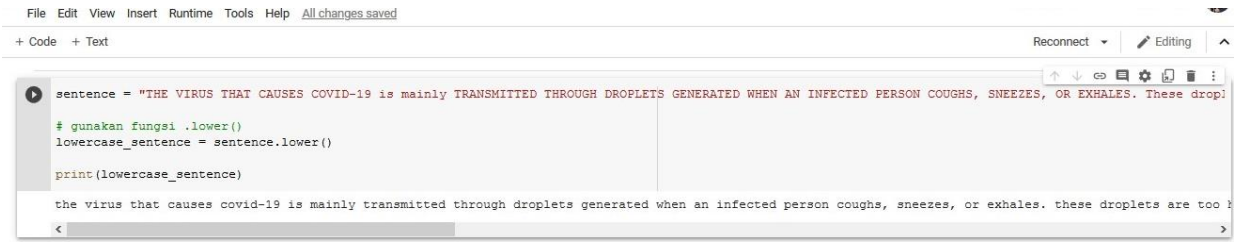
gunakan fungsi .lower()

lowercase_sentence =sentence.lower()

print(lowercase_sentence)



Setelah itu RUN sintax coding diatas dengan menekan tombol RUN  adapun hasilnya sebagai berikut :



Didapatkan semua hasil untuk hurufnya menjadi huruf kecil semua, sehingga secara fungsi sintax codinya berjalan dengan baik.

Nama : Siti Ratu Delima

Nim :202420025

Kelas : MTI 23

Teks Mining

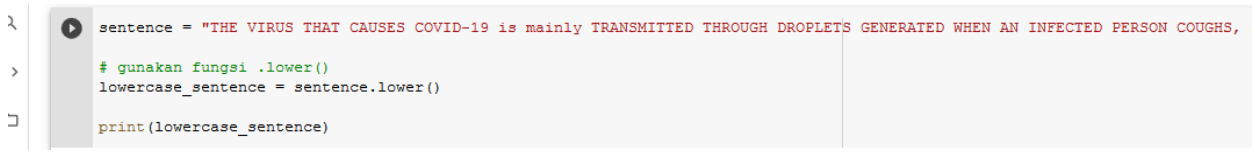
Case Folding adalah tahap untuk konversi text menjadi suatu bentuk yang standar. Pada tahap ini biasanya dipilih lowercase untuk membuat huruf kapital menjadi lowercase.

Pada kesempatan ini saya akan mengambil contoh beberapa kalimat secara acak masing – masing akan saya bedakan apakah itu huruf besar atau kecil, adapun samplenya sebagai berikut :

THE VIRUS THAT CAUSES COVID-19 is mainly TRANSMITTED THROUGH DROPLETS GENERATED WHEN AN INFECTED PERSON COUGHS, SNEEZES, OR EXHALES. These droplETS ARE TOO HEAVY TO
HANG IN THE air, and QUICKLY FALL ON FLOORS OR SURFACES. You can be infected by breathing in the virus if you are Within Close Proximity Of Someone Who Has COVID-19, Or By Touching A Contaminated surface and then your eyes, nose or mouth.

Dari data diatas kita gunakan *colab.research.google.com* untuk memproses sesuai dengan text mining case folding, adapun sintax codingnya sebagai berikut :

```
sentence = " THE VIRUS THAT CAUSES COVID-19 is mainly TRANSMITTED THROUGH DROPLETS GENERATED WHEN AN INFECTED PERSON COUGHS, SNEEZES, OR EXHALES. These droplETS ARE TOO HEAVY TO HANG IN THE air, and QUICKLY FALL ON FLOORS OR SURFACES. You can be infected
by breathing in the virus if you are Within Close Proximity Of Someone Who Has COVID-19, Or By Touching A Contaminated surface and then your eyes, nose or mouth."

# gunakan fungsi .lower()

lowercase_sentence = sentence.lower()

print(lowercase_sentence)
```

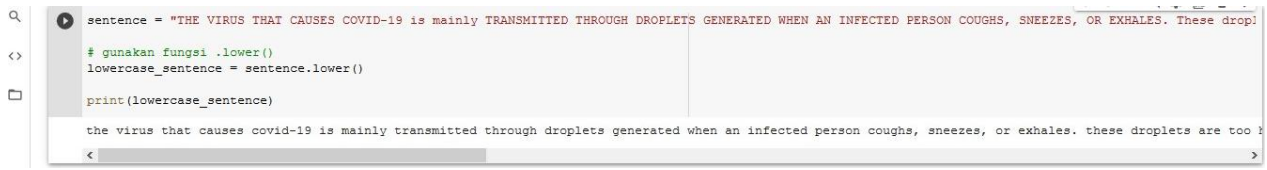


```
sentence = "THE VIRUS THAT CAUSES COVID-19 is mainly TRANSMITTED THROUGH DROPLETS GENERATED WHEN AN INFECTED PERSON COUGHS,

# gunakan fungsi .lower()
lowercase_sentence = sentence.lower()

print(lowercase_sentence)
```

Setelah itu RUN sintax coding diatas dengan menekan tombol RUN

adapun hasilnya sebagai berikut :

Didapatkan semua hasil untuk hurufnya menjadi huruf kecil semua, sehingga secara fungsi sintax



```
sentence = "THE VIRUS THAT CAUSES COVID-19 is mainly TRANSMITTED THROUGH DROPLETS GENERATED WHEN AN INFECTED PERSON COUGHS, SNEEZES, OR EXHALES. These drop

# gunakan fungsi .lower()
lowercase_sentence = sentence.lower()

print(lowercase_sentence)

the virus that causes covid-19 is mainly transmitted through droplets generated when an infected person coughs, sneezes, or exhales. these droplets are too
```

codinya berjalan dengan baik.

TUTORIAL TEXT MINING

NAMA : SURTA WIJAYA
NIM : 202420014

- **Persiapan : Library yang dibutuhkan**

  Salah satu keunggulan python adalah mendukung banyak *open-source library*. Ada banyak *library* python yang dapat digunakan untuk melakukan dan mengimplementasikan masalah dalam NLP.

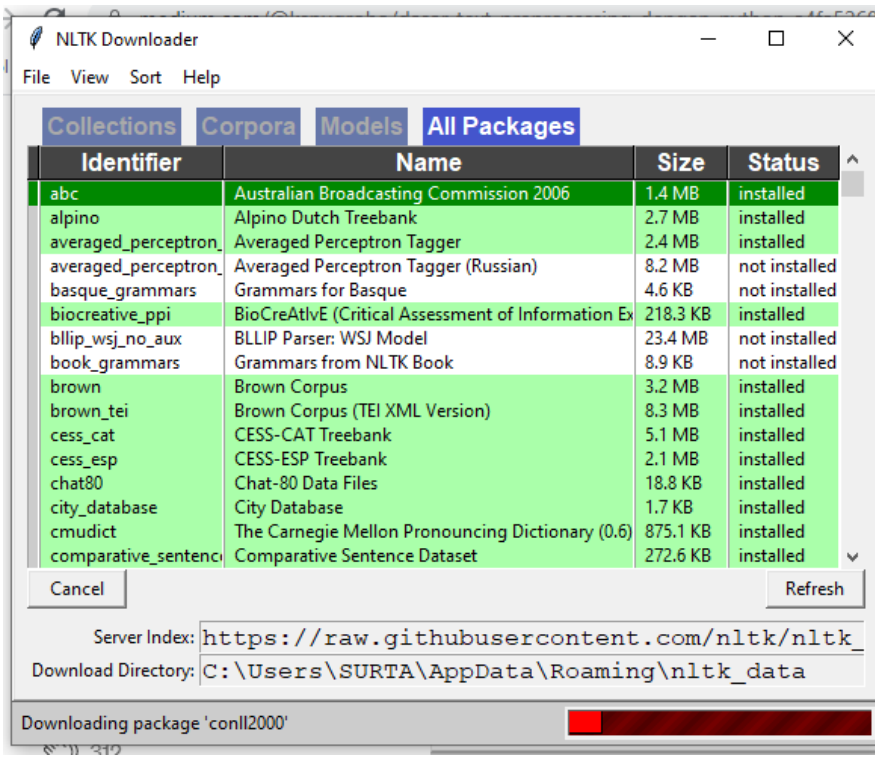
- **Natural Language Toolkit (NLTK)**

  Natural Language Toolkit atau disingkat NLTK, adalah *libray* python untuk bekerja dengan permodelan teks. NLTK menyediakan alat yang baik mempersiapkan teks sebelum digunakan pada *machine learning* atau algoritma *deep learnin*g. Cara termudah untuk menginstall NLTK adalah menggunakan "pip" pada *command line*/*terminal*.

```
pip install nltk
```

Langkah pertama yang perlu anda lakukan setelah menginstall NLTK adalah mengunduh paket NLTK.

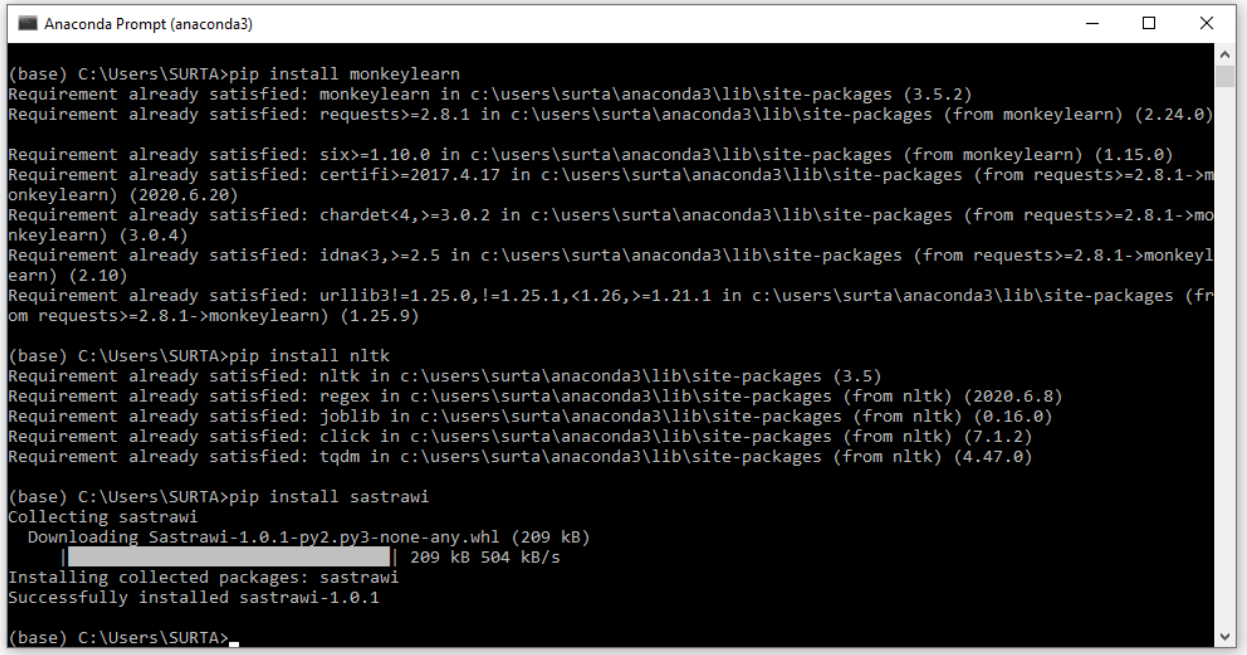
```
Import nltk
nltk.download()
```



- **Python Sastrawi (Stemming Bahasa Indonesia)**

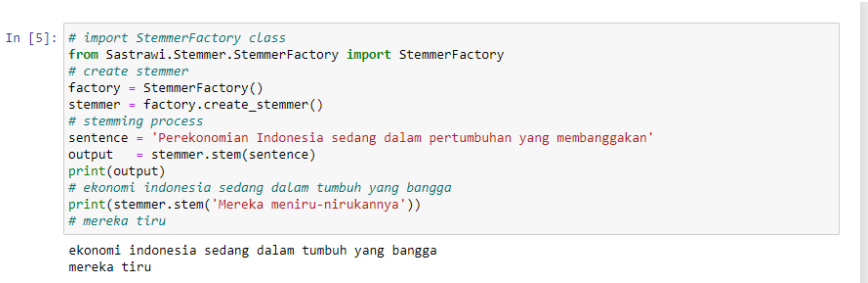
  Python Sastrawi adalah pengembangan dari proyek PHP Sastrawi. Python Sastrawi merupakan library sederhana yang dapat mengubah kata berimbuhan bahasa Indonesia menjadi bentuk dasarnya. Sastrawi juga dapat diinstal melalui “pip”.

```
pip install Sastrawi
```



Penggunaan Python Sastrawi sangat sederhana seperti baris kode dibawah ini :

```
# import StemmerFactory class
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory# create stemmer
factory = StemmerFactory()
stemmer = factory.create_stemmer()# stemming process
sentence = 'Perekonomian Indonesia sedang dalam pertumbuhan yang membanggakan'output   = stemmer.stem(sentence)print(output)
# ekonomi indonesia sedang dalam tumbuh yang banggaprint(stemmer.stem('Mereka meniru-nirukannya'))
# mereka tiru
```



1. Case folding

Case folding adalah salah satu bentuk *text preprocessing* yang paling sederhana dan efektif meskipun sering diabaikan. Tujuan dari *case folding* untuk mengubah semua huruf dalam dokumen menjadi huruf kecil. Hanya huruf 'a' sampai 'z' yang diterima. Karakter selain huruf dihilangkan dan dianggap *delimiter*. Pada tahap ini tidak menggunakan *external library* apapun, kita bisa memanfaatkan modul yang tersedia di python.

Ada beberapa cara yang dapat digunakan dalam tahap *case folding*, anda dapat menggunakan beberapa atau menggunakan semuanya, tergantung pada tugas yang diberikan.

Mengubah text menjadi lowercase

Salah satu contoh pentingnya penggunaan *lower case* adalah untuk mesin pencarian. Bayangkan anda sedang mencari dokumen yang mengandung "indonesia" namun tidak ada hasil yang muncul karena "indonesia" di indeks sebagai "INDONESIA". Siapa yang harus kita salahkan? U.I. *designer* yang mengatur *user interface* atau *developer* yang mengatur sistem dalam indeks pencarian?

Contoh dibawah menunjukan bagaimana python mengubah teks menjadi *lowercase* :

```
kalimat = "Berikut ini adalah 5 negara dengan pendidikan terbaik di dunia adalah Korea Selatan, Jepang, Singapura, Hong Kong, dan Finlandia."lower_case = kalimat.lower()
print(lower_case)# output
# berikut ini adalah 5 negara dengan pendidikan terbaik di dunia adalah korea selatan, jepang, singapura, hong kong, dan finlandia.
```

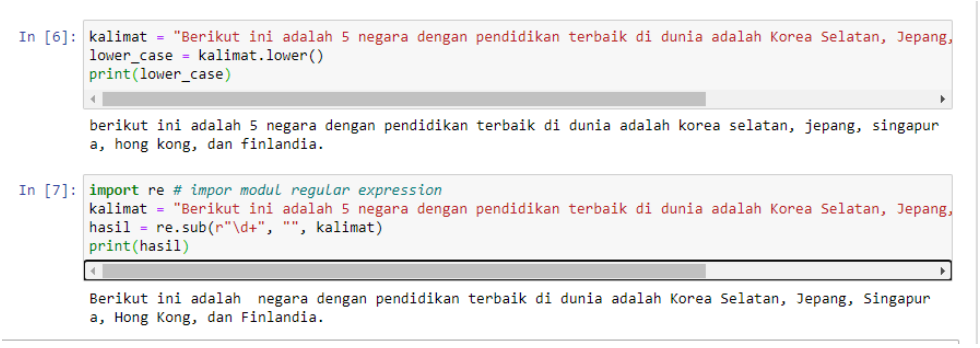
2. Menghapus angka

Hapuslah angka jika tidak relevan dengan apa yang akan anda analisa, contohnya seperti nomor rumah, nomor telepon, dll. *Regular expression* (*regex*) dapat digunakan untuk menghapus karakter angka. Python memiliki modulre untuk melakukan hal – hal yang berkaitan dengan *regex*.

Contoh dibawah menunjukan bagaimana python menghapus angka dalam sebuah kalimat :

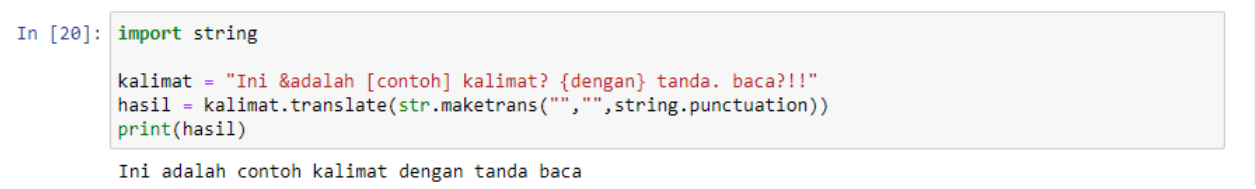
```
import re # impor modul regular expressionkalimat = "Berikut ini adalah 5 negara dengan pendidikan terbaik di dunia adalah Korea Selatan, Jepang, Singapura, Hong Kong, dan Finlandia."
hasil = re.sub(r"\d+", "", kalimat)
print(hasil)# ouput
```

Berikut ini adalah negara dengan pendidikan terbaik di dunia adalah Korea Selatan, Jepang, Singapura, Hong Kong, dan Finlandia.



3. Menghapus tanda baca

Sama halnya dengan angka, tanda baca dalam kalimat tidak memiliki pengaruh pada *text preprocessing*. Menghapus tanda baca seperti [!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~] dapat dilakukan di pyhton seperti dibawah ini :



4. Menghapus whitepace (karakter kosong)

Untuk menghapus spasi di awal dan akhir, anda dapat menggunakan fungsi strip()pada pyhton. Perhatikan kode dibawah ini :

kalimat = " \t ini kalimat contoh\t "

hasil = kalimat.strip()

print(**hasil**)# output

ini kalimat contoh

5. Tokenizing

Tokenizing adalah proses pemisahan teks menjadi potongan-potongan yang disebut sebagai token untuk kemudian di analisa. Kata, angka, simbol, tanda baca dan entitas penting lainnya dapat

dianggap sebagai token. Didalam NLP, token diartikan sebagai “kata” meskipun *tokenize* juga dapat dilakukan pada paragraf maupun kalimat.

Fungsi split()pada pyhton dapat digunakan untuk memisahkan teks. Perhatikan contoh dibawah ini :

```
kalimat = "rumah idaman adalah rumah yang bersih"
pisah = kalimat.split()
print(pisah)# output
# ['rumah', 'idaman', 'adalah', 'rumah', 'yang', 'bersih']
```

Sayangnya, tahap *tokenizing* tidak sesederhana itu. Dari *output* kode diatas kita akan mengolah kata “rumah” dan “rumah” sebagai 2 entitas yang berbeda. Permasalahan ini tentunya akan mempengaruhi hasil dari analisa teks itu sendiri.

Untuk mengatasi masalah ini kita dapat menggunakan modul dari NLTK yang sudah diinstal sebelumnya.

6. Tokenizing kata

Sebuah kalimat atau data dapat dipisah menjadi kata-kata dengan kelas word_tokenize() pada modul NLTK.

```
# impor word_tokenize dari modul nltk
from nltk.tokenize import word_tokenize

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online."

tokens = nltk.tokenize.word_tokenize(kalimat)
print(tokens)# ouput
# ['Andi', 'kerap', 'melakukan', 'transaksi', 'rutin', 'secara', 'daring', 'atau', 'online', '.']
```

Dari *output* kode diatas terdapat kemunculan tanda baca titik(.) dan koma (,) serta token “Andi” yang masih menggunakan huruf besar pada awal kata. Hal tersebut nantinya dapat menggangu proses perhitungan dalam penerapan algoritma. Jadi, sebaiknya teks telah melewati tahap *case folding* sebelum di *tokenize* agar menghasilkan hasil yang lebih konsisten.

Kita dapat menambahkan fungsi *case folding* untuk menghilangkan tanda baca dan mengubah *teks* ke dalam bentuk *lowercase*.

```
kalimat = kalimat.translate(str.maketrans("",string.punctuation)).lower()# output
# ['andi', 'kerap', 'melakukan', 'transaksi', 'rutin', 'secara', 'daring', 'atau', 'online']
```

Kita juga bisa mendapatkan informasi frekuensi kemunculan setiap token dengan kelas FreqDist() yang sudah tersedia pada modul NLTK.

```
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDistkalimat = "Andi kerap melakukan transaksi rutin secara daring atau online. Menurut Andi belanja online lebih praktis & murah."
kalimat = kalimat.translate(str.maketrans("",string.punctuation)).lower()

tokens = nltk.tokenize.word_tokenize(kalimat)
kemunculan = nltk.FreqDist(tokens)
print(kemunculan.most_common())# output
# [('andi', 2), ('online', 2), ('kerap', 1), ('melakukan', 1), ('transaksi', 1), ('rutin', 1), ('secara', 1), ('daring', 1), ('atau', 1), ('menurut', 1), ('belanja', 1), ('lebih', 1), ('praktis', 1), ('murah', 1)]
```

Untuk menggambarkan ke dalam bentuk grafik, kita perlu menginstall library Matplotlib.

```
import matplotlib.pyplot as pltkemunculan.plot(30,cumulative=False)
plt.show()
```

Grafik frekuensi kemunculan kata pada dokumen

7. Tokenizing kalimat

Prinsip yang sama dapat diterapkan untuk memisahkan kalimat pada paragraf. Anda dapat menggunkan kelas sent_tokenize() pada modul NLTK. Saya telah menambahkan kalimat pada contoh seperti dibawah ini :

```
# impor sent_tokenize dari modul nltk
from nltk.tokenize import sent_tokenizekalimat = "Andi kerap melakukan transaksi rutin secara daring atau online. Menurut Andi belanja online lebih praktis & murah."

tokens = nltk.tokenize.sent_tokenize(kalimat)
print(tokens)# ouput
```

```
# ['Andi kerap melakukan transaksi rutin secara daring atau online.', 'Menurut Andi belanja online lebih praktis & murah.']
```

8. Filtering (Stopword Removal)

Filtering adalah tahap mengambil kata-kata penting dari hasil token dengan menggunakan algoritma *stoplist* (membuang kata kurang penting) atau *wordlist* (menyimpan kata penting).
Stopword adalah kata umum yang biasanya muncul dalam jumlah besar dan dianggap tidak memiliki makna. Contoh *stopword* dalam bahasa Indonesia adalah "yang", "dan", "di", "dari", dll. Makna di balik penggunaan *stopword* yaitu dengan menghapus kata-kata yang memiliki informasi rendah dari sebuah teks, kita dapat fokus pada kata-kata penting sebagai gantinya.
Contoh penggunaan *filtering* dapat kita temukan pada konteks mesin pencarian. Jika permintaan pencarian anda adalah "apa itu pengertian manajemen?" tentunya anda ingin sistem pencarian fokus pada memunculkan dokumen dengan topik tentang "pengertian manajemen" di atas dokumen dengan topik "apa itu". Hal ini dapat dilakukan dengan mencegah kata dari daftar *stopword* dianalisa.

9. Filtering dengan NLTK

```
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online. Menurut Andi belanja online lebih praktis & murah."
kalimat = kalimat.translate(str.maketrans("",string.punctuation)).lower()

tokens = word_tokenize(kalimat)
listStopword =  set(stopwords.words('indonesian'))

removed = []
for t in tokens:
    if t not in listStopword:
        removed.append(t)
```

```
print(removed)# ouput
# ['andi', 'kerap', 'transaksi', 'rutin', 'daring', 'online', 'andi', 'belanja', 'online', 'praktis', 'murah']
```

10. Filtering dengan sastrawi

Selain untuk *stemming,* library Sastrawi juga mendukung proses *filtering*. Kita dapat menggunakan stopWordRemoverFactory dari modul sastrawi.

Untuk melihat daftar *stopword* yang telah didefinisikan dalam library Sastrawi dapat menggunakan kode berikut :

```
from Sastrawi.StopWordRemover.StopWordRemoverFactory import
StopWordRemoverFactoryfactory = StopWordRemoverFactory()
stopwords = factory.get_stop_words()
print(stopwords)
```

Kode diatas akan menampikan *stopword* yang tersedia di library Sastrawi. Prosos *filtering* pada Sastrawi dapat dilihat pada baris kode dibawah :

```
from Sastrawi.StopWordRemover.StopWordRemoverFactory import StopWordRemoverFactory
from nltk.tokenize import word_tokenizefactory = StopWordRemoverFactory()
stopword = factory.create_stop_word_remover()kalimat = "Andi kerap melakukan transaksi rutin
secara daring atau online. Menurut Andi belanja online lebih praktis & murah."
kalimat = kalimat.translate(str.maketrans(",",string.punctuation)).lower()stop =
stopword.remove(kalimat)
tokens = nltk.tokenize.word_tokenize(stop)
print(tokens)# output
# ['andi', 'kerap', 'transaksi', 'rutin', 'daring', 'online', 'andi', 'belanja', 'online', 'praktis', 'murah']
```

Kita dapat menambah atau mengurangi kata pada daftar *stopword* sesuai dengan kebutuhan analisa. Pada dasarnya daftar *stopword* pada library Sastrawi tersimpan di dalam list yang anda lihat. Jadi sebenarnya kita tinggal mengubah daftar pada list tersebut. Tetapi hal tersebut bisa menjadi permasalahan apabila pada suatu kasus kita diharuskan menambahkan *stopword* secara dinamis.

Library Sastrawi dapat mengatasi permasalahan tersebut, perhatikan kode dibawah ini :

```
from Sastrawi.StopWordRemover.StopWordRemoverFactory import StopWordRemoverFactory,
StopWordRemover, ArrayDictionary
from nltk.tokenize import word_tokenize

stop_factory = StopWordRemoverFactory().get_stop_words() #load defaul stopword
more_stopword = ['daring', 'online'] #menambahkan stopwordkalimat = "Andi kerap melakukan
transaksi rutin secara daring atau online. Menurut Andi belanja online lebih praktis & murah."
kalimat = kalimat.translate(str.maketrans("",string.punctuation)).lower()data = stop_factory +
more_stopword #menggabungkan stopword

dictionary = ArrayDictionary(data)
str = StopWordRemover(dictionary)
tokens = nltk.tokenize.word_tokenize(str.remove(kalimat))

print(tokens)# output
# ['andi', 'kerap', 'transaksi', 'rutin', 'andi', 'belanja', 'praktis', 'murah']
```

Menurut Jim Geovedi pada artikelnya menjelaskan bahwa penyesuaian daftar *stopwords* perlu dilakukan setiap pertama kali melakukan proyek analisa. Memang bukan sesuatu yang melelahkan, tapi jika tidak dilakukan maka ini akan dapat mengakibatkan salah interpretasi terhadap data.

11. **Stemming**

Stemming adalah proses menghilangkan infleksi kata ke bentuk dasarnya, namun bentuk dasar tersebut tidak berarti sama dengan akar kata (*root word*). Misalnya kata “mendengarkan”, “dengarkan”, “didengarkan” akan ditransformasi menjadi kata “dengar”.

Idenya adalah ketika anda mencari dokumen “cara membuka lemari”, anda juga ingin melihat dokumen yang menyebutkan “cara terbuka lemari” atau “cara dibuka lemari” meskipun terdengar tidak enak. Tentunya anda ingin mencocokan semua variasi kata untuk memunculkan dokumen yang paling relevan.

12. Stemming dengan NLTK (bahasa inggris)

Ada banyak algoritma yang digunakan untuk *stemming*. Salah satu algoritma yang tertua dan paling populer adalah algoritma Porter. Algoritma ini tersedia dalam modul NLTK melalui kelas PorterStemmer().

```
from nltk.stem import PorterStemmer
ps = PorterStemmer()
kata = ["program", "programs", "programer", "programing", "programers"]
for k in kata:
   print(k, " : ", ps.stem(k))# ouput
# program  :  program
programs  :  program
programer  :  program
programing  :  program
programers  :  program
```

Selain Porter, NLTK juga mendukung algoritma Lancester, WordNet Lemmatizer, dan SnowBall. Sayangnya proses *stemming* Bahasa Indonesia pada modul NLTK belum didukung. Stemming bahasa indonesia menggunakan Python Sastrawi

Proses *stemming* antara satu bahasa dengan bahasa yang lain tentu berbeda. Contohnya pada teks berbahasa inggris, proses yang diperlukan hanya proses menghilangkan sufiks. Sedangkan pada teks berbahasa Indonesia semua kata imbuhan baik itu sufiks dan prefiks juga dihilangkan. Untuk melakukan *stemming* bahasa Indonesia kita dapat menggunakan library Python Sastrawi yang sudah kita siapkan di awal. *Library* Sastrawi menerapkan Algoritma Nazief dan Adriani dalam melakukan *stemming* bahasa Indonesia.

```
from Sastrawi.Stemmer.StemmerFactory import StemmerFactoryfactory = StemmerFactory()
stemmer = factory.create_stemmer()

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online. Menurut Andi belanja online lebih praktis & murah."hasil = stemmer.stem(kalimat)print(hasil)# ouput
# andi kerap laku transaksi rutin cara daring atau online turut andi belanja online lebih praktis murah
```

Apakah kita memerlukan semua tahapan pada text preprocessing*?*

Tidak ada aturan pasti yang membahas setiap tahapan pada *text preprocessing*. Tentu saja untuk memastikan hasil yang lebih baik dan konsisten semua tahapan harus dilakukan. Untuk memberi gambaran tentang apa yang minimal seharusnya dilakukan, saya telah menguraikan tahapan menjadi **harus dilakukan**, **sebaiknya dilakukan**, dan **tergantung tugas**. Perlu diingat, *less is more*, anda ingin menjaga pendekatan dengan seindah mungkin. Semakin banyak fitur atau tahapan yang anda tambahkan, semakin banyak pula lapisan yang anda harus kupas.

- **Harus dilakukan** meliputi *case folding* (dapat tergantung tugas dalam beberapa kasus)
- **Sebaiknya dilakukan** meliputi normalisasi sederhana — misalnya menstandarkan kata yang hampir sama
- **Tergantung tugas** meliputi normalisasi tingkat lanjut — misalnya mengatasi kata yang tidak biasa, s*topword removal* dan *stemming*

Jadi, untuk mengatasi tugas apapun minimal anda harus melakukan *case folding*. Selanjutnya dapat ditambahkan beberapa normalisasi dasar untuk mendapatkan hasil yang lebih konsisten dan secara bertahap menambahkan tahapan yang lainnya sesuai yang anda inginkan.

Penutup

Dalam tulisan ini kita telah mengetahui langkah dasar dan praktis pada *text preprocessing* beserta *library* yang digunakan dalam python. Selanjutnya hasil dari *text preprocessing* dapat digunakan untuk analisa NLP yang lebih rumit, contohnya *machine translation*. Tidak semua kasus membutuhkan level *preprocessing* yang sama. Dalam beberapa kasus anda bisa menggunakan salah satu tahap dari *preprocessing* paling sederhana yaitu *case folding*. Namun semua tahap akan dibutuhkan apabila anda mempunyai *dataset* dengan level *noise* sangat tinggi.

SUMBER :

https://medium.com/@ksnugroho/dasar-text-preprocessing-dengan-python-a4fa52608ffe

Nama : Trada Ayang Pratiwi
NIM : 202420020

Dasar Text Preprocessing dengan Python

Pengantar Singkat : Text Preprocessing

Pada *natural language processing* (NLP), informasi yang akan digali berisi data-data yang strukturnya "sembarang" atau tidak terstruktur. Oleh karena itu, diperlukan proses pengubahan bentuk menjadi data yang terstruktur untuk kebutuhan lebih lanjut (*sentiment analysis*, *topic modelling*, dll).

Text data needs to be cleaned and encoded to numerical values before giving them to machine learning models, this process of cleaning and encoding is called as ***text preprocessing.***

Persiapan : Library yang dibutuhkan

Salah satu keunggulan python adalah mendukung banyak *open-source library*. Ada banyak *library* python yang dapat digunakan untuk melakukan dan mengimplementasikan masalah dalam NLP.

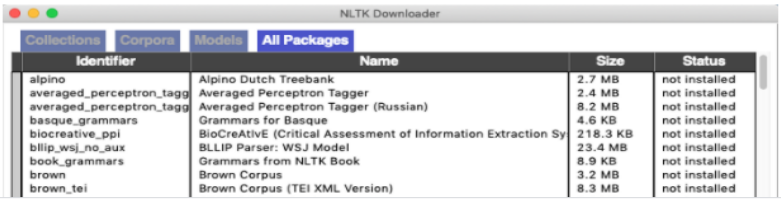
Natural Language Toolkit (NLTK)

Natural Language Toolkit atau disingkat NLTK, adalah *libray* python untuk bekerja dengan permodelan teks. NLTK menyediakan alat yang baik mempersiapkan teks sebelum digunakan pada *machine learning* atau algoritma *deep learnin*g. Cara termudah untuk menginstall NLTK adalah menggunakan "pip" pada *command line*/*terminal*.

```
pip install nltk
```

Langkah pertama yang perlu anda lakukan setelah menginstall NLTK adalah mengunduh paket NLTK.

```
import nltk
nltk.download()
```



Python Sastrawi (Stemming Bahasa Indonesia)

Python Sastrawi adalah pengembangan dari proyek PHP Sastrawi. Python Sastrawi merupakan library sederhana yang dapat mengubah kata berimbuhan bahasa Indonesia menjadi bentuk dasarnya. Sastrawi juga dapat diinstal melalui “pip”.

```
pip install Sastrawi
```

Penggunaan Python Sastrawi sangat sederhana seperti baris kode dibawah ini :

```
# import StemmerFactory class
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory

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

# stemming process
sentence = 'Perekonomian Indonesia sedang dalam pertumbuhan yang
membanggakan'

output   = stemmer.stem(sentence)

print(output)
# ekonomi indonesia sedang dalam tumbuh yang bangga

print(stemmer.stem('Mereka meniru-nirukannya'))
# mereka tiru
```

1. Case folding

Case folding adalah salah satu bentuk *text preprocessing* yang paling sederhana dan efektif meskipun sering diabaikan. Tujuan dari *case folding* untuk mengubah semua huruf dalam dokumen menjadi huruf kecil. Hanya huruf 'a' sampai 'z' yang diterima. Karakter selain huruf dihilangkan dan dianggap *delimiter*. Pada tahap ini tidak menggunakan *external library* apapun, kita bisa memanfaatkan modul yang tersedia di python.

Ada beberapa cara yang dapat digunakan dalam tahap *case folding*, anda dapat menggunakan beberapa atau menggunakan semuanya, tergantung pada tugas yang diberikan.

Mengubah text menjadi lowercase

Salah satu contoh pentingnya penggunaan *lower case* adalah untuk mesin pencarian. Bayangkan anda sedang mencari dokumen yang mengandung "indonesia" namun tidak ada hasil yang muncul karena "indonesia" di indeks sebagai "INDONESIA". Siapa yang harus kita salahkan? U.I. *designer* yang mengatur *user interface* atau *developer* yang mengatur sistem dalam indeks pencarian?

Contoh dibawah menunjukan bagaimana python mengubah teks menjadi *lowercase* :

```
kalimat = "Berikut ini adalah 5 negara dengan pendidikan terbaik di
dunia adalah Korea Selatan, Jepang, Singapura, Hong Kong, dan
Finlandia."

lower_case = kalimat.lower()
print(lower_case)

# output
# berikut ini adalah 5 negara dengan pendidikan terbaik di dunia
adalah korea selatan, jepang, singapura, hong kong, dan finlandia.
```

Menghapus angka

Hapuslah angka jika tidak relevan dengan apa yang akan anda analisa, contohnya seperti nomor rumah, nomor telepon, dll. *Regular expression* (*regex*) dapat digunakan untuk

menghapus karakter angka. Python memiliki modulre untuk melakukan hal – hal yang berkaitan dengan *regex*.

Contoh dibawah menunjukan bagaimana python menghapus angka dalam sebuah kalimat :

```
import re # impor modul regular expression

kalimat = "Berikut ini adalah 5 negara dengan pendidikan terbaik di
dunia adalah Korea Selatan, Jepang, Singapura, Hong Kong, dan
Finlandia."
hasil = re.sub(r"\d+", "", kalimat)
print(hasil)

# ouput
# Berikut ini adalah negara dengan pendidikan terbaik di dunia adalah
Korea Selatan, Jepang, Singapura, Hong Kong, dan Finlandia.
```

Menghapus tanda baca

Sama halnya dengan angka, tanda baca dalam kalimat tidak memiliki pengaruh pada *text preprocessing*. Menghapus tanda baca seperti [!”#$%&’()*+,-./:;<=>?@[\]^_`{|}~] dapat dilakukan di pyhton seperti dibawah ini :

```
kalimat = "Ini &adalah [contoh] kalimat? {dengan} tanda. baca?!!"
hasil = kalimat.translate(str.maketrans("","",string.punctuation))
print(hasil)

# output
# Ini adalah contoh kalimat dengan tanda baca
```

Menghapus whitepace (karakter kosong)

Untuk menghapus spasi di awal dan akhir, anda dapat menggunakan fungsi strip()pada pyhton. Perhatikan kode dibawah ini :

```
kalimat = " \t ini kalimat contoh\t "
hasil = kalimat.strip()
print(hasil)

# output
# ini kalimat contoh
```

2. Tokenizing

Tokenizing adalah proses pemisahan teks menjadi potongan-potongan yang disebut sebagai token untuk kemudian di analisa. Kata, angka, simbol, tanda baca dan entitas penting lainnya dapat dianggap sebagai token. Didalam NLP, token diartikan sebagai "kata" meskipun *tokenize* juga dapat dilakukan pada paragraf maupun kalimat.

Fungsi split()pada pyhton dapat digunakan untuk memisahkan teks. Perhatikan contoh dibawah ini :

```
kalimat = "rumah idaman adalah rumah yang bersih"
pisah = kalimat.split()
print(pisah)


# output
# ['rumah', 'idaman', 'adalah', 'rumah', 'yang', 'bersih']
```

Tokenizing kata

Sebuah kalimat atau data dapat dipisah menjadi kata-kata dengan kelas word_tokenize() pada modul NLTK.

```
# impor word_tokenize dari modul nltk
from nltk.tokenize import word_tokenize

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau
online."

tokens = nltk.tokenize.word_tokenize(kalimat)
print(tokens)


# ouput
# ['Andi', 'kerap', 'melakukan', 'transaksi', 'rutin', 'secara',
'daring', 'atau', 'online', '.']
```

Dari *output* kode diatas terdapat kemunculan tanda baca titik(.) dan koma (,) serta token "Andi" yang masih menggunakan huruf besar pada awal kata. Hal tersebut nantinya dapat menggangu proses perhitungan dalam penerapan algoritma. Jadi, sebaiknya teks telah melewati tahap *case folding* sebelum di *tokenize* agar menghasilkan hasil yang lebih konsisten. Kita dapat menambahkan fungsi *case folding* untuk menghilangkan tanda baca dan mengubah *teks* ke dalam bentuk *lowercase*

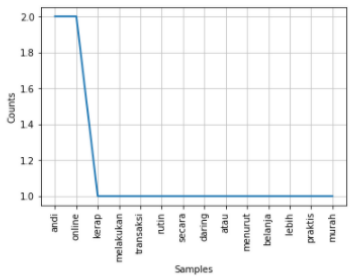
```
kalimat =
kalimat.translate(str.maketrans('','',string.punctuation)).lower()


# output
# ['andi', 'kerap', 'melakukan', 'transaksi', 'rutin', 'secara',
'daring', 'atau', 'online']
```

Kita juga bisa mendapatkan informasi frekuensi kemunculan setiap token dengan kelas FreqDist() yang sudah tersedia pada modul NLTK.

```
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDist


kalimat = "Andi kerap melakukan transaksi rutin secara daring atau
online. Menurut Andi belanja online lebih praktis & murah."
kalimat =
kalimat.translate(str.maketrans('','',string.punctuation)).lower()

tokens = nltk.tokenize.word_tokenize(kalimat)
kemunculan = nltk.FreqDist(tokens)
print(kemunculan.most_common())


# output
# [('andi', 2), ('online', 2), ('kerap', 1), ('melakukan', 1),
('transaksi', 1), ('rutin', 1), ('secara', 1), ('daring', 1),
('atau', 1), ('menurut', 1), ('belanja', 1), ('lebih', 1),
('praktis', 1), ('murah', 1)]
```

Untuk menggambarkan ke dalam bentuk grafik, kita perlu menginstall library Matplotlib.

```
import matplotlib.pyplot as plt


kemunculan.plot(30,cumulative=False)
plt.show()
```



Grafik frekuensi kemunculan kata pada dokumen

3. Filtering (Stopword Removal)

Filtering adalah tahap mengambil kata-kata penting dari hasil token dengan menggunakan algoritma *stoplist* (membuang kata kurang penting) atau *wordlist* (menyimpan kata penting). *Stopword* adalah kata umum yang biasanya muncul dalam jumlah besar dan dianggap tidak memiliki makna. Contoh *stopword* dalam bahasa Indonesia adalah “yang”, “dan”, “di”, “dari”,

dll. Makna di balik penggunaan *stopword* yaitu dengan menghapus kata-kata yang memiliki informasi rendah dari sebuah teks, kita dapat fokus pada kata-kata penting sebagai gantinya.

Contoh penggunaan *filtering* dapat kita temukan pada konteks mesin pencarian. Jika permintaan pencarian anda adalah "apa itu pengertian manajemen?" tentunya anda ingin sistem pencarian fokus pada memunculkan dokumen dengan topik tentang "pengertian manajemen" di atas dokumen dengan topik "apa itu". Hal ini dapat dilakukan dengan mencegah kata dari daftar *stopword* dianalisa.

Filtering dengan NLTK

```
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau
online. Menurut Andi belanja online lebih praktis & murah."
kalimat =
kalimat.translate(str.maketrans('','',string.punctuation)).lower()

tokens = word_tokenize(kalimat)
listStopword =  set(stopwords.words('indonesian'))

removed = []
for t in tokens:
    if t not in listStopword:
        removed.append(t)

print(removed)


# ouput
# ['andi', 'kerap', 'transaksi', 'rutin', 'daring', 'online', 'andi',
'belanja', 'online', 'praktis', 'murah']
```

Filtering dengan sastrawi

Selain untuk *stemming,* library Sastrawi juga mendukung proses *filtering*. Kita dapat menggunakan `stopWordRemoverFactory` dari modul sastrawi.

Untuk melihat daftar *stopword* yang telah didefinisikan dalam library Sastrawi dapat menggunakan kode berikut :

```
from Sastrawi.StopWordRemover.StopWordRemoverFactory import
StopWordRemoverFactory


factory = StopWordRemoverFactory()
stopwords = factory.get_stop_words()
print(stopwords)
```

Kode diatas akan menampikan *stopword* yang tersedia di library Sastrawi.
Prosos *filtering* pada Sastrawi dapat dilihat pada baris kode dibawah :

```
from Sastrawi.StopWordRemover.StopWordRemoverFactory import
StopWordRemoverFactory
from nltk.tokenize import word_tokenize


factory = StopWordRemoverFactory()
stopword = factory.create_stop_word_remover()


kalimat = "Andi kerap melakukan transaksi rutin secara daring atau
online. Menurut Andi belanja online lebih praktis & murah."
kalimat =
kalimat.translate(str.maketrans('','',string.punctuation)).lower()


stop = stopword.remove(kalimat)
tokens = nltk.tokenize.word_tokenize(stop)
print(tokens)


# output
# ['andi', 'kerap', 'transaksi', 'rutin', 'daring', 'online', 'andi',
'belanja', 'online', 'praktis', 'murah']
```

Kita dapat menambah atau mengurangi kata pada daftar *stopword* sesuai dengan kebutuhan analisa. Pada dasarnya daftar *stopword* pada library Sastrawi tersimpan di dalam list yang anda lihat . Jadi sebenarnya kita tinggal mengubah daftar pada list tersebut. Tetapi hal tersebut bisa menjadi permasalahan apabila pada suatu kasus kita diharuskan menambahkan *stopword* secara dinamis.

4. Stemming

Stemming adalah proses menghilangkan infleksi kata ke bentuk dasarnya, namun bentuk dasar tersebut tidak berarti sama dengan akar kata (*root word*). Misalnya kata “mendengarkan”, “dengarkan”, “didengarkan” akan ditransformasi menjadi kata “dengar”.

Idenya adalah ketika anda mencari dokumen “cara membuka lemari”, anda juga ingin melihat dokumen yang menyebutkan “cara terbuka lemari” atau “cara dibuka lemari” meskipun terdengar tidak enak. Tentunya anda ingin mencocokan semua variasi kata untuk memunculkan dokumen yang paling relevan.

Stemming dengan NLTK (bahasa inggris)

Ada banyak algoritma yang digunakan untuk *stemming*. Salah satu algoritma yang tertua dan paling populer adalah algoritma Porter. Algoritma ini tersedia dalam modul NLTK melalui kelasPorterStemmer().

```
from nltk.stem import PorterStemmer

ps = PorterStemmer()

kata = ["program", "programs", "programer", "programing",
"programers"]

for k in kata:
    print(k, " : ", ps.stem(k))


# ouput
# program  :  program
programs  :  program
programer  :  program
programing  :  program
programers  :  program
```

Selain Porter, NLTK juga mendukung algoritma Lancester, WordNet Lemmatizer, dan SnowBall. Sayangnya proses *stemming* Bahasa Indonesia pada modul NLTK belum didukung.

Apakah kita memerlukan semua tahapan pada text preprocessing?

Tidak ada aturan pasti yang membahas setiap tahapan pada *text preprocessing*. Tentu saja untuk memastikan hasil yang lebih baik dan konsisten semua tahapan harus dilakukan. Untuk memberi gambaran tentang apa yang minimal seharusnya dilakukan, saya telah menguraikan tahapan menjadi **harus dilakukan**, **sebaiknya dilakukan**, dan **tergantung tugas**. Perlu diingat, *less is more*, anda ingin menjaga pendekatan dengan seindah mungkin. Semakin banyak fitur atau tahapan yang anda tambahkan, semakin banyak pula lapisan yang anda harus kupas.

- **Harus dilakukan** meliputi *case folding* (dapat tergantung tugas dalam beberapa kasus)
- **Sebaiknya dilakukan** meliputi normalisasi sederhana — misalnya menstandarkan kata yang hampir sama
- **Tergantung tugas** meliputi normalisasi tingkat lanjut — misalnya mengatasi kata yang tidak biasa, s*topword removal* dan *stemming*

Jadi, untuk mengatasi tugas apapun minimal anda harus melakukan *case folding*. Selanjutnya dapat ditambahkan beberapa normalisasi dasar untuk mendapatkan hasil yang lebih konsisten dan secara bertahap menambahkan tahapan yang lainnya sesuai yang anda inginkan.

Penutup

Dalam tulisan ini kita telah mengetahui langkah dasar dan praktis pada *text preprocessing* beserta *library* yang digunakan dalam python. Selanjutnya hasil dari *text preprocessing* dapat digunakan untuk analisa NLP yang lebih rumit, contohnya *machine translation*. Tidak semua kasus membutuhkan level *preprocessing* yang sama. Dalam beberapa kasus anda bisa menggunakan salah satu tahap dari *preprocessing* paling sederhana yaitu *case folding*. Namun semua tahap akan dibutuhkan apabila anda mempunyai *dataset* dengan level *noise* sangat tinggi.

Masih ada teknik yang bisa dilakukan pada *text preprocessing*, tetapi sesuai kata pengantar diatas, tulisan ini hanya mengulas langkah dasar dan praktis dalam *text preprocessing*.

Selamat mencoba dan bersenang-senang dengan NLP :)

Referensi

1. Adriani, M., Asian, J., Nazief, B., Tahaghoghi, S. M. M., & Williams, H. E. (2007). Stemming Indonesian. ACM Transactions on Asian Language Information Processing, 6(4), 1–33.
2. Tala, Fadillah Z.(1999). *A Study of Stemming Effect on Information Retrieval in* Bahasa Indonesia.
3. Geovedi, Jim.(2014).Karena Data Gak Mungkin Bohong dan karena Bisa Diolah Sesuai Pesanan (https://medium.com/curahan-rekanalar/karena-data-gak-mungkin-bohong-a17ff90cef87).
4. Python Sastrawi (https://github.com/har07/PySastrawi).
5. Natural Language Toolkit -NLTK (https://www.nltk.org).
6. Matplotlib (https://matplotlib.org).

TUGAS 8

Nama : Vero Faloris

Nim : 202420032

Kelas : MTI 23 Reguler A

Mk : Advanced Database

Silahkan cari tutorial lain yang membahas tutorial teks mining, lalu buat ringkasannya dan tulis dalam format ms word dan dikumpulkan sebelum batas waktu pengumpulan berakhir.

Text mining memiliki definisi menambang data yang berupa teks dimana sumber data biasanya di dapatkan dari dokumen, dan tujuannya adalah mencari kata-kata yang dapat mewakili isi dari dokumen sehingga dapat dilakukan analisa keterhubungan antardokumen. tekt mining juga di definisikan sebagai penerapan konsep dan teknik data mining untuk mencari pola dalam teks, yaitu proses penganalisisan teks guna menyarikan informasi yang bermanfaat untuk tujuan tertentu.

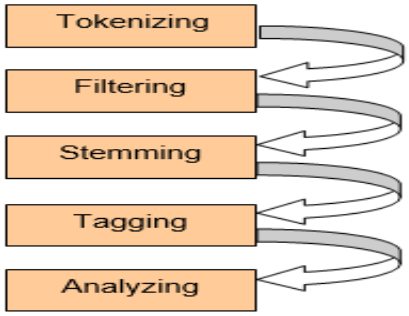
kegunaan dari text mining adalah untuk mendapatkan informasi yang berguna dari sekumpulan dokumen. Jadi, sumber data yang digunakan pada text mining adalah kumpulan teks yang memiliki format yang tidak terstruktur atau minimal semi terstruktur. Adapun tugas khusus dari text mining antara lain yaitu pengkategorisasian teks (text categorization) dan pengelompokan teks text clustering).

Rujukan : http://tessy.lecturer.pens.ac.id/kuliah/dm/6Text%20Mining.pdf

Text mining memiliki definisi menambang data yang berupa teks dimana sumber data biasanya didapatkan dari dokumen, dan tujuannya adalah mencari kata-kata yang dapat mewakili isi dari dokumen sehingga dapat dilakukan analisa keterhubungan antar dokumen

Tahapan dalam Text Mining

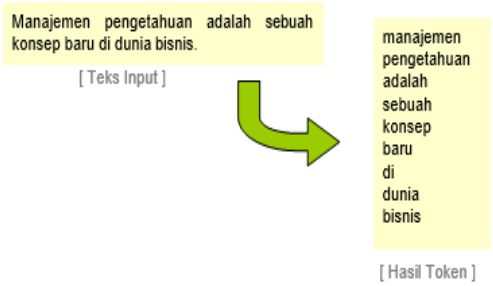
• Tahapan yang dilakukan secara umum adalah:



Tokenizing

•Tahap tokenizing adalah tahap pemotongan string input berdasarkan tiap kata yang menyusunnya.

•Contoh dari tahap ini adalah sebagai berikut:

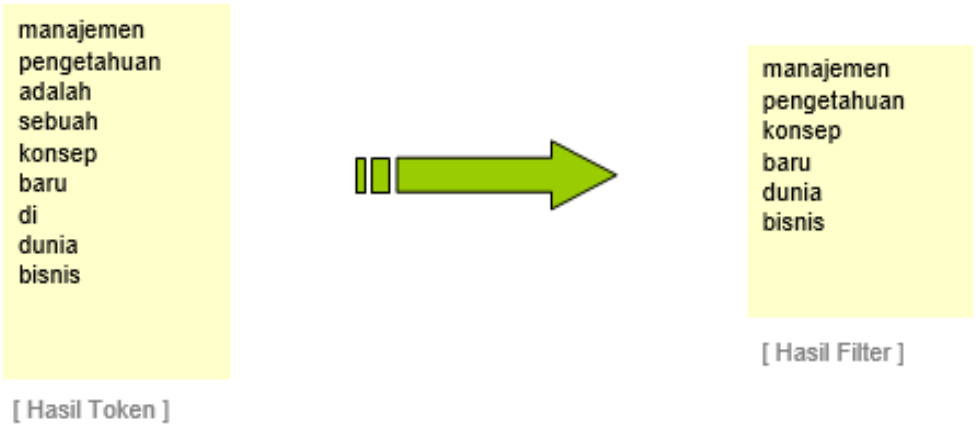


Filtering

•Tahap filtering adalah tahap mengambil kata-kata penting dari hasil token.

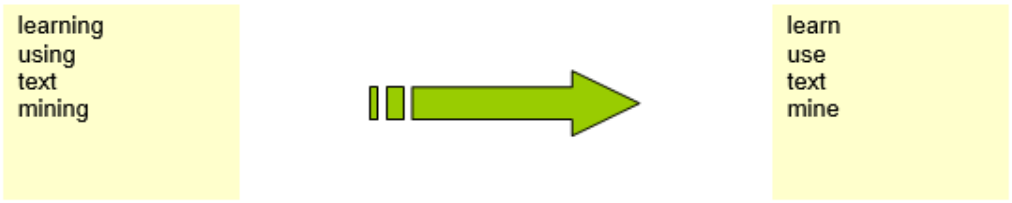
•Bisa menggunakan algoritma stop list (membuang kata yang kurang penting) atau word list (menyimpan kata penting).

•Contoh dari tahap ini adalah sebagai berikut:



Stemming

- Tahap stemming adalah tahap mencari root kata dari tiap kata hasil filtering.
- Contoh dari tahap ini adalah sebagai berikut:



[Hasil Filter] [Hasil Stemming]

Tagging

- Tahap tagging adalah tahap mencari bentuk awal/root dari tiap kata lampau atau kata hasil stemming.
- Contoh dari tahap ini adalah sebagai berikut:

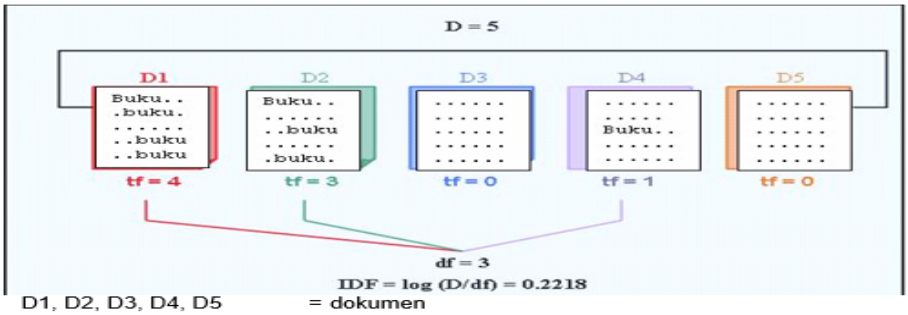
[Hasil Stemming]

[Hasil tangging]

Analyzing

- Tahap Analyzing merupakan tahap penentuan seberapa jauh keterhubungan antar kata-kata antar dokumen yang ada.

Ilustrasi Algoritma Text Mining



D1, D2, D3, D4, D5 = dokumen
tf = banyak kata yang dicari pada sebuah dokumen
D = total dokumen
df = banyak dokumen yang mengandung kata yang dicari

Nama : Wahyu Putra Adi Wibowo

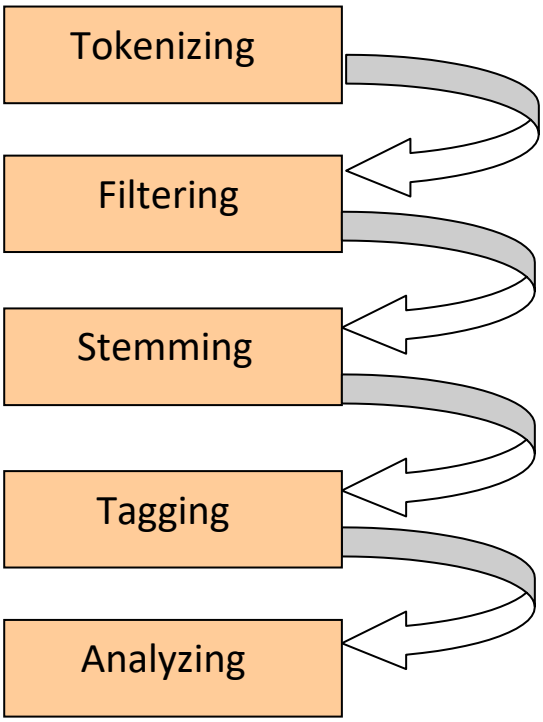
Nim : 202420041

Text Mining

Text mining memiliki definisi menambang data yang berupa teks dimana sumber data biasanya didapatkan dari dokumen, dan tujuannya adalah mencari kata-kata yang dapat mewakili isi dari dokumen sehingga dapat dilakukan analisa keterhubungan antar dokumen.

Tahapan dalam Text Mining

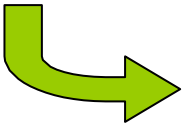
Tahapan yang dilakukan secara umum adalah:



Tokenizing

- Tahap tokenizing adalah tahap pemotongan string input berdasarkan tiap kata yang menyusunnya.
- Contoh dari tahap ini adalah sebagai berikut:

Manajemen pengetahuan adalah sebuah konsep baru di dunia bisnis.



manajemen pengetahuan adalah sebuah konsep baru di dunia bisnis

Filtering

- Tahap filtering adalah tahap mengambil kata-kata penting dari hasil token.
- Bisa menggunakan algoritma stop list (membuang kata yang kurang penting) atau word list (menyimpan kata penting).
- Contoh dari tahap ini adalah sebagai berikut

Manajemen pengetahuan
adalah sebuah konsep
baru di dunia bisnis

manajemen pengetahuan konsep baru
dunia bisnis

Stemming

- Tahap stemming adalah tahap mencari root kata dari tiap kata hasil filtering.
- Contoh dari tahap ini adalah sebagai berikut:

learning using text mining

learn use text mine

Tagging

- Tahap tagging adalah tahap mencari bentuk awal/root dari tiap kata lampau atau kata hasil stemming.
- Contoh dari tahap ini adalah sebagai berikut:

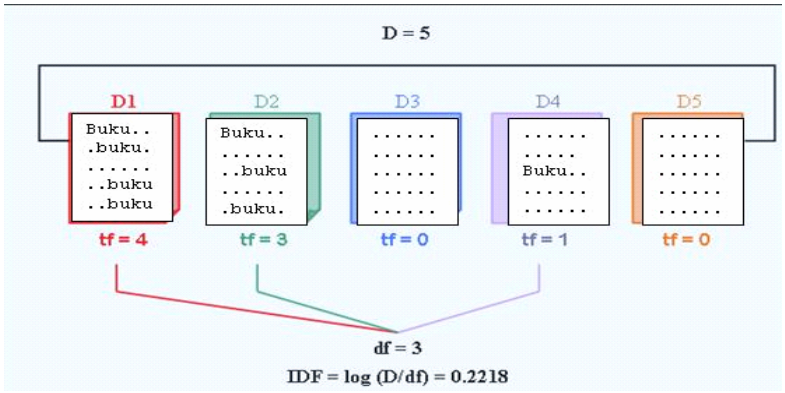
was used stori

be use story

Analyzing

Tahap Analyzing merupakan tahap penentuan seberapa jauh keterhubungan antar kata-kata antar dokumen yang ada.

Ilustrasi Algoritma Text Mining



D1, D2, D3, D4, D5 = dokumen
tf = banyak kata yang dicari pada sebuah dokumen
D = total dokumen
df = banyak dokumen yang mengandung kata yang dicari

Algoritma TF/IDF

- Formula yang digunakan untuk menghitung bobot (w) masing-masing dokumen terhadap kata kunci adalah
- Wd,t = tfd,t * IDFt
- Dimana:
- d = dokumen ke-d
- t = kata ke-t dari kata kunci
- W = bobot dokumen ke-d terhadap kata ke-t
- Setelah bobot (w) masing-masing dokumen diketahui, maka dilakukan proses sorting/pengurutan dimana semakin besar nilai w, semakin besar tingkat similaritas dokumen tersebut terhadap kata yang dicari, demikian sebaliknya

Ilustrasi TF/IDF

Kata kunci (kk) = pengetahuan logistik Dokumen 1 (D1) = Manajemen transaksi logistik Dokumen 2 (D2) = **Pengetahuan antar individu**
Dokumen 3 (D3) = **Dalam manajemen pengetahuan terdapat**
transfer pengetahuan logistik Jadi jumlah dokumen (D) = 3

€ Setelah melalui proses filtering, maka kata antar pada dokumen 2 serta kata dalam dan terdapat pada dokumen 3 dihapus **tabel perhitungan TF-IDF**

token	tf				df	D/df	IDF log(D/df)	W			
	kk	D1	D2	D3				kk	D1	D2	D3
manajemen	0	1	0	1	2	1.5	0.176	0	0.176	0	0.176
transaksi	0	1	0	0	1	3	0.477	0	0.477	0	0
logistik	1	1	0	1	2	1.5	0.176	0.176	0.176	0	0.176
transfer	0	0	0	1	1	3	0.477	0	0	0	0.477
pengetahuan	1	0	1	2	2	1.5	0.176	0.176	0	0.176	0.352
individu	0	0	1	0	1	3	0.477	0	0	0.477	0
Total	0.352	0.829	0.653	1.181							

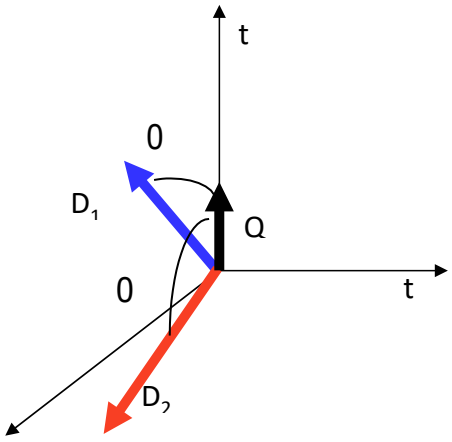
bobot (w) untuk D1 = 0.176 + 0 = 0.176 bobot (w) untuk D2 = 0 + 0.176 = 0.176 bobot (w) untuk D3 = 0.176 + 0.352 = 0.528

Analisa TFIIDF

- Dari contoh studi kasus di atas, dapat diketahui bahwa nilai bobot (w) dari D1 dan D2 adalah sama.
- Apabila diurutkan maka proses sorting juga tidak akan dapat mengurutkan secara tepat, karena nilai w keduanya sama.
- Untuk mengatasi hal ini, digunakan algoritma dari vector-space model

Vector Space Model

Ide dari metode ini adalah dengan menghitung nilai cosinus sudut dari dua vektor, yaitu W dari tiap dokumen dan W dari kata kunci



t = kata di database
D = dokumen
Q = kata kunci

Tabel perhitungan Vector-Space Model

token	kk	D1	D 2	D3	Kk*D1	Kk*D2	kk*D3
manajemen	0	0.031	0	0.031	0	0	0
transaksi	0	0.228	0	0	0	0	0
logistik	0.031	0.031	0	0.031	0.031	0	0.031
transfer	0	0	0	0.228	0	0	0
pengetahuan	0.031	0	0.031	0.124	0	0.031	0.062
individu	0	0	0.228	0	0	0	0
	Sqrt(kk)	Sqrt(Di)			Sum(kk dot Di)		
	0.249	0.539	0.509	0.643	0.031	0.031	0.093

Analisa Vector Space Model

- Demikian juga untuk Cosine dari D1 dan D2. Sehingga hasil yang diperoleh untuk ketiga dokumen di atas adalah seperti berikut ini.

- Tabel 2.3 Tabel hasil Vector-Space Model

	D1	D2	D3
Cosine	0.231	0.245	0.581
	Rank 3	**Rank 2**	**Rank 1**

Dari hasil akhir (Cosine) maka dapat :

Diketahui bahwa document 3 (D3) memiliki tingkat similaritas tertinggi kemudian disusul dengan D2 lalu D1

Hasil Reverensi

http://tessy.lecturer.pens.ac.id/kuliah/dm/6Text%20Mining.pdf

http://222.124.22.22/budsus/pdf/textwebmining/TextMining_Kuliah.pdf

NAMA **: WIDIA ASTUTI**

NIM **: 202420021**

MATA KULIAH **: ADVANCED DATABASE**

TUGAS 8

Silahkan cari tutorial lain yang membahas tutorial teks mining, lalu buat ringkasannya dan tulis dalam format ms word dan dikumpulkan sebelum batas waktu pengumpulan berakhir.

Jawab :

Salah satu aplikasi pemograman teks mining yaitu Borland C++

A. Langkah-lagkah :

1. **Install Borland C++**
2. **Memulai Borland C++ 5.02** → Jalankan Borland C++ 5.02→Buat project baru:
   - Pilih menu: File-New-Project → dialog New Target pada Project Path and Name (isikan nama proyeknya lengkap, ex: c:\saya\proyek1.ide
   - Pada Target Type, pilih: Application [.exe]
   - Pada Platform, pilih: DOS (Standard)
     (boleh pula memilih Target Type EasyWin [.exe] dengan Platform Windows 3.x (16))
   - Pada Target Model, pilih: Large
     hilangkan tanda centang pada Frameworks – Class Library →OK
3. Muncul proyek baru dengan target proyek1.exe dan file proyek1.cpp
   - Klik dobel pada proyek1.cpp untuk mengeditnya. Siap untuk menuliskan programnya
4. Untuk menjalankan program (pilih salah satu):
   1. pilih menu: Debug-Run
   2. tekan tombol Ctrl-F9
   3. klik tombol di toolbar yang bergambar kilat kuning

B. Latihan pemograman, Input / Output dan Jenis Data

1. **Output menggunakan stream: Hello World! dalam C++**
   #include <iostream.h>
   main()
   {cout << “Hello world!\nWe’re in C++ now”;}
   Jalankan, dan selamat! Anda telah berhasil membuat program pertama dalam bahasa C++.
   Gantilah isi programnya menjadi:
   cout << “Hello world!” << endl
   << “We’re in C++ now”;
   Hasilnya sama dengan sebelumnya. Apakah fungsi dari endl?
2. **Input string menggunakan stream: program nama**
   #include <iostream.h>
   main()
   {

3. char nama[80];
   cout << “Masukkan nama: “;
   cin >> nama;
   cout << “Halo ” << nama << endl
   << “Betul kan, kamu si ” << nama;
   }

Menggantikan fungsi apakah cin dan cout

C. **Program input ke variabel bilangan: menghitung akar**

```
#include <iostream.h>
#include <math.h>
main()
{
int a;
float b;
cout << “Masukkan nilai a = “;
cin >> a;
b = sqrt(a);
cout << “akar dari a = ” << b;
}
```

TUGAS 08
TEXT MINNING

Dibuat Oleh
Aan Novrianto

Dosen Pengampu
TRI BASUKI KURNIAWAN, S.Kom, M.Eng, Ph.D

Program Pasca Sarjana
Universitas Binadarma Palembang
2020/2021

Dasar Text Preprocessing dengan Python

Pengantar Singkat : Text Preprocessing

Pada *natural language processing* (NLP), informasi yang akan digali berisi data-data yang strukturnya "sembarang" atau tidak terstruktur. Oleh karena itu, diperlukan proses pengubahan bentuk menjadi data yang terstruktur untuk kebutuhan lebih lanjut (*sentiment analysis*, *topic modelling*, dll).

Text data needs to be cleaned and encoded to numerical values before giving them to machine learning models, this process of cleaning and encoding is called as ***text preprocessing.***

Persiapan : Library yang dibutuhkan

Salah satu keunggulan python adalah mendukung banyak *open-source library*. Ada banyak *library* python yang dapat digunakan untuk melakukan dan mengimplementasikan masalah dalam NLP.

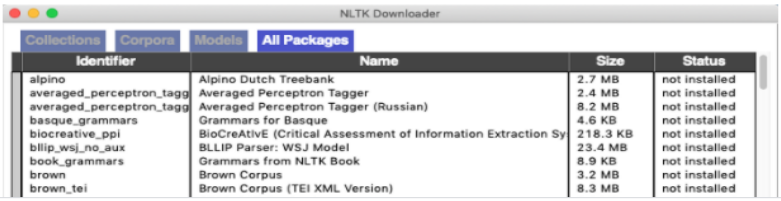
Natural Language Toolkit (NLTK)

Natural Language Toolkit atau disingkat NLTK, adalah *libray* python untuk bekerja dengan permodelan teks. NLTK menyediakan alat yang baik mempersiapkan teks sebelum digunakan pada *machine learning* atau algoritma *deep learning*. Cara termudah untuk menginstall NLTK adalah menggunakan "pip" pada *command line*/*terminal*.

```
pip install nltk
```

Langkah pertama yang perlu anda lakukan setelah menginstall NLTK adalah mengunduh paket NLTK.

```
import nltk
nltk.download()
```



Python Sastrawi (Stemming Bahasa Indonesia)

Python Sastrawi adalah pengembangan dari proyek PHP Sastrawi. Python Sastrawi merupakan library sederhana yang dapat mengubah kata berimbuhan bahasa Indonesia menjadi bentuk dasarnya. Sastrawi juga dapat diinstal melalui “pip”.

```
pip install Sastrawi
```

Penggunaan Python Sastrawi sangat sederhana seperti baris kode dibawah ini :

```
# import StemmerFactory class
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory

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

# stemming process
sentence = 'Perekonomian Indonesia sedang dalam pertumbuhan yang
membanggakan'

output   = stemmer.stem(sentence)

print(output)
# ekonomi indonesia sedang dalam tumbuh yang bangga

print(stemmer.stem('Mereka meniru-nirukannya'))
# mereka tiru
```

1. Case folding

Case folding adalah salah satu bentuk *text preprocessing* yang paling sederhana dan efektif meskipun sering diabaikan. Tujuan dari *case folding* untuk mengubah semua huruf dalam dokumen menjadi huruf kecil. Hanya huruf 'a' sampai 'z' yang diterima. Karakter selain huruf dihilangkan dan dianggap *delimiter*. Pada tahap ini tidak menggunakan *external library* apapun, kita bisa memanfaatkan modul yang tersedia di python.

Ada beberapa cara yang dapat digunakan dalam tahap *case folding*, anda dapat menggunakan beberapa atau menggunakan semuanya, tergantung pada tugas yang diberikan.

Mengubah text menjadi lowercase

Salah satu contoh pentingnya penggunaan *lower case* adalah untuk mesin pencarian. Bayangkan anda sedang mencari dokumen yang mengandung "indonesia" namun tidak ada hasil yang muncul karena "indonesia" di indeks sebagai "INDONESIA". Siapa yang harus kita salahkan? U.I. *designer* yang mengatur *user interface* atau *developer* yang mengatur sistem dalam indeks pencarian?

Contoh dibawah menunjukan bagaimana python mengubah teks menjadi *lowercase* :

```
kalimat = "Berikut ini adalah 5 negara dengan pendidikan terbaik di
dunia adalah Korea Selatan, Jepang, Singapura, Hong Kong, dan
Finlandia."

lower_case = kalimat.lower()
print(lower_case)

# output
# berikut ini adalah 5 negara dengan pendidikan terbaik di dunia
adalah korea selatan, jepang, singapura, hong kong, dan finlandia.
```

Menghapus angka

Hapuslah angka jika tidak relevan dengan apa yang akan anda analisa, contohnya seperti nomor rumah, nomor telepon, dll. *Regular expression* (*regex*) dapat digunakan untuk menghapus karakter angka. Python memiliki modul `re` untuk melakukan hal – hal yang berkaitan dengan *regex*.

Contoh dibawah menunjukan bagaimana python menghapus angka dalam sebuah kalimat :

```
import re # impor modul regular expression

kalimat = "Berikut ini adalah 5 negara dengan pendidikan terbaik di
dunia adalah Korea Selatan, Jepang, Singapura, Hong Kong, dan
Finlandia."
hasil = re.sub(r"\d+", "", kalimat)
print(hasil)

# ouput
# Berikut ini adalah negara dengan pendidikan terbaik di dunia adalah
Korea Selatan, Jepang, Singapura, Hong Kong, dan Finlandia.
```

Menghapus tanda baca

Sama halnya dengan angka, tanda baca dalam kalimat tidak memiliki pengaruh pada *text preprocessing*. Menghapus tanda baca seperti [!”#$%&’()*+,-./:;<=>?@[\]^_`{|}~] dapat dilakukan di pyhton seperti dibawah ini :

```
kalimat = "Ini &adalah [contoh] kalimat? {dengan} tanda. baca?!!"
hasil = kalimat.translate(str.maketrans("","",string.punctuation))
print(hasil)

# output
# Ini adalah contoh kalimat dengan tanda baca
```

Menghapus whitepace (karakter kosong)

Untuk menghapus spasi di awal dan akhir, anda dapat menggunakan fungsi `strip()` pada pyhton. Perhatikan kode dibawah ini :

```
kalimat = " \t ini kalimat contoh\t "
hasil = kalimat.strip()
print(hasil)

# output
# ini kalimat contoh
```

2. Tokenizing

Tokenizing adalah proses pemisahan teks menjadi potongan-potongan yang disebut sebagai token untuk kemudian di analisa. Kata, angka, simbol, tanda baca dan entitas penting lainnya dapat dianggap sebagai token. Didalam NLP, token diartikan sebagai "kata" meskipun *tokenize* juga dapat dilakukan pada paragraf maupun kalimat.

Fungsi `split()` pada pyhton dapat digunakan untuk memisahkan teks. Perhatikan contoh dibawah ini :

```
kalimat = "rumah idaman adalah rumah yang bersih"
pisah = kalimat.split()
print(pisah)

# output
# ['rumah', 'idaman', 'adalah', 'rumah', 'yang', 'bersih']
```

Tokenizing kata

Sebuah kalimat atau data dapat dipisah menjadi kata-kata dengan kelas `word_tokenize()` pada modul NLTK.

```
# impor word_tokenize dari modul nltk
from nltk.tokenize import word_tokenize

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau
online."

tokens = nltk.tokenize.word_tokenize(kalimat)
print(tokens)

# ouput
# ['Andi', 'kerap', 'melakukan', 'transaksi', 'rutin', 'secara',
'daring', 'atau', 'online', '.']
```

Dari *output* kode diatas terdapat kemunculan tanda baca titik(.) dan koma (,) serta token "Andi" yang masih menggunakan huruf besar pada awal kata. Hal tersebut nantinya dapat menggangu proses perhitungan dalam penerapan algoritma. Jadi, sebaiknya teks telah melewati tahap *case folding* sebelum di *tokenize* agar menghasilkan hasil yang lebih konsisten.

Kita dapat menambahkan fungsi *case folding* untuk menghilangkan tanda baca dan mengubah *teks* ke dalam bentuk *lowercase*

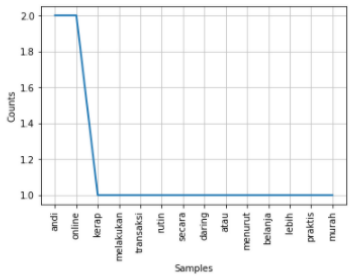
```
kalimat =
kalimat.translate(str.maketrans('','',string.punctuation)).lower()


# output
# ['andi', 'kerap', 'melakukan', 'transaksi', 'rutin', 'secara',
'daring', 'atau', 'online']
```

Kita juga bisa mendapatkan informasi frekuensi kemunculan setiap token dengan kelas `FreqDist()` yang sudah tersedia pada modul NLTK.

```
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDist


kalimat = "Andi kerap melakukan transaksi rutin secara daring atau
online. Menurut Andi belanja online lebih praktis & murah."
kalimat =
kalimat.translate(str.maketrans('','',string.punctuation)).lower()

tokens = nltk.tokenize.word_tokenize(kalimat)
kemunculan = nltk.FreqDist(tokens)
print(kemunculan.most_common())


# output
# [('andi', 2), ('online', 2), ('kerap', 1), ('melakukan', 1),
('transaksi', 1), ('rutin', 1), ('secara', 1), ('daring', 1),
('atau', 1), ('menurut', 1), ('belanja', 1), ('lebih', 1),
('praktis', 1), ('murah', 1)]
```

Untuk menggambarkan ke dalam bentuk grafik, kita perlu menginstall library Matplotlib.

```
import matplotlib.pyplot as plt


kemunculan.plot(30,cumulative=False)
plt.show()
```



Grafik frekuensi kemunculan kata pada dokumen

3. Filtering (Stopword Removal)

Filtering adalah tahap mengambil kata-kata penting dari hasil token dengan menggunakan algoritma *stoplist* (membuang kata kurang penting) atau *wordlist* (menyimpan kata penting). *Stopword* adalah kata umum yang biasanya muncul dalam jumlah besar dan dianggap tidak memiliki makna. Contoh *stopword* dalam bahasa Indonesia adalah “yang”, “dan”, “di”, “dari”, dll. Makna di balik penggunaan *stopword* yaitu dengan menghapus kata-kata yang memiliki informasi rendah dari sebuah teks, kita dapat fokus pada kata-kata penting sebagai gantinya.

Contoh penggunaan *filtering* dapat kita temukan pada konteks mesin pencarian. Jika permintaan pencarian anda adalah “apa itu pengertian manajemen?” tentunya anda ingin sistem pencarian fokus pada memunculkan dokumen dengan topik tentang “pengertian manajemen” di atas dokumen dengan topik “apa itu”. Hal ini dapat dilakukan dengan mencegah kata dari daftar *stopword* dianalisa.

Filtering dengan NLTK

```
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau
online. Menurut Andi belanja online lebih praktis & murah."
kalimat =
kalimat.translate(str.maketrans('','',string.punctuation)).lower()

tokens = word_tokenize(kalimat)
listStopword =  set(stopwords.words('indonesian'))

removed = []
for t in tokens:
    if t not in listStopword:
        removed.append(t)

print(removed)


# ouput
# ['andi', 'kerap', 'transaksi', 'rutin', 'daring', 'online', 'andi',
'belanja', 'online', 'praktis', 'murah']
```

Filtering dengan sastrawi

Selain untuk *stemming,* library Sastrawi juga mendukung proses *filtering*. Kita dapat menggunakan `stopWordRemoverFactory` dari modul sastrawi.

Untuk melihat daftar *stopword* yang telah didefinisikan dalam library Sastrawi dapat menggunakan kode berikut :

```
from Sastrawi.StopWordRemover.StopWordRemoverFactory import
StopWordRemoverFactory

factory = StopWordRemoverFactory()
stopwords = factory.get_stop_words()
print(stopwords)
```

Kode diatas akan menampikan *stopword* yang tersedia di library Sastrawi.
Prosos *filtering* pada Sastrawi dapat dilihat pada baris kode dibawah :

```
from Sastrawi.StopWordRemover.StopWordRemoverFactory import
StopWordRemoverFactory
from nltk.tokenize import word_tokenize

factory = StopWordRemoverFactory()
stopword = factory.create_stop_word_remover()

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau
online. Menurut Andi belanja online lebih praktis & murah."
kalimat =
kalimat.translate(str.maketrans('','',string.punctuation)).lower()

stop = stopword.remove(kalimat)
tokens = nltk.tokenize.word_tokenize(stop)
print(tokens)

# output
# ['andi', 'kerap', 'transaksi', 'rutin', 'daring', 'online', 'andi',
'belanja', 'online', 'praktis', 'murah']
```

Kita dapat menambah atau mengurangi kata pada daftar *stopword* sesuai dengan kebutuhan analisa. Pada dasarnya daftar *stopword* pada library Sastrawi tersimpan di dalam list yang anda lihat . Jadi sebenarnya kita tinggal mengubah daftar pada list tersebut. Tetapi hal tersebut bisa menjadi permasalahan apabila pada suatu kasus kita diharuskan menambahkan *stopword* secara dinamis.

4. Stemming

Stemming adalah proses menghilangkan infleksi kata ke bentuk dasarnya, namun bentuk dasar tersebut tidak berarti sama dengan akar kata (*root word*). Misalnya kata “mendengarkan”, “dengarkan”, “didengarkan” akan ditransformasi menjadi kata “dengar”.

Idenya adalah ketika anda mencari dokumen “cara membuka lemari”, anda juga ingin melihat dokumen yang menyebutkan “cara terbuka lemari” atau “cara dibuka lemari” meskipun terdengar tidak enak. Tentunya anda ingin mencocokan semua variasi kata untuk memunculkan dokumen yang paling relevan.

Stemming dengan NLTK (bahasa inggris)

Ada banyak algoritma yang digunakan untuk *stemming*. Salah satu algoritma yang tertua dan paling populer adalah algoritma Porter. Algoritma ini tersedia dalam modul NLTK melalui kelas`PorterStemmer()`.

```
from nltk.stem import PorterStemmer

ps = PorterStemmer()

kata = ["program", "programs", "programer", "programing",
"programers"]

for k in kata:
    print(k, " : ", ps.stem(k))


# ouput
# program  :  program
programs  :  program
programer  :  program
programing  :  program
programers  :  program
```

Selain Porter, NLTK juga mendukung algoritma Lancester, WordNet Lemmatizer, dan SnowBall. Sayangnya proses *stemming* Bahasa Indonesia pada modul NLTK belum didukung.

Apakah kita memerlukan semua tahapan pada text preprocessing?

Tidak ada aturan pasti yang membahas setiap tahapan pada *text preprocessing*. Tentu saja untuk memastikan hasil yang lebih baik dan konsisten semua tahapan harus dilakukan. Untuk memberi gambaran tentang apa yang minimal seharusnya dilakukan, saya telah menguraikan tahapan menjadi **harus dilakukan**, **sebaiknya dilakukan**, dan **tergantung tugas**. Perlu diingat, *less is more*, anda ingin menjaga pendekatan dengan seindah mungkin. Semakin banyak fitur atau tahapan yang anda tambahkan, semakin banyak pula lapisan yang anda harus kupas.

- **Harus dilakukan** meliputi *case folding* (dapat tergantung tugas dalam beberapa kasus)
- **Sebaiknya dilakukan** meliputi normalisasi sederhana — misalnya menstandarkan kata yang hampir sama
- **Tergantung tugas** meliputi normalisasi tingkat lanjut — misalnya mengatasi kata yang tidak biasa, *stopword removal* dan *stemming*

Jadi, untuk mengatasi tugas apapun minimal anda harus melakukan *case folding*. Selanjutnya dapat ditambahkan beberapa normalisasi dasar untuk mendapatkan hasil yang lebih konsisten dan secara bertahap menambahkan tahapan yang lainnya sesuai yang anda inginkan.

Penutup

Dalam tulisan ini kita telah mengetahui langkah dasar dan praktis pada *text preprocessing* beserta *library* yang digunakan dalam python. Selanjutnya hasil dari *text preprocessing* dapat digunakan untuk analisa NLP yang lebih rumit, contohnya *machine translation*. Tidak semua kasus membutuhkan level *preprocessing* yang sama. Dalam beberapa kasus anda bisa menggunakan salah satu tahap dari *preprocessing* paling sederhana yaitu *case folding*. Namun semua tahap akan dibutuhkan apabila anda mempunyai *dataset* dengan level *noise* sangat tinggi.

Masih ada teknik yang bisa dilakukan pada *text preprocessing*, tetapi sesuai kata pengantar diatas, tulisan ini hanya mengulas langkah dasar dan praktis dalam *text preprocessing*.

Selamat mencoba dan bersenang-senang dengan NLP :)

Referensi

1. Adriani, M., Asian, J., Nazief, B., Tahaghoghi, S. M. M., & Williams, H. E. (2007). Stemming Indonesian. ACM Transactions on Asian Language Information Processing, 6(4), 1–33.
2. Tala, Fadillah Z.(1999). *A Study of Stemming Effect on Information Retrieval in* Bahasa Indonesia.
3. Geovedi, Jim.(2014).Karena Data Gak Mungkin Bohong dan karena Bisa Diolah Sesuai Pesanan (https://medium.com/curahan-rekanalar/karena-data-gak-mungkin-bohong-a17ff90cef87).
4. Python Sastrawi (https://github.com/har07/PySastrawi).
5. Natural Language Toolkit -NLTK (https://www.nltk.org).
6. Matplotlib (https://matplotlib.org).

NAMA : AHMAD ALI MA'MUN

NIM : 202420037

Text Mining

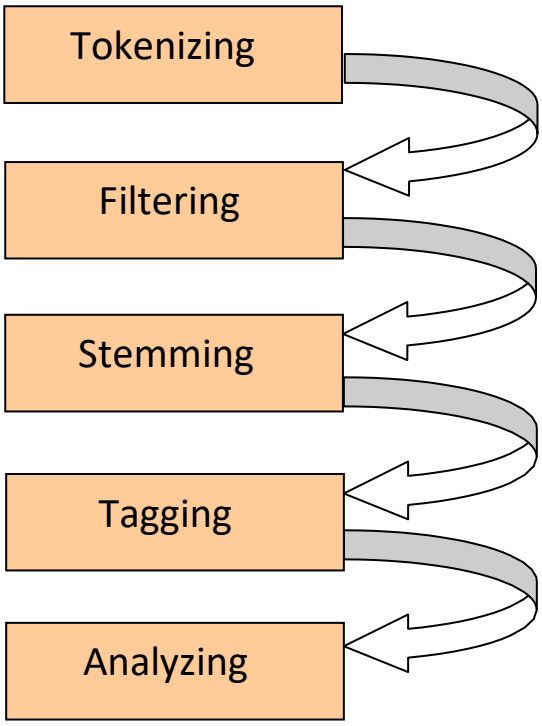
Text mining memiliki definisi menambang data yang berupa teks dimana sumber data biasanya didapatkan dari dokumen, dan tujuannya adalah mencari kata-kata yang dapat mewakili isi dari dokumen sehingga dapat dilakukan analisa keterhubungan antar dokumen

Berdasarkan ketidakteraturan struktur data teks, maka proses text mining memerlukan beberapa tahap awal yang pada intinya adalah mempersiapkan agar teks dapat diubah menjadi lebih terstruktur.

Tujuan dari text mining adalah untuk mendapatkan informasi yang berguna dari sekumpulan dokumen. Jadi, sumber data yang digunakan pada text mining adalah kumpulan teks yang memiliki format yang tidak terstruktur atau minimal semi terstruktur. Adapun tugas khusus dari text mining antara lain yaitu pengkategorisasian teks (text categorization) dan pengelompokan teks text clustering).

Tahapan dalam Text Mining

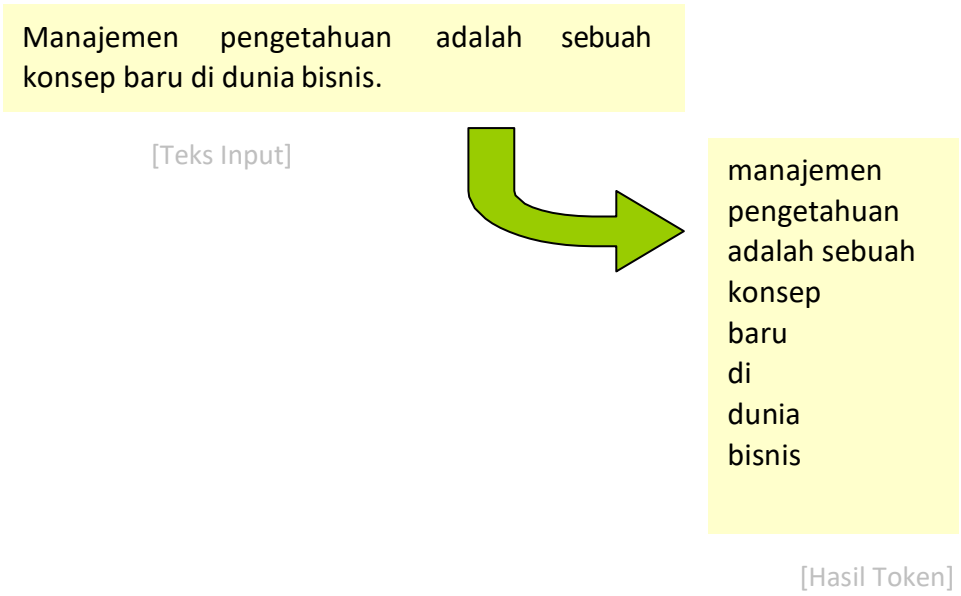
Tahapan yang dilakukan secara umum adalah:



1.Tokenizing

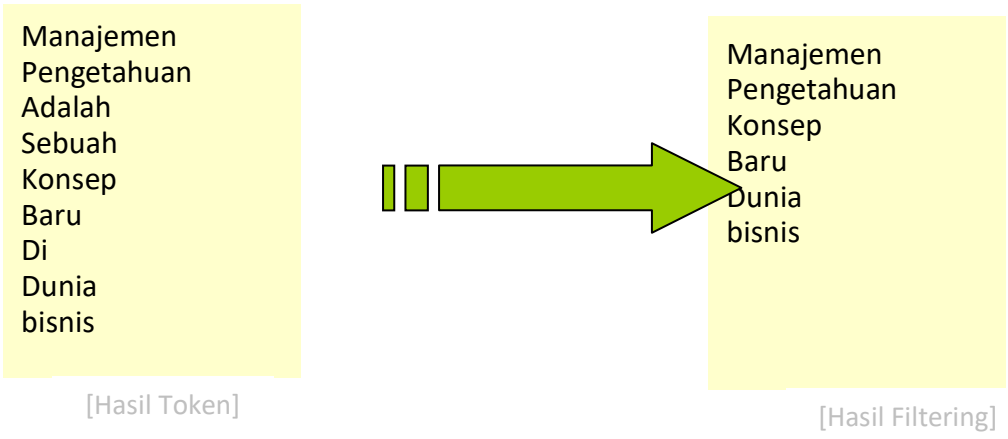
Tahap tokenizing adalah tahap pemotongan string input berdasarkantiap kata yang menyusunnya.

Contoh dari tahap ini adalahsebagai berikut



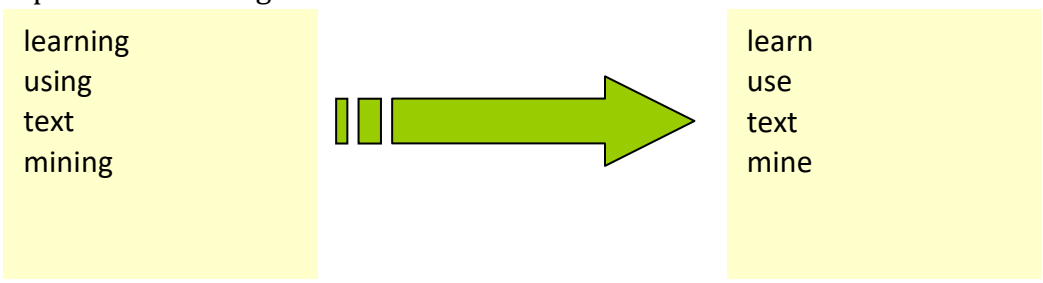
2. Filtering

- Tahap filtering adalah tahap mengambil kata-kata penting dari hasil token.
- Bisa menggunakan algoritma stop list (membuang kata yang kurang penting) atau word list (menyimpan kata penting).
- Contoh dari tahap ini adalah sebagai berikut:



3. Stemming

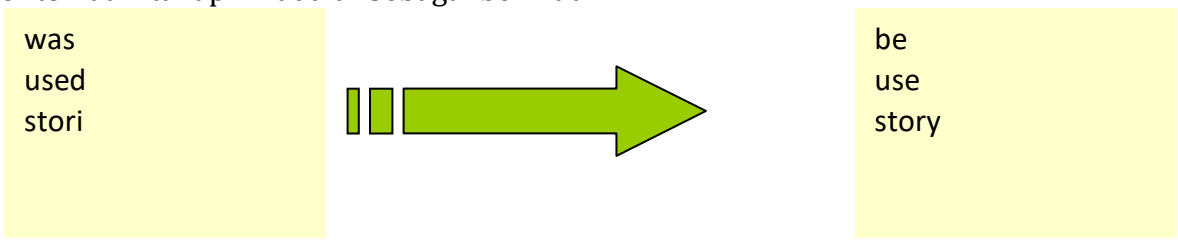
- Tahap stemming adalah tahapmencari root kata dari tiap kata hasil filtering.
- Contoh dari tahap ini adalahsebagai berikut:



[Hasil Filter] [Hasil Stemming]

4. Tagging

- Tahap tagging adalah tahap mencari bentuk awal/root dari tiap katalampau atau kata hasil stemming.
- Contoh dari tahap ini adalahsebagai berikut:

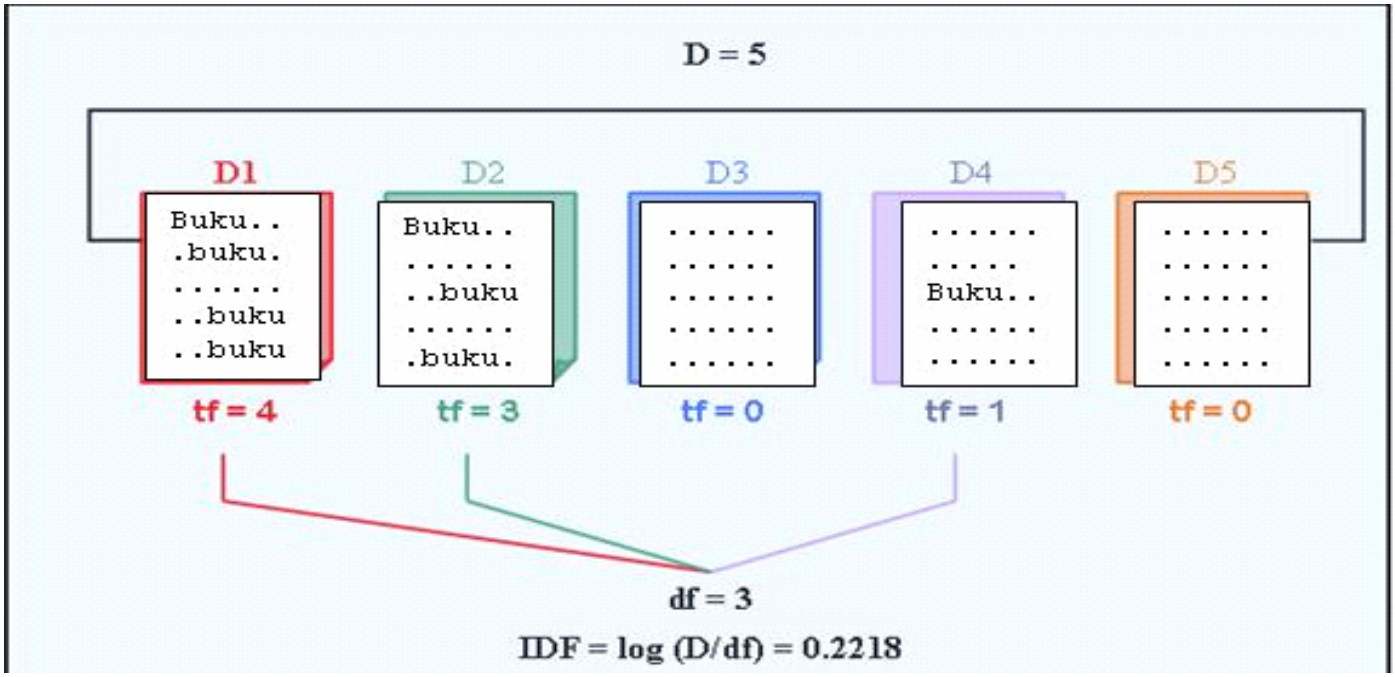


[Hasil Stemming] [Hasil tangging]

5. Analyzing

- Tahap Analyzing merupakan tahap penentuan seberapa jauhketerhubungan antar kata-kata antar dokumen yangada.

Ilustrasi Algoritma Text Mining



D1, D2, D3, D4, D5 = dokumen

Tf = banyak kata yang dicari pada sebuah dokumen

D = total dokumen

Df = banyak dokumen yang mengandung kata yang dicari

Algoritma TF/IDF

Formula yang digunakan untuk menghitung bobot (w) masing-masing dokumen terhadap kata kunci adalah

$$W_{d,t} = tf_{d,t} * IDF_t$$

Dimana:

d = dokumen ke-d

t = kata ke-t dari kata kunci

W = bobot dokumen ke-d terhadap kata ke-t

Setelah bobot (w) masing-masing dokumen diketahui, maka dilakukan proses sorting/pengurutan dimana semakin besar nilai w, semakin besar tingkat similaritas dokumen tersebut terhadap kata yang dicari, demikian sebaliknya.

Ilustrasi TF/IDF

Kata kunci (kk) = pengetahuan logistik Dokumen 1
(D1) = Manajemen transaksi logistik Dokumen 2
(D2) = Pengetahuan antar individu
Dokumen 3 (D3) = Dalam manajemen pengetahuan terdapat transfer pengetahuan logistik
Jadi jumlah dokumen (D) = 3

Setelah melalui proses filtering, maka kata antar pada dokumen 2 serta kata dalam dan terdapat pada dokumen 3 dihapus

Tabel perhitungan TF-IDF

	tf				df	D/df	IDF	W			
token	kk	D1	D2	D3			og(D/df)	kk	D1	D2	D3
manajemen	0	1	0	1	2	1.5	0.176	0	0.176	0	0.176
transaksi	0	1	0	0	1	3	0.477	0	0.477	0	0
logistik	1	1	0	1	2	1.5	0.176	0.176	0.176	0	0.176
transfer	0	0	0	1	1	3	0.477	0	0	0	0.477
pengetahuan	1	0	1	2	2	1.5	0.176	0.176	0	0.176	0.352
individu	0	0	1	0	1	3	0.477	0	0	0.477	0
Total		0.352	0.829	0.653	1.181						

bobot (w) untuk D1 = 0.176 + 0 = 0.176

bobot (w) untuk D2 = 0 + 0.176 = 0.176

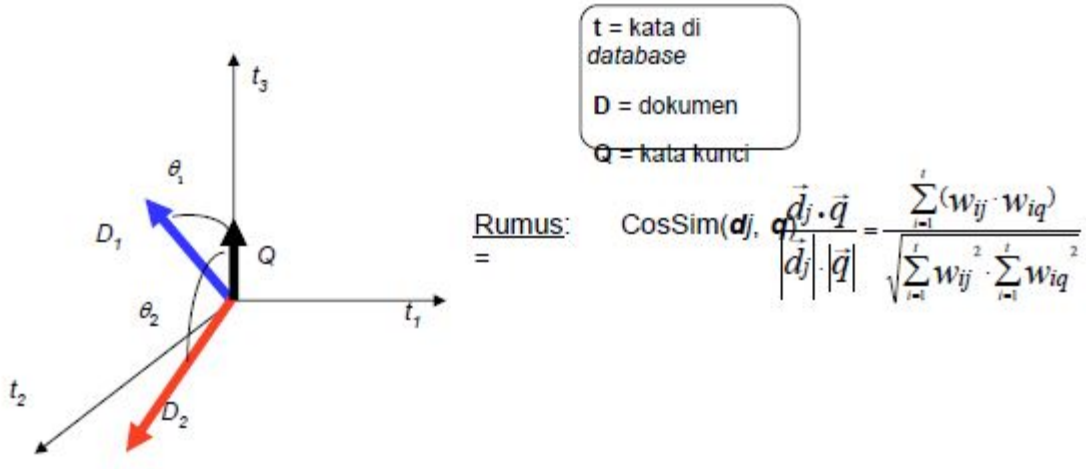
bobot (w) untuk D3 = 0.176 + 0.352 = 0.528

Analisa TFIIDF

- Dari contoh studi kasus di atas, dapat diketahui bahwa nilai bobot (w) dari D1 dan D2 adalah sama.
- Apabila diurutkan maka proses sorting juga tidak akan dapat mengurutkan secara tepat, karena nilai w keduanya sama.
- Untuk mengatasi hal ini, digunakan algoritma dari vector-space model

Vector Space Model

- Ide dari metode ini adalah dengan menghitung nilai cosinus sudut dari dua vektor, yaitu W dari tiap dokumen dan W dari kata kunci



Tabel perhitungan Vector-Space Model

token	Kk	D1	D2	D3	Kk*D1	Kk*D2	kk*D3
manajemen	0	0.031	0	0.031	0	0	0
Transaksi	0	0.228	0	0	0	0	0
Logistik	0.031	0.031	0	0.031	0.031	0	0.031
Transfer	0	0	0	0.228	0	0	0
pengetahuan	0.031	0	0.031	0.124	0	0.031	0.062
Individu	0	0	0.228	0	0	0	0
	Sqrt(kk)	**Sqrt(Di)**			**Sum(kk dot Di)**		
	0.249	**0.539**	**0.509**	**0.643**	**0.031**	**0.031**	

Perhitungan

- **Sqrt(kk)** = Sqrt($\sum_{j=1}^{n} kk_j^2$)
  dimana j = kata di database
- Misalnya untuk Sqrt(kk) = Sqrt($\sum_{j=1}^{n} kk_j^2$)
  = $\sqrt{(0+0+0.031+0+0.031+0)}$
  = $\sqrt{0.062}$ = 0.249
- **Sqrt(D_i)** = Sqrt($\sum_{j=1}^{n} D_{i,j}^2$)
  dimana j = kata di database
- Misalnya untuk Sqrt(D_2) = Sqrt($\sum_{j=1}^{n} D_{2,j}^2$)
  = $\sqrt{(0+0+0+0+0.031+0.228)}$
  = $\sqrt{0.259}$ = 0.509

- **Sum(kk dot D_i)** = $\sum_{j=1}^{n} kk_j D_{i,j}$

  dimana j = kata di database
- Misalnya untuk Sum(kk dot D_3) = $\sum_{j=1}^{n} kk_j D_{3,j}$

  = 0 + 0 + 0.031 + 0 + 0.062 + 0
  = 0.093

Selanjutnya menghitung nilai Cosinus sudut antara vektor kata kunci dengan tiap dokumen dengan rumus:

Cosine (D_i) = sum(kk dot D_i) / [sqrt(kk) * sqrt(D_i))]

- Misalnya untuk D_3 maka:
- Cosine (D_3)= sum(kk dot D_3) / [sqrt(kk) * sqrt(D_3))]
  = 0.093 / [0.249 * 0.643]
  = 0.581

Analisa Vector Space Model

- Demikian juga untuk Cosine dari D_1 dan D_2. Sehingga hasil yang diperoleh untuk ketiga dokumen di atas adalah seperti berikut ini.
- Tabel 2.3 Tabel hasil Vector-Space Model

	D1	D2	D3
Cosine	0.231	0.245	0.581
	Rank 3	Rank 2	Rank 1

- Dari hasil akhir (Cosine) maka dapat diketahui bahwa document 3 (D3) memiliki tingkat similaritas tertinggi kemudian disusul dengan D2 lalu D1.

TUGAS 8
TUTORIAL TEKS MINING

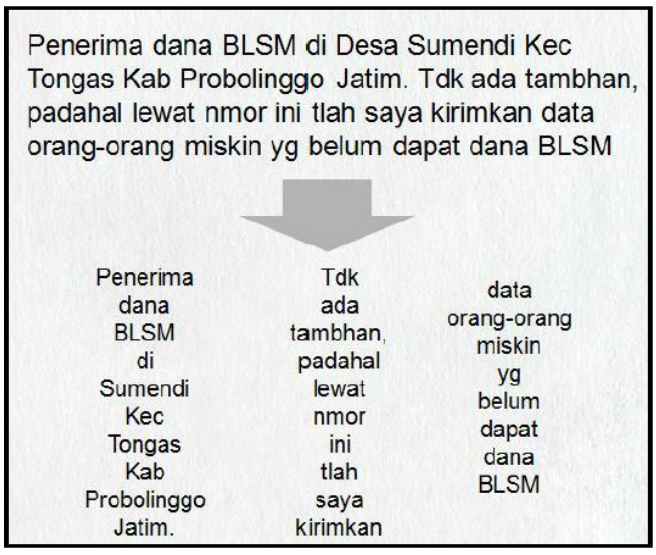
Contoh data sebelum di proses.

Isi Laporan
Yth. Kementerian Dalam Negeri, Saya mau tanya kalau untuk membuat Kartu Keluarga baru, KTP baru dan Akte kelahiran biayanya berapa ya? Mohon informasinya, terima kasih.
Yth. Kepolisian RI, Mengapa Polantas di setiap melakukan penilangan tidak pernah menanyakan kepada pengendara apakah ingin mengikuti sidang karena tidak mengakui kesalahan atau membayar langsung tilang ke bank karena mengakui kesalahan? Selama 20 tahun saya berkendara, saya pernah ditilang 2x, namun tidak pernah diberi lembar biru untuk membayar langsung ke bank. Saya selalu di beri lembar merah untuk ikut sidang padahal saya sudah mengaku bersalah. Sebagai tambahan bukti materi, coba lihat di youtube siaran polisi yang menilang, Polantas selalu memberikan surat tilang merah. Mohon penjelasannya secara lengkap bagaimana sebenarnya aturan penggunaan slip merah dan slip biru ini ketika penilangan? Terima kasih.
Lapor!... saya penerima KKS, ingin bertanya kapan dana bantuan nya akan keluar, dan berapa jumlah dananya, berapa kali pengambilannya dalam satu tahun...
lampu2 PJU dari Jln.Margacinta sampe Derwati perlu diganti,selain banyak mati juga kurang terang.
359mn22552001 Rumah tangga saya tidak menerima bantuan sekolah untuk anak saya

Langkah – langkah preprocessing text:

1. Tokenization

   Proses yang paling awal dilakukan yaitu tokenization. Pada prinsipnya, tokenization adalah proses pemisahan teks menjadi potongan kata yang disebut token. Tokenization dilakukan untuk mendapatkan token atau potongan kata yang akan menjadi entitas yang memiliki nilai dalam penyusunan matriks dokumen pada proses selanjutnya.

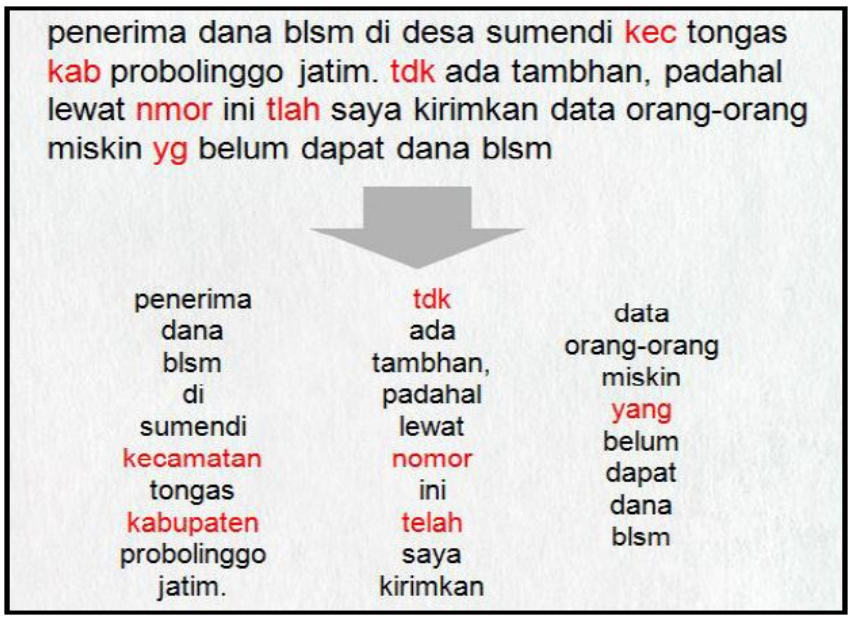


2. Case Folding

Case Folding merupakan proses pengubahan huruf dalam dokumen menjadi satu bentuk, misalnya huruf kapital menjadi huruf kecil dan sebaliknya.

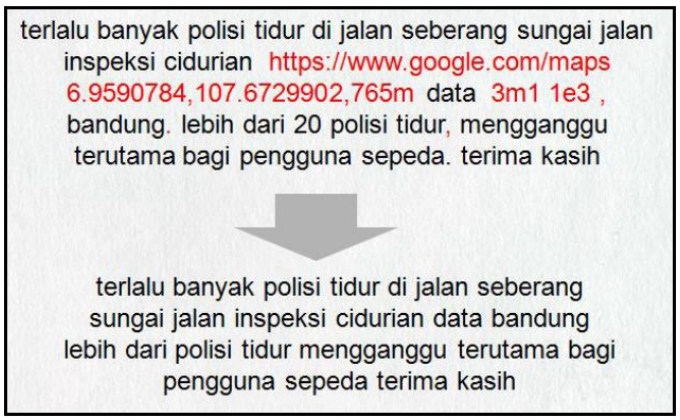
3. **Spelling Normalization**

Proses ini merupakan proses perbaikan atau subtitusi kata-kata yang salah eja atau disingkat dalam bentuk tertentu. Subtitusi kata dilakukan untuk menghindari jumlah perhitungan dimensi kata yang melebar.



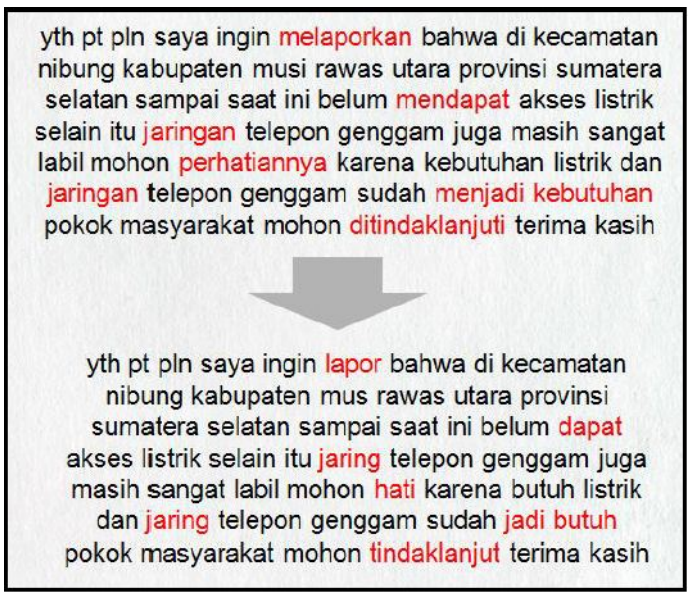
4. **Filtering**

Kata dan tanda baca yang tidak memiliki arti yang signifikan atau termasuk noise (pengganggu) akan dieliminasi. Kata atau frase yang tidak bermakna secara signifikan, misalnya hashtag (#), url, tanda baca tertentu (emoticon), dan lainnya. Laporan banyak diterima lewat sms, sehingga menyebabkan banyaknya tanda baca dan frase yang masuk pada penarikan laporan pada sistem LAPOR! yang tidak bisa diproses atau mengurangi performa pengolahan data pada tahap selanjutnya.



5. **Stemming**

Pada bagian ini dilakukan proses untuk menemukan akar kata atau kata dasar dari sebuah kata. Proses *stemming* dilakukan dengan menghilangkan semua imbuhan (afiks) baik yang terdiri dari awalan (prefiks) sisipan (infiks) maupun akhiran (sufiks) dan kombinasi dari awalan dan akhiran (konfiks). Stemming digunakan untuk mengganti bentuk dari suatu kata menjadi kata dasar sesuai dengan struktur morfologi bahasa indonesia yang baik dan benar.



Data setelah proses Stemming

Isi Laporan
yth menteri dalam neger saya mau tanya kalau untuk buat kartu keluarga baru kartu tanda duduk baru dan akte lahir biaya berapa ya mohon informasi terima kasih
yth polisi r apa polantas di tiap laku ilang tidak pernah anya kepada endara apakah ingin ikut sidang karena tidak aku salah atau bayar langsung tilang ke bank karena aku salah lama tahun saya kendara saya pernah tilang x namun tidak pernah beri lembar biru untuk bayar langsung ke bank saya selalu di beri lembar merah untuk ikut sidang padahal saya sudah a sa bagai tambah bukti materi coba lihat di youtube siar polisi yang ilang polantas selalu ikan surat tilang merah mohon jelas cara lengkap bagaimana benar atur guna slip merah dan slip biru ini ketika ilang terima kasih
lapor saya erima kks ingin ta kapan dana bantu nya akan keluar dan berapa jumlah dana berapa kali ambil dalam satu tahun
lampu pju dari jln margacinta sampai derwat perlu ganti selain banyak mati juga kurang terang
rumah tangga saya tidak erima bantu sekolah untuk anak saya

Nama : Andry Meylani
NIM : 202420009
TUGAS 08

Tutorial text mining

Tokenizing

Tokenizing adalah proses pemisahan teks menjadi potongan-potongan yang disebut sebagai token untuk kemudian di analisa. Kata, angka, simbol, tanda baca dan entitas penting lainnya dapat dianggap sebagai token. Didalam NLP, token diartikan sebagai "kata" meskipun tokenize juga dapat dilakukan pada paragraf maupun kalimat.

Tokenizing kata

Sebuah kalimat atau data dapat dipisah menjadi kata-kata dengan kelas *word_tokenize()* pada modul NLTK.

```
# impor word_tokenize dari modul nltk
from nltk.tokenize import word_tokenize

kalimat = "Andi kerap melakukan transaksi rutin secara daring
atau online."

tokens = nltk.tokenize.word_tokenize(kalimat)
print(tokens)# ouput
# ['Andi', 'kerap', 'melakukan', 'transaksi', 'rutin', 'secara',
'daring', 'atau', 'online', '.']
```

Dari output kode diatas terdapat kemunculan tanda baca titik(.) dan koma (,) serta token "Andi" yang masih menggunakan huruf besar pada awal kata. Hal tersebut nantinya dapat menggangu proses perhitungan dalam penerapan algoritma. Jadi, sebaiknya teks telah melewati tahap case folding sebelum di tokenize agar menghasilkan hasil yang lebih konsisten.

Kita dapat menambahkan fungsi case folding untuk menghilangkan tanda baca dan mengubah teks ke dalam bentuk lowercase.

```
kalimat =
kalimat.translate(str.maketrans('','',string.punctuation)).lower
()# output
# ['andi', 'kerap', 'melakukan', 'transaksi', 'rutin', 'secara',
'daring', 'atau', 'online']
```

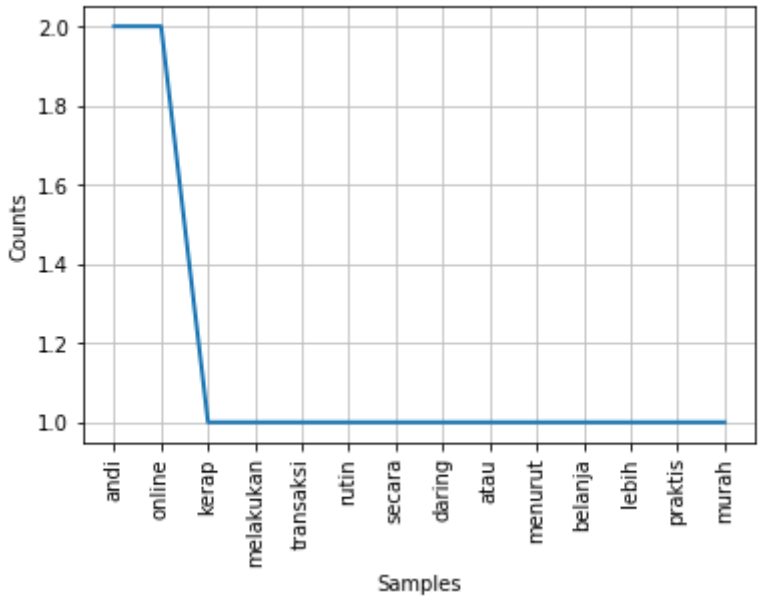
Kita juga bisa mendapatkan informasi frekuensi kemunculan setiap token dengan kelas FreqDist() yang sudah tersedia pada modul NLTK.

```
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDistkalimat = "Andi kerap
melakukan transaksi rutin secara daring atau online. Menurut
Andi belanja online lebih praktis & murah."
kalimat =
```

```
kalimat.translate(str.maketrans('','',string.punctuation)).lower
()

tokens = nltk.tokenize.word_tokenize(kalimat)
kemunculan = nltk.FreqDist(tokens)
print(kemunculan.most_common())# output
# [('andi', 2), ('online', 2), ('kerap', 1), ('melakukan', 1),
('transaksi', 1), ('rutin', 1), ('secara', 1), ('daring', 1),
('atau', 1), ('menurut', 1), ('belanja', 1), ('lebih', 1),
('praktis', 1), ('murah', 1)]
```

Untuk menggambarkan ke dalam bentuk grafik, kita perlu menginstall library Matplotlib.

```
import matplotlib.pyplot as
pltkemunculan.plot(30,cumulative=False)
plt.show()
```



Grafik frekuensi kemunculan kata pada dokumen

TUGAS 08

ADVANCED DATABASE

KELAS MTI 23 A

Di Susun Oleh :

Ari Hardiyantoro Susanto

NIM : 202420015

Dosen Pengasuh :

Tri Basuki Kurniawan , S.Kom., M.Eng. Ph.D

Program Pasca Sarjana

Universitas Bina Darma Palembang

2020/2021

Salah satu keunggulan python adalah mendukung banyak *open-source library*. Ada banyak *library* python yang dapat digunakan untuk melakukan dan mengimplementasikan masalah dalam NLP.

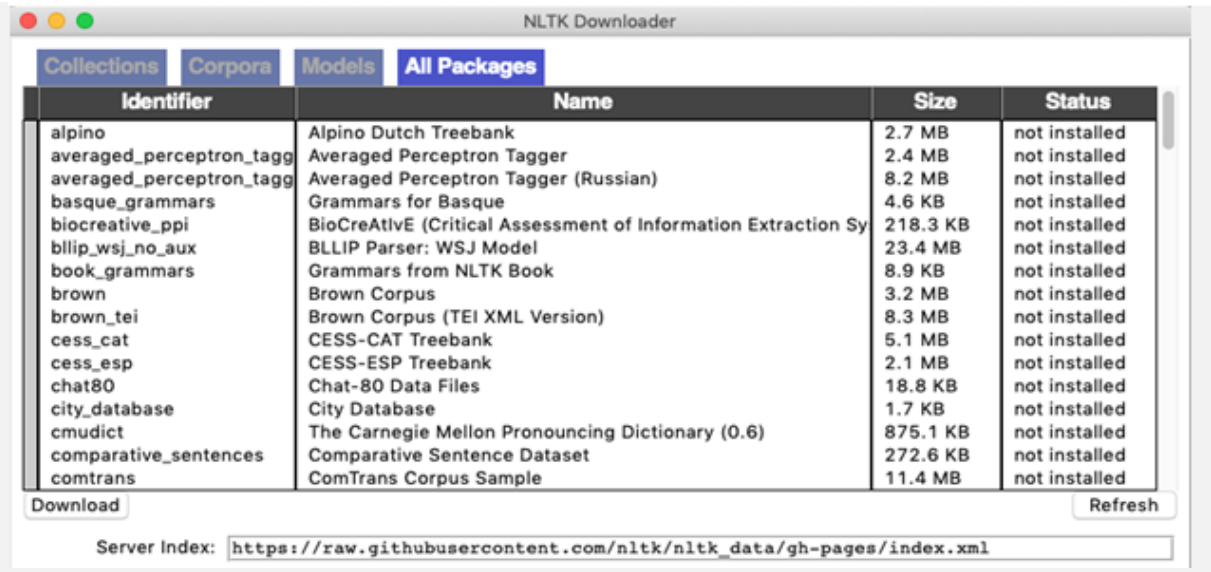
Natural Language Toolkit (NLTK)

Natural Language Toolkit atau disingkat NLTK, adalah *libray* python untuk bekerja dengan permodelan teks. NLTK menyediakan alat yang baik mempersiapkan teks sebelum digunakan pada *machine learning* atau algoritma *deep learning*. Cara termudah untuk menginstall NLTK adalah menggunakan "pip" pada *command line/terminal*.

```
pip install nltk
```

Langkah pertama yang perlu anda lakukan setelah menginstall NLTK adalah mengunduh paket NLTK.

```
Import nltk
nltk.download()
```



NLTK

Python Sastrawi

Python Sastrawi adalah pengembangan dari proyek PHP Sastrawi. Python Sastrawi merupakan library sederhana yang dapat mengubah kata berimbuhan bahasa Indonesia menjadi bentuk dasarnya. Sastrawi juga dapat diinstal melalui "pip".

```
pip install Sastrawi
```

Penggunaan Python Sastrawi sangat sederhana seperti baris kode dibawah ini :

```
# import StemmerFactory class
from Sastrawi.Stemmer.StemmerFactory import
StemmerFactory

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

# stemming process
sentence = 'Perekonomian Indonesia sedang dalam pertumbuhan yang membanggakan'

output   = stemmer.stem(sentence)

print(output)
# ekonomi indonesia sedang dalam tumbuh yang bangga

print(stemmer.stem('Mereka meniru-nirukannya'))
# mereka tiru
```

1. Case folding

Case folding adalah salah satu bentuk *text preprocessing* yang paling sederhana dan efektif meskipun sering diabaikan. Tujuan dari *case folding* untuk mengubah semua huruf dalam dokumen menjadi huruf kecil. Hanya huruf 'a' sampai 'z' yang diterima. Karakter selain huruf dihilangkan dan dianggap *delimiter*. Pada tahap ini tidak menggunakan *external library* apapun, kita bisa memanfaatkan modul yang tersedia di python.

Ada beberapa cara yang dapat digunakan dalam tahap *case folding*, anda dapat menggunakan beberapa atau menggunakan semuanya, tergantung pada tugas yang diberikan.

Mengubah text menjadi lowercase

Salah satu contoh pentingnya penggunaan *lower case* adalah untuk mesin pencarian. Bayangkan anda sedang mencari dokumen yang mengandung "indonesia" namun tidak ada hasil yang muncul karena "indonesia" di indeks sebagai "INDONESIA". Siapa yang

harus kita salahkan? U.I. *designer* yang mengatur *user interface* atau *developer* yang mengatur sistem dalam indeks pencarian?

Contoh dibawah menunjukan bagaimana python mengubah teks menjadi *lowercase* :

```
kalimat = "Berikut ini adalah 5 negara dengan pendidikan terbaik di dunia adalah Korea Selatan, Jepang, Singapura, Hong Kong, dan Finlandia."
lower_case = kalimat.lower()
print(lower_case)
# output
# berikut ini adalah 5 negara dengan pendidikan terbaik di dunia adalah korea selatan, jepang, singapura, hong kong, dan finlandia.
```

Menghapus angka

Hapuslah angka jika tidak relevan dengan apa yang akan anda analisa, contohnya seperti nomor rumah, nomor telepon, dll. *Regular expression* (*regex*) dapat digunakan untuk menghapus karakter angka. Python memiliki modulre untuk melakukan hal – hal yang berkaitan dengan *regex*.

Contoh dibawah menunjukan bagaimana python menghapus angka dalam sebuah kalimat :

```
import re # impor modul regular expression
kalimat = "Berikut ini adalah 5 negara dengan pendidikan terbaik di dunia adalah Korea Selatan, Jepang, Singapura, Hong Kong, dan Finlandia."
hasil = re.sub(r"\d+", "", kalimat)
print(hasil)

# ouput
# Berikut ini adalah negara dengan pendidikan terbaik di dunia adalah Korea Selatan, Jepang, Singapura, Hong Kong, dan Finlandia.
```

Menghapus tanda baca

Sama halnya dengan angka, tanda baca dalam kalimat tidak memiliki pengaruh pada *text preprocessing*. Menghapus tanda baca seperti [!”#$%&’()*+,-./:;<=>?@[\]^_`{|}~] dapat dilakukan di pyhton seperti dibawah ini :

```
kalimat = "Ini &adalah [contoh] kalimat? {dengan} tanda. baca?!!"
hasil =
 kalimat.translate(str.maketrans("","",string.punctuation))
print(hasil)
```

```
# output
# Ini adalah contoh kalimat dengan tanda baca
```

Menghapus whitepace (karakter kosong)

Untuk menghapus spasi di awal dan akhir, anda dapat menggunakan fungsi strip() pada pyhton. Perhatikan kode dibawah ini :

```
kalimat = " \t ini kalimat contoh\t "
hasil = kalimat.strip()
print(hasil)

# output
# ini kalimat contoh
```

2. Tokenizing

Tokenizing adalah proses pemisahan teks menjadi potongan-potongan yang disebut sebagai token untuk kemudian di analisa. Kata, angka, simbol, tanda baca dan entitas penting lainnya dapat dianggap sebagai token. Didalam NLP, token diartikan sebagai "kata" meskipun *tokenize* juga dapat dilakukan pada paragraf maupun kalimat.

Fungsi split() pada pyhton dapat digunakan untuk memisahkan teks. Perhatikan contoh dibawah ini :

```
kalimat = "rumah idaman adalah rumah yang bersih"
pisah = kalimat.split()
print(pisah)

# output
# ['rumah', 'idaman', 'adalah', 'rumah', 'yang', 'bersih']
```

Sayangnya, tahap *tokenizing* tidak sesederhana itu. Dari *output* kode diatas kita akan mengolah kata "rumah" dan "rumah" sebagai 2 entitas yang berbeda. Permasalahan ini tentunya akan mempengaruhi hasil dari analisa teks itu sendiri.

Untuk mengatasi masalah ini kita dapat menggunakan modul dari NLTK yang sudah diinstal sebelumnya.

Tokenizing kata

Sebuah kalimat atau data dapat dipisah menjadi kata-kata dengan kelas word_tokenize() pada modul NLTK.

```
# impor word_tokenize dari modul nltk
from nltk.tokenize import word_tokenize

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online."

tokens = nltk.tokenize.word_tokenize(kalimat)
print(tokens)

# ouput
# ['Andi', 'kerap', 'melakukan', 'transaksi', 'rutin', 'secara', 'daring', 'atau', 'online', '.']
```

Dari *output* kode diatas terdapat kemunculan tanda baca titik(.) dan koma (,) serta token "Andi" yang masih menggunakan huruf besar pada awal kata. Hal tersebut nantinya dapat menggangu proses perhitungan dalam penerapan algoritma. Jadi, sebaiknya teks telah melewati tahap *case folding* sebelum di *tokenize* agar menghasilkan hasil yang lebih konsisten.

Kita dapat menambahkan fungsi *case folding* untuk menghilangkan tanda baca dan mengubah *teks* ke dalam bentuk *lowercase*.

```
kalimat = kalimat.translate(str.maketrans('','',string.punctuation)).lower()

# output
# ['andi', 'kerap', 'melakukan', 'transaksi', 'rutin', 'secara', 'daring', 'atau', 'online']
```

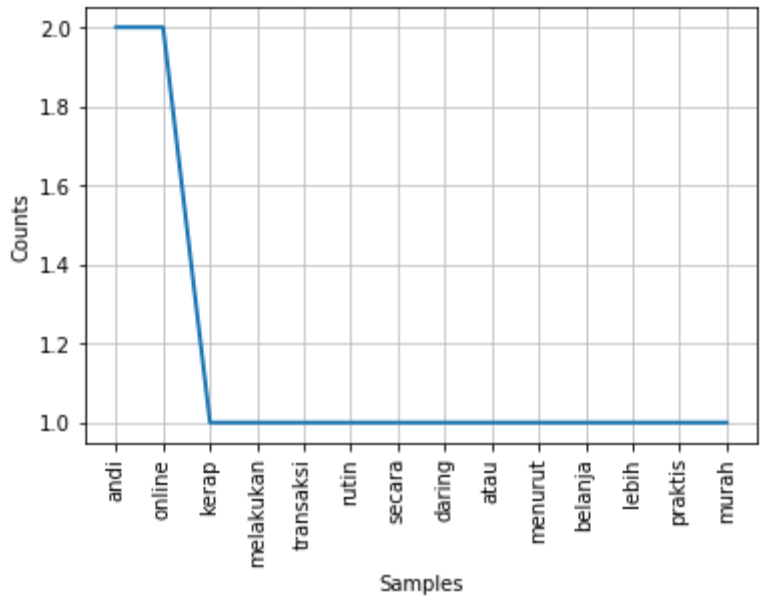
Kita juga bisa mendapatkan informasi frekuensi kemunculan setiap token dengan kelas FreqDist() yang sudah tersedia pada modul NLTK.

```
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDist
kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online. Menurut Andi belanja online lebih praktis & murah."
kalimat = kalimat.translate(str.maketrans('','',string.punctuation)).lower()
tokens = nltk.tokenize.word_tokenize(kalimat)
kemunculan = nltk.FreqDist(tokens)
print(kemunculan.most_common())
```

```
# output
# [('andi', 2), ('online', 2), ('kerap', 1), ('melakukan', 1), ('transaksi', 1), ('rutin', 1), ('secara', 1), ('daring', 1), ('atau', 1), ('menurut', 1), ('belanja', 1), ('lebih', 1), ('praktis', 1), ('murah', 1)]
```

Untuk menggambarkan ke dalam bentuk grafik, kita perlu menginstall library Matplotlib.

```
import matplotlib.pyplot as plt

kemunculan.plot(30,cumulative=False)
plt.show()
```



Grafik frekuensi kemunculan kata pada dokumen

Tokenizing kalimat

Prinsip yang sama dapat diterapkan untuk memisahkan kalimat pada paragraf. Anda dapat menggunkan kelas sent_tokenize() pada modul NLTK. Saya telah menambahkan kalimat pada contoh seperti dibawah ini :

```
# impor sent_tokenize dari modul nltk
from nltk.tokenize import sent_tokenize

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online. Menurut Andi belanja online lebih praktis & murah."

tokens = nltk.tokenize.sent_tokenize(kalimat)
print(tokens)
```

```
# ouput
# ['Andi kerap melakukan transaksi rutin secara daring atau online.', 'Menurut Andi belanja online lebih praktis & murah.']
```

3. Filtering (Stopword Removal)

Filtering adalah tahap mengambil kata-kata penting dari hasil token dengan menggunakan algoritma *stoplist* (membuang kata kurang penting) atau *wordlist* (menyimpan kata penting).

Stopword adalah kata umum yang biasanya muncul dalam jumlah besar dan dianggap tidak memiliki makna. Contoh *stopword* dalam bahasa Indonesia adalah “yang”, “dan”, “di”, “dari”, dll. Makna di balik penggunaan *stopword* yaitu dengan menghapus kata-kata yang memiliki informasi rendah dari sebuah teks, kita dapat fokus pada kata-kata penting sebagai gantinya.

Contoh penggunaan *filtering* dapat kita temukan pada konteks mesin pencarian. Jika permintaan pencarian anda adalah “apa itu pengertian manajemen?” tentunya anda ingin sistem pencarian fokus pada memunculkan dokumen dengan topik tentang “pengertian manajemen” di atas dokumen dengan topik “apa itu”. Hal ini dapat dilakukan dengan mencegah kata dari daftar *stopword* dianalisa.

Filtering dengan NLTK

```
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online. Menurut Andi belanja online lebih praktis & murah."
kalimat = kalimat.translate(str.maketrans('','',string.punctuation)).lower()

tokens = word_tokenize(kalimat)
listStopword =  set(stopwords.words('indonesian'))

removed = []
for t in tokens:
    if t not in listStopword:
        removed.append(t)

print(removed)
```

```
# ouput
# ['andi', 'kerap', 'transaksi', 'rutin', 'daring', 'online', 'andi', 'belanja', 'online', 'praktis', 'murah']
```

Filtering dengan sastrawi

Selain untuk *stemming,* library Sastrawi juga mendukung proses *filtering*. Kita dapat menggunakan **stopWordRemoverFactory** dari modul sastrawi.

Untuk melihat daftar *stopword* yang telah didefinisikan dalam library Sastrawi dapat menggunakan kode berikut :

```
from Sastrawi.StopWordRemover.StopWordRemoverFactory
import StopWordRemoverFactory

factory = StopWordRemoverFactory()
stopwords = factory.get_stop_words()
print(stopwords)
```

Kode diatas akan menampikan *stopword* yang tersedia di library Sastrawi. Prosos *filtering* pada Sastrawi dapat dilihat pada baris kode dibawah :

```
from Sastrawi.StopWordRemover.StopWordRemoverFactory
import StopWordRemoverFactory
from nltk.tokenize import word_tokenize

factory = StopWordRemoverFactory()
stopword = factory.create_stop_word_remover()

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online. Menurut Andi belanja online lebih praktis & murah."
kalimat = kalimat.translate(str.maketrans('','',string.punctuation)).lower()

stop = stopword.remove(kalimat)
tokens = nltk.tokenize.word_tokenize(stop)
print(tokens)

# output
# ['andi', 'kerap', 'transaksi', 'rutin', 'daring', 'online', 'andi', 'belanja', 'online', 'praktis', 'murah']
```

Kita dapat menambah atau mengurangi kata pada daftar *stopword* sesuai dengan kebutuhan analisa. Pada dasarnya daftar *stopword* pada library Sastrawi tersimpan di dalam list yang anda lihat disini. Jadi sebenarnya kita tinggal mengubah daftar pada list tersebut. Tetapi hal tersebut bisa menjadi permasalahan apabila pada suatu kasus kita diharuskan menambahkan *stopword* secara dinamis.

Library Sastrawi dapat mengatasi permasalahan tersebut, perhatikan kode dibawah ini :

```
from Sastrawi.StopWordRemover.StopWordRemoverFactory
import StopWordRemoverFactory, StopWordRemover, ArrayDictionary
from nltk.tokenize import word_tokenize

stop_factory = StopWordRemoverFactory().get_stop_words() #load defaul stopword
more_stopword = ['daring', 'online'] #menambahkan stopword

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online. Menurut Andi belanja online lebih praktis & murah."
kalimat = kalimat.translate(str.maketrans('','',string.punctuation)).lower()

data = stop_factory + more_stopword #menggabungkan stopword

dictionary = ArrayDictionary(data)
str = StopWordRemover(dictionary)
tokens = nltk.tokenize.word_tokenize(str.remove(kalimat))

print(tokens)

# output
# ['andi', 'kerap', 'transaksi', 'rutin', 'andi', 'belanja', 'praktis', 'murah']
```

Menurut Jim Geovedi pada artikelnya menjelaskan bahwa penyesuaian daftar *stopwords* perlu dilakukan setiap pertama kali melakukan proyek analisa. Memang bukan sesuatu yang melelahkan, tapi jika tidak dilakukan maka ini akan dapat mengakibatkan salah interpretasi terhadap data.

4. Stemming

Stemming adalah proses menghilangkan infleksi kata ke bentuk dasarnya, namun bentuk dasar tersebut tidak berarti sama dengan akar kata (*root word*). Misalnya kata “mendengarkan”, “dengarkan”, “didengarkan” akan ditransformasi menjadi kata “dengar”.

Idenya adalah ketika anda mencari dokumen “cara membuka lemari”, anda juga ingin melihat dokumen yang menyebutkan “cara terbuka lemari” atau “cara dibuka lemari” meskipun terdengar tidak enak. Tentunya anda ingin mencocokan semua variasi kata untuk memunculkan dokumen yang paling relevan.

Stemming dengan NLTK (bahasa inggris)

Ada banyak algoritma yang digunakan untuk *stemming*. Salah satu algoritma yang tertua dan paling populer adalah algoritma Porter. Algoritma ini tersedia dalam modul NLTK melalui kelas PorterStemmer().

```
from nltk.stem import PorterStemmer
ps = PorterStemmer()
kata = ["program", "programs", "programer", "programing", "programers"]

for k in kata:
    print (k, " : ", ps.stem(k))

# ouput
# program : program
programs : program
programer : program
programing : program
programers : program
```

Selain Porter, NLTK juga mendukung algoritma Lancester, WordNet Lemmatizer, dan SnowBall. Sayangnya proses *stemming* Bahasa Indonesia pada modul NLTK belum didukung.

Stemming bahasa indonesia menggunakan Python Sastrawi

Proses *stemming* antara satu bahasa dengan bahasa yang lain tentu berbeda. Contohnya pada teks berbahasa inggris, proses yang diperlukan hanya proses menghilangkan sufiks. Sedangkan pada teks berbahasa Indonesia semua kata imbuhan baik itu sufiks dan prefiks juga dihilangkan.

Untuk melakukan *stemming* bahasa Indonesia kita dapat menggunakan library Python Sastrawi yang sudah kita siapkan di awal. *Library* Sastrawi menerapkan Algoritma Nazief dan Adriani dalam melakukan *stemming* bahasa Indonesia.

```
from Sastrawi.Stemmer.StemmerFactory
import StemmerFactory

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

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online. Menurut Andi belanja online lebih praktis & murah."

hasil = stemmer.stem(kalimat)

print(hasil)

# ouput
# andi kerap laku transaksi rutin cara daring atau online turut andi belanja online lebih praktis murah
```

TEXT MINING

Text mining adalah salah satu bidang khusus dalam data mining yang memiliki definisi menambang data berupa teks dimana sumber data biasanya didapatkan dari dokumen dan tujuannya adalah mencari kata-kata yang dapat mewakili isi dari dokumen sehingga dapat dilakukan analisa keterhubungan antar dokumen (Mooney, 2006).

Mining dapat juga diartikan sebagai penambangan data berupa teks yang bersumber dari dokumen untuk mencari karta-kata yang merupakan perwakilan isi atau pembentuk dokumen teks sehingga penganalisisan dapat dibuat

Text mining dapat menganalisa dokumen, mengelompokkan dokumen berdasarkan kata-kata yang terkandung di dalamnya, serta menentukan kesamaan di antara dokumen untuk mengetahui bagaimana mereka berhubungan dengan variabel lainnya (Statsoft, 2015). Penerapan yang paling umum dilakukan text mining saat ini misalnya penyaringan spam, analisa sentimen, mengukur preferensi pelanggan, meringkas dokumen, pengelompokan topik penelitian, dan banyak lainnya.

Tujuan Text Mining

Tujuan dari text mining adalah untuk mendapatkan informasi yang berguna dari sekumpulan dokumen. Jadi, sumber data yang digunakan pada text mining adalah kumpulan teks yang memiliki format yang tidak terstruktur atau minimal semi terstruktur. Adapun tugas khusus dari text mining antara lain yaitu pengkategorisasian teks (text categorization) dan pengelompokan teks text clustering).

Text mining merupakan penerapan konsep dan teknik data mining untuk mencari pola dalam teks, yaitu proses penganalisisan teks guna menyarikan informasi yang bermanfaat untuk tujuan tertentu.

Berdasarkan ketidakteraturan struktur data teks, maka proses text mining memerlukan beberapa tahap awal yang pada intinya adalah mempersiapkan agar teks dapat diubah menjadi lebih terstruktur.

Perbedaan Text Mining dengan Data Mining

Perbedaan mendasar dengan data mining pada umumnya, Text Mining mengolah data teks yang tidak terstruktur, maka proses text mining memerlukan beberapa tahap awal (Preprocessing) yang pada intinya adalah mempersiapkan agar teks dapat dapat di ubah menjadi lebih terstruktur.

Perbedaan	Data Mining	Text Mining
Data Object	Numerical & categorical data	Textual data
Data structure	Structured	Unstructured & semi-structured
Data representation	Straightforward	Complex
Space dimension	< tens of thousands	> tens of thousands
Methods	Data analysis, machine learning, Neural Network, etc.	Data mining, information Retrieval, NLP, etc.
Maturity	Broad implementation since 1994	Broad implementation starting 2000

Cara Kerja Text Mining

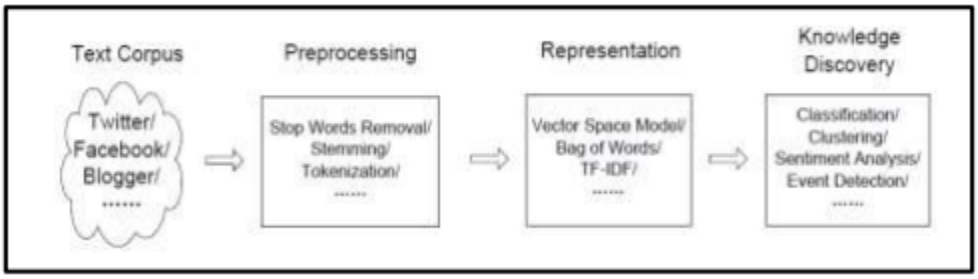
1. Langkah dasar dalam text mining melibatkan konversi teks menjadi data semi terstruktur.
2. Setelah mengubah teks yang tidak terstruktur menjadi data semi terstruktur, langkah selanjutnya adalah menerapkan Teknik analisis untuk proses classification, Clustering, prediction
3. Menemukan pola yang lebih baik dari hasil perubahan teks yang tidak terstruktur menjadi data semi terstruktur.
4. Melakukan pelatihan mode untuk mendeteksi pola pada teks baru dan tidak terlihat.

Text Mining sendiri memiliki beberapa tipe antara lain (Abbot, 2013) :

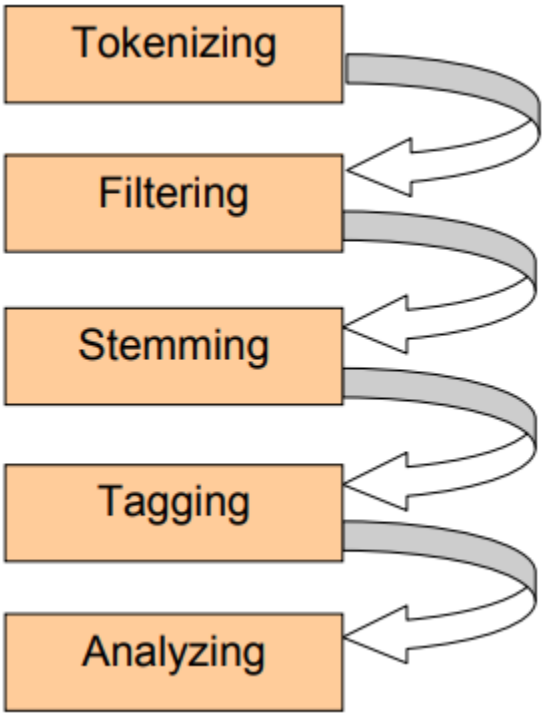
1. *Search and Information Retrieval*
   Menyimpan dan menemukan kembali dokumen teks, termasuk mesin pencari dan kata kunci pencarian.
2. *Document Clustering*
   Pengelompokan dan pengkategorian istilah, potongan, paragraf, atau dokumen menggunakan metode mining.
3. *Document Classification*
   Pengelompokan dan pengkategorian istilah, potongan, paragraf, atau dokumen menggunakan metode document classification.
4. *Web Mining*
   Data dan text mining pada internet yang fokus pada skala dan antar hubungan pada website.
5. *Information Extraction*
   Identifikasi dan ekstraksi fakta yang relevan.

6. *Natural Language Processing*
   Pemrosesan bahasa tingkat rendah yang biasanya digunakan untuk bahasa komputasi.
7. *Concept Extraction*
   Pengelompokan kata dan frase dalam grup yang sama.

Data terpilih yang akan dianalisis pertama akan melewati tahap Pra-proses dan representasi teks, hingga akhirnya dapat dilakukan knowledge discovery.

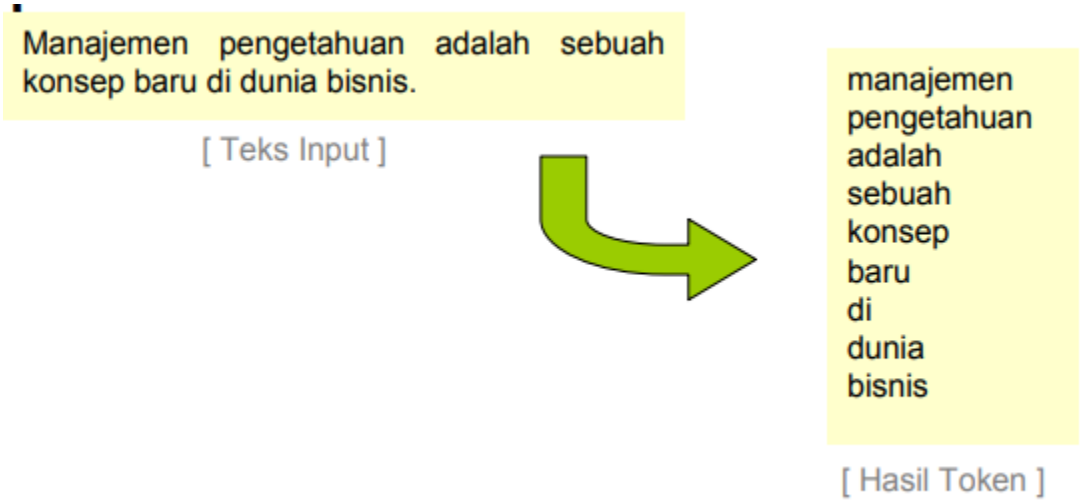


Tahapan pra-pemrosesan dalam Text Mining antara lain menurut Mooney (2006) terdiri dari beberapa fitur antara lain:



1. Tokenizing

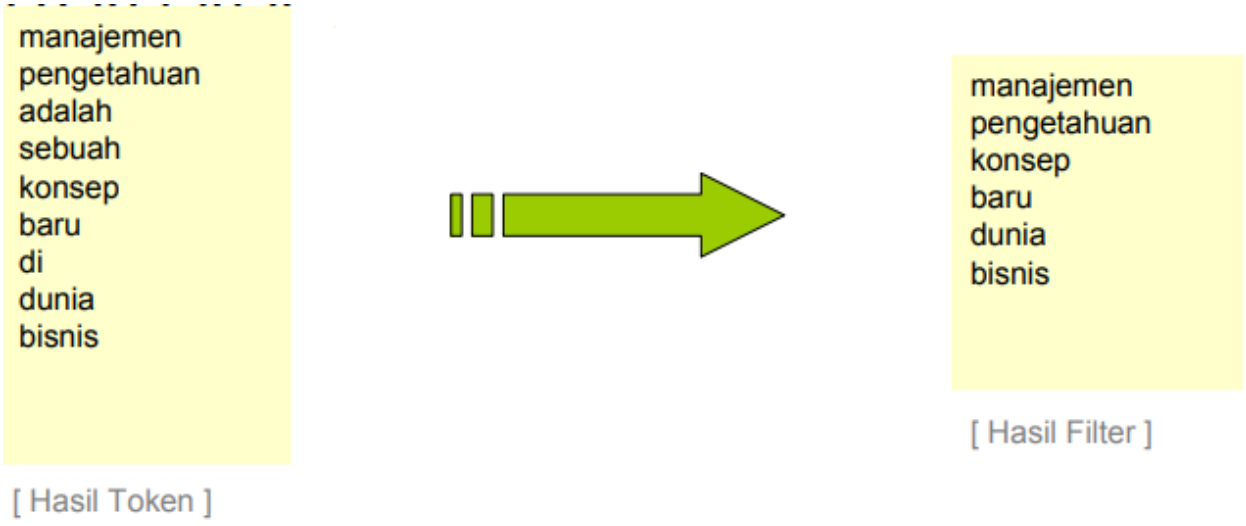
Tahap pemotongan string input berdasarkan tiap kata yang menyusunnya.



2. Filtering

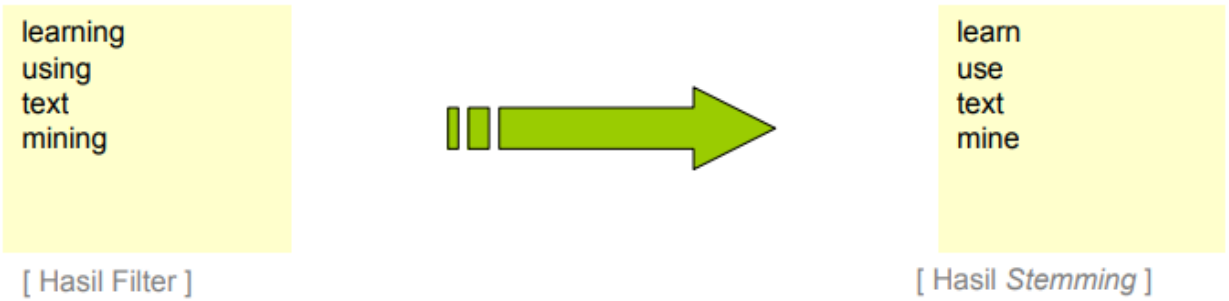
Tahap pengambilan kata-kata yang penting dari hasil token.

Bisa menggunakan algoritma stop list (membuang kata yang kurang penting) atau word list (menyimpan kata yang penting).



3. Stemming

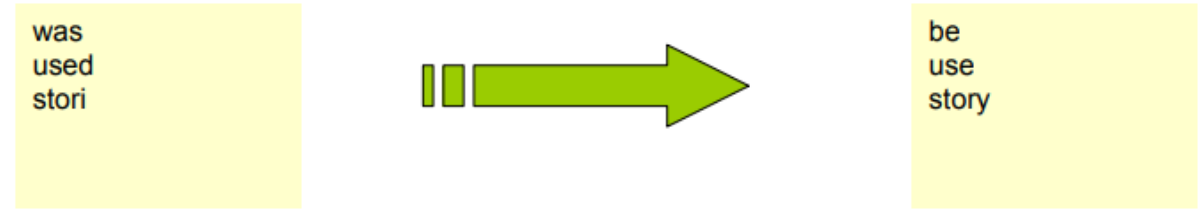
Tahap mencari akar kata hasil filtering.



[Hasil Filter] [Hasil *Stemming*]

4. Tagging

Tahap mencari bentuk awal dari kata lampau hasil stemming.



[Hasil *Stemming*] [Hasil *tangging*]

5. Analyzing

Tahap penentuan seberapa jauh keterhubungan antar katakata antar dokumen yang ada.

Referensi :

https://garudacyber.co.id/artikel/1254-apa-itu-text-mining

https://slideplayer.info/slide/12144111/#:~:text=6%20Text%20Mining%20Perbedaan%20mendasar,dapat%20diubah%20menjadi%20lebih%20terstruktur

https://slideplayer.info/slide/15996476/

https://dspace.uii.ac.id/bitstream/handle/123456789/10239/Tesis%20Rona%20Neysa%20Dewi%2012917229%20Scan.pdf?sequence=1&isAllowed=y

http://tessy.lecturer.pens.ac.id/kuliah/dm/6Text%20Mining.pdf

https://docplayer.info/64911560-Bab-2-landasan-teori-2-1-pengertian-text-mining.html

Nama : BHIJANTA WYASA WM
NIM : 202420019
Kelas : MTI 23

Teks Mining

Silahkan cari tutorial lain yang membahas tutorial teks mining, lalu buat ringkasannya dan tulis dalam format ms word dan dikumpulkan sebelum batas waktu pengumpulan berakhir.

Jawaban,

Case Folding

Case Folding adalah tahap untuk konversi text menjadi suatu bentuk yang standar. Pada tahap ini biasanya dipilih lowercase untuk membuat huruf kapital menjadi lowercase.

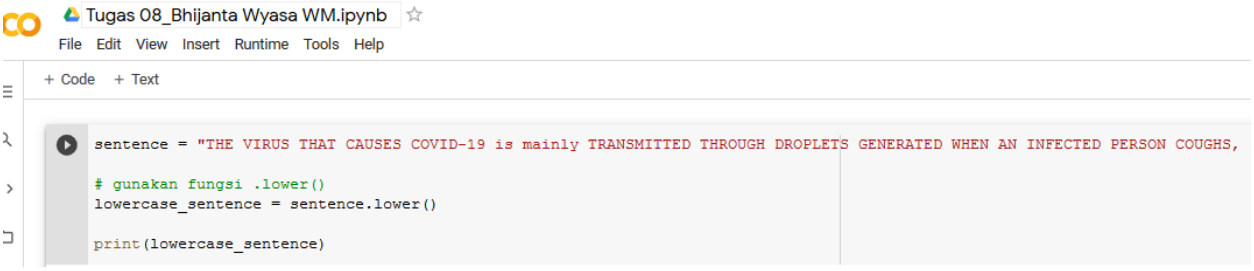
Pada kesempatan ini saya akan mengambil contoh beberapa kalimat secara acak masing – masing akan saya bedakan apakah itu huruf besar atau kecil, adapun samplenya sebagai berikut :

THE VIRUS THAT CAUSES COVID-19 is mainly TRANSMITTED THROUGH DROPLETS GENERATED WHEN AN INFECTED PERSON COUGHS, SNEEZES, OR EXHALES. These droplETS ARE TOO HEAVY TO HANG IN THE air, and QUICKLY FALL ON FLOORS OR SURFACES. You can be infected by breathing in the virus if you are Within Close Proximity Of Someone Who Has COVID-19, Or By Touching A Contaminated surface and then your eyes, nose or mouth.

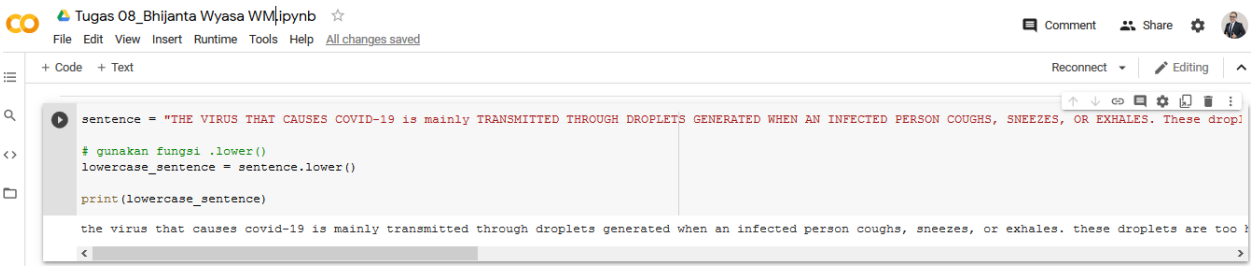
Dari data diatas kita gunakan *colab.research.google.com* untuk memproses sesuai dengan text mining case folding, adapun sintax codingnya sebagai berikut :

```
sentence = " THE VIRUS THAT CAUSES COVID-19 is mainly TRANSMITTED THROUGH DROPLETS GENERATED WHEN AN INFECTED PERSON COUGHS, SNEEZES, OR EXHALES. These droplETS ARE TOO HEAVY TO HANG IN THE air, and QUICKLY FALL ON FLOORS OR SURFACES. You can be infected by breathing in the virus if you are Within Close Proximity Of Someone Who Has COVID-19, Or By Touching A Contaminated surface and then your eyes, nose or mouth."

# gunakan fungsi .lower()

lowercase_sentence = sentence.lower()

print(lowercase_sentence)
```

Nama : BHIJANTA WYASA WM
NIM : 202420019
Kelas : MTI 23



Setelah itu RUN sintax coding diatas dengan menekan tombol RUN  adapun hasilnya sebagai berikut :



Didapatkan semua hasil untuk hurufnya menjadi huruf kecil semua, sehingga secara fungsi sintax codinya berjalan dengan baik.

Nama : Cornelia Tri Wahyuni
NIM : 202420044
Kelas : MTI Reguler A
MK : Advanced Database

Tugas 8

Dalam text mining dikenal istilah Text Preprocessing. Text Preprocessing adalah tahapan dimana kita melakukan seleksi data agar data yang akan kita olah menjadi lebih terstruktur. Disini dijelaskan bagiamana melakukan proses Text Preprocessing menggunakan Python dengan Library NLTK. Tahapan proses text preprocessing adalah Case Folding, Tokenization dan Filtering, Stopword Removal, Stemming.

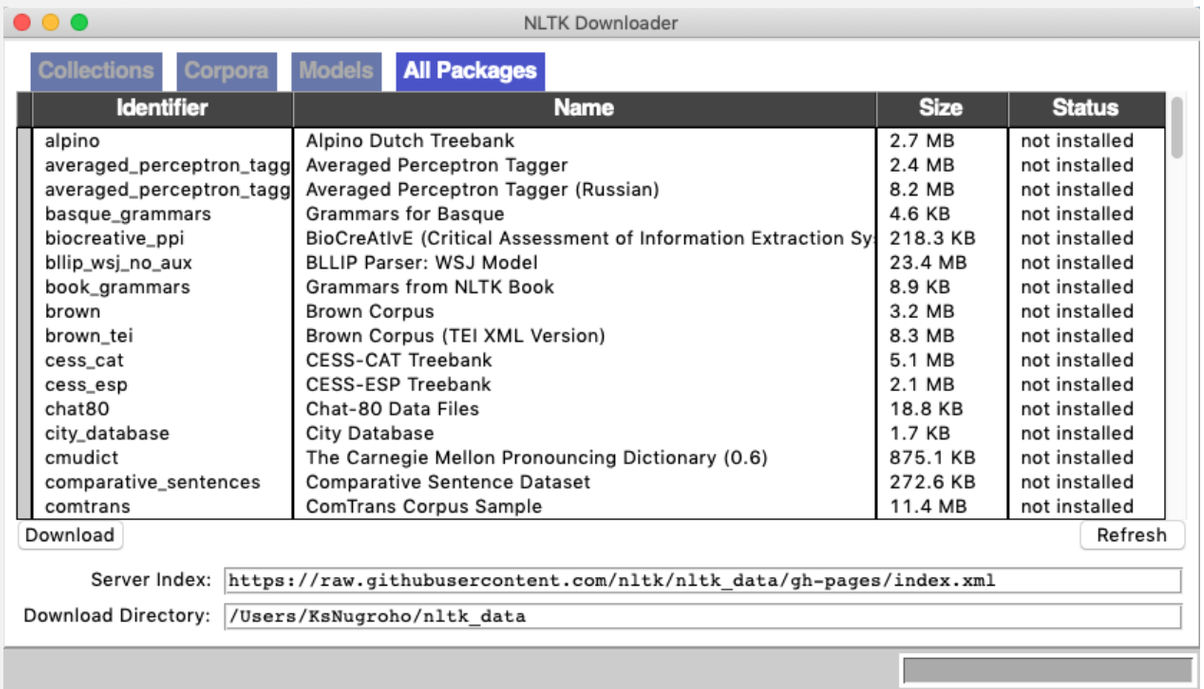
Natural Language Toolkit (NLTK)

Natural Language Toolkit atau disingkat NLTK, adalah *libray* python untuk bekerja dengan permodelan teks. NLTK menyediakan alat yang baik mempersiapkan teks sebelum digunakan pada *machine learning* atau algoritma *deep learnin*g. Cara termudah untuk menginstall NLTK adalah menggunakan “pip” pada *command line*/*terminal*.

```
pip install nltk
```

- Langkah pertama yang perlu anda lakukan setelah menginstall NLTK adalah mengunduh paket NLTK.

```
import nltk
nltk.download()
```



NLTK

- **Python Sastrawi (Stemming Bahasa Indonesia)**

  Python Sastrawi adalah pengembangan dari proyek PHP Sastrawi Python Sastrawi merupakan library sederhana yang dapat mengubah kata berimbuhan bahasa Indonesia menjadi bentuk dasarnya. Sastrawi juga dapat diinstal melalui “pip”.

```
pip install Sastrawi
```

  Penggunaan Python Sastrawi baris kode seperti berikut :

```
# import StemmerFactory class
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory# create stemmer
factory = StemmerFactory()
stemmer = factory.create_stemmer()# stemming process
sentence = 'Perekonomian Indonesia sedang dalam pertumbuhan yang membanggakan'output   = stemmer.stem(sentence)print(output)
# ekonomi indonesia sedang dalam tumbuh yang banggaprint(stemmer.stem('Mereka meniru-nirukannya'))
# mereka tiru
```

Bagian atau tahapan dokumentasi lengkap dari Python Sastrawi sebagai berikut :

1. **Case folding**

Case folding adalah salah satu bentuk *text preprocessing* yang paling sederhana dan efektif meskipun sering diabaikan. Tujuan dari *case folding* untuk mengubah semua huruf dalam dokumen menjadi huruf kecil. Hanya huruf ‘a’ sampai ‘z’ yang diterima. Karakter selain huruf dihilangkan dan dianggap *delimiter*. Pada tahap ini tidak menggunakan *external library* apapun, kita bisa memanfaatkan modul yang tersedia di python. Ada beberapa cara yang dapat digunakan dalam tahap *case folding*, anda dapat menggunakan beberapa atau menggunakan semuanya, tergantung pada tugas yang diberikan.

- Mengubah text menjadi lowercase

  Salah satu contoh pentingnya penggunaan *lower case* adalah untuk mesin pencarian. Bayangkan anda sedang mencari dokumen yang mengandung “indonesia” namun tidak ada hasil yang muncul karena “indonesia” di indeks sebagai “INDONESIA”. Contoh dibawah menunjukan bagaimana python mengubah teks menjadi *lowercase* :

```
kalimat = "Berikut ini adalah 5 negara dengan pendidikan terbaik di dunia adalah Korea Selatan, Jepang, Singapura, Hong Kong, dan Finlandia."lower_case = kalimat.lower()
print(lower_case)# output

# berikut ini adalah 5 negara dengan pendidikan terbaik di dunia adalah korea selatan, jepang, singapura, hong kong, dan finlandia.
```

- Menghapus angka

Hapuslah angka jika tidak relevan dengan apa yang akan anda analisa, contohnya seperti nomor rumah, nomor telepon, dll. *Regular expression* (*regex*) dapat digunakan untuk menghapus karakter angka. Python memiliki modulre untuk melakukan hal – hal yang berkaitan dengan *regex*. Contoh dibawah menunjukan bagaimana python menghapus angka dalam sebuah kalimat :

```
import re # impor modul regular expressionkalimat = "Berikut
ini adalah 5 negara dengan pendidikan terbaik di dunia adalah
Korea Selatan, Jepang, Singapura, Hong Kong, dan Finlandia."
hasil = re.sub(r"\d+", "", kalimat)

print(hasil)# ouput
# Berikut ini adalah negara dengan pendidikan terbaik di dunia
adalah Korea Selatan, Jepang, Singapura, Hong Kong, dan
Finlandia.
```

- Menghapus tanda baca

Sama halnya dengan angka, tanda baca dalam kalimat tidak memiliki pengaruh pada *text preprocessing*. Menghapus tanda baca seperti [!”#$%&’()*+,-./:;<=>?@[\]^_`{|}~] dapat dilakukan di pyhton seperti dibawah ini :

```
kalimat = "Ini &adalah [contoh] kalimat? {dengan} tanda.
baca?!!"
hasil =
kalimat.translate(str.maketrans("","",string.punctuation))
print(hasil)# output
# Ini adalah contoh kalimat dengan tanda baca
```

- Menghapus whitepace (karakter kosong)

Untuk menghapus spasi di awal dan akhir, anda dapat menggunakan fungsi `strip()`pada pyhton. Perhatikan kode dibawah ini :

```
kalimat = " \t ini kalimat contoh\t "
hasil = kalimat.strip()
print(hasil)# output
# ini kalimat contoh
```

2. Tokenizing

Tokenizing adalah proses pemisahan teks menjadi potongan-potongan yang disebut sebagai token untuk kemudian di analisa. Kata, angka, simbol, tanda baca dan entitas penting lainnya dapat dianggap sebagai token. Didalam NLP, token diartikan sebagai “kata” meskipun *tokenize* juga dapat dilakukan pada paragraf maupun kalimat. Fungsi `split()`pada pyhton dapat digunakan untuk memisahkan teks. seperti contoh dibawah ini :

```
kalimat = "rumah idaman adalah rumah yang bersih"
pisah = kalimat.split()
```

```
print(pisah)# output
# ['rumah', 'idaman', 'adalah', 'rumah', 'yang', 'bersih']
```

Sayangnya, tahap *tokenizing* tidak sesederhana itu. Dari *output* kode diatas kita akan mengolah kata “rumah” dan “rumah” sebagai 2 entitas yang berbeda. Permasalahan ini tentunya akan mempengaruhi hasil dari analisa teks itu sendiri.

Untuk mengatasi masalah ini kita dapat menggunakan modul dari NLTK yang sudah diinstal sebelumnya.

- Tokenizing kata

  Sebuah kalimat atau data dapat dipisah menjadi kata-kata dengan kelas word_tokenize() pada modul NLTK.

```
# impor word_tokenize dari modul nltk
from nltk.tokenize import word_tokenize

kalimat = "Andi kerap melakukan transaksi rutin secara daring
atau online."

tokens = nltk.tokenize.word_tokenize(kalimat)
print(tokens)# ouput
# ['Andi', 'kerap', 'melakukan', 'transaksi', 'rutin',
'secara', 'daring', 'atau', 'online', '.']
```

  Dari *output* kode diatas terdapat kemunculan tanda baca titik(.) dan koma (,) serta token “Andi” yang masih menggunakan huruf besar pada awal kata. Hal tersebut nantinya dapat menggangu proses perhitungan dalam penerapan algoritma. Jadi, sebaiknya teks telah melewati tahap *case folding* sebelum di *tokenize* agar menghasilkan hasil yang lebih konsisten.

  Kita dapat menambahkan fungsi *case folding* untuk menghilangkan tanda baca dan mengubah *teks* ke dalam bentuk *lowercase*.

```
kalimat =
kalimat.translate(str.maketrans('','',string.punctuation)).low
er()# output
# ['andi', 'kerap', 'melakukan', 'transaksi', 'rutin',
'secara', 'daring', 'atau', 'online']
```

  Kita juga bisa mendapatkan informasi frekuensi kemunculan setiap token dengan kelas FreqDist() yang sudah tersedia pada modul NLTK.

```
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDistkalimat = "Andi kerap
melakukan transaksi rutin secara daring atau online. Menurut
Andi belanja online lebih praktis & murah."
kalimat =
kalimat.translate(str.maketrans('','',string.punctuation)).low
er()

tokens = nltk.tokenize.word_tokenize(kalimat)
```

```
kemunculan = nltk.FreqDist(tokens)
print(kemunculan.most_common())# output
# [('andi', 2), ('online', 2), ('kerap', 1), ('melakukan', 1),
('transaksi', 1), ('rutin', 1), ('secara', 1), ('daring', 1),
('atau', 1), ('menurut', 1), ('belanja', 1), ('lebih', 1),
('praktis', 1), ('murah', 1)]
```

- Tokenizing kalimat

Prinsip yang sama dapat diterapkan untuk memisahkan kalimat pada paragraf. Anda dapat menggunkan kelas sent_tokenize() pada modul NLTK kalimat pada contoh seperti dibawah ini :

```
# impor sent_tokenize dari modul nltk
from nltk.tokenize import sent_tokenizekalimat = "Andi kerap
melakukan transaksi rutin secara daring atau online. Menurut
Andi belanja online lebih praktis & murah."

tokens = nltk.tokenize.sent_tokenize(kalimat)
print(tokens)# ouput
# ['Andi kerap melakukan transaksi rutin secara daring atau
online.', 'Menurut Andi belanja online lebih praktis &
murah.']
```

3. Filtering (Stopword Removal)

Filtering adalah tahap mengambil kata-kata penting dari hasil token dengan menggunakan algoritma *stoplist* (membuang kata kurang penting) atau *wordlist* (menyimpan kata penting).

Stopword adalah kata umum yang biasanya muncul dalam jumlah besar dan dianggap tidak memiliki makna. Contoh *stopword* dalam bahasa Indonesia adalah “yang”, “dan”, “di”, “dari”, dll. Makna di balik penggunaan *stopword* yaitu dengan menghapus kata-kata yang memiliki informasi rendah dari sebuah teks, kita dapat fokus pada kata-kata penting sebagai gantinya.

Contoh penggunaan *filtering* dapat kita temukan pada konteks mesin pencarian. Jika permintaan pencarian anda adalah “apa itu pengertian manajemen?” tentunya anda ingin sistem pencarian fokus pada memunculkan dokumen dengan topik tentang “pengertian manajemen” di atas dokumen dengan topik “apa itu”. Hal ini dapat dilakukan dengan mencegah kata dari daftar *stopword* dianalisa.

- Filtering dengan NLTK

```
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords

kalimat = "Andi kerap melakukan transaksi rutin secara daring
atau online. Menurut Andi belanja online lebih praktis &
murah."
kalimat =
kalimat.translate(str.maketrans('','',string.punctuation)).low
```

```
er()

tokens = word_tokenize(kalimat)
listStopword =  set(stopwords.words('indonesian'))

removed = []
for t in tokens:
    if t not in listStopword:
        removed.append(t)

print(removed)# ouput
# ['andi', 'kerap', 'transaksi', 'rutin', 'daring', 'online',
'andi', 'belanja', 'online', 'praktis', 'murah']
```

- Filtering dengan sastrawi

  Selain untuk *stemming,* library Sastrawi juga mendukung proses *filtering*. Kita dapat menggunakan stopWordRemoverFactory dari modul sastrawi. Untuk melihat daftar *stopword* yang telah didefinisikan dalam library Sastrawi dapat menggunakan kode berikut :

```
from Sastrawi.StopWordRemover.StopWordRemoverFactory import
StopWordRemoverFactoryfactory = StopWordRemoverFactory()
stopwords = factory.get_stop_words()
print(stopwords)
```

  Kode diatas akan menampikan *stopword* yang tersedia di library Sastrawi. Prosos *filtering* pada Sastrawi dapat dilihat pada baris kode dibawah :

```
from Sastrawi.StopWordRemover.StopWordRemoverFactory import
StopWordRemoverFactory
from nltk.tokenize import word_tokenizefactory =
StopWordRemoverFactory()
stopword = factory.create_stop_word_remover()kalimat = "Andi
kerap melakukan transaksi rutin secara daring atau online.
Menurut Andi belanja online lebih praktis & murah."
kalimat =
kalimat.translate(str.maketrans('','',string.punctuation)).low
er()stop = stopword.remove(kalimat)
tokens = nltk.tokenize.word_tokenize(stop)
print(tokens)# output
# ['andi', 'kerap', 'transaksi', 'rutin', 'daring', 'online',
'andi', 'belanja', 'online', 'praktis', 'murah']
```

  Kita dapat menambah atau mengurangi kata pada daftar *stopword* sesuai dengan kebutuhan analisa.

4. Stemming

Stemming adalah proses menghilangkan infleksi kata ke bentuk dasarnya, namun bentuk dasar tersebut tidak berarti sama dengan akar kata (*root word*). Misalnya kata “mendengarkan”, “dengarkan”, “didengarkan” akan ditransformasi menjadi kata “dengar”. Idenya adalah ketika anda mencari dokumen “cara membuka lemari”, anda juga ingin melihat dokumen yang menyebutkan “cara terbuka lemari” atau “cara dibuka

lemari” meskipun terdengar tidak enak. Tentunya anda ingin mencocokan semua variasi kata untuk memunculkan dokumen yang paling relevan.

- Stemming dengan NLTK (bahasa inggris)

  Ada banyak algoritma yang digunakan untuk *stemming*. Salah satu algoritma yang tertua dan paling populer adalah algoritma porter. Algoritma ini tersedia dalam modul NLTK melalui kelas `PorterStemmer()`.

```
from nltk.stem import PorterStemmer

ps = PorterStemmer()

kata = ["program", "programs", "programer", "programing",
"programers"]

for k in kata:
    print(k, " : ", ps.stem(k))# ouput
# program  :  program
programs  :  program
programer  :  program
programing  :  program
programers  :  program
```

- Stemming bahasa indonesia menggunakan Python Sastrawi

  Proses *stemming* antara satu bahasa dengan bahasa yang lain tentu berbeda. Contohnya pada teks berbahasa inggris, proses yang diperlukan hanya proses menghilangkan surfiks. Sedangkan pada teks berbahasa Indonesia semua kata imbuhan baik itu sufiks dan prefiks juga dihilangkan.

```
from Sastrawi.Stemmer.StemmerFactory import
StemmerFactoryfactory = StemmerFactory()
stemmer = factory.create_stemmer()

kalimat = "Andi kerap melakukan transaksi rutin secara daring
atau online. Menurut Andi belanja online lebih praktis &
murah."hasil = stemmer.stem(kalimat)print(hasil)# ouput
# andi kerap laku transaksi rutin cara daring atau online
turut andi belanja online lebih praktis murah
```

  Tidak ada aturan pasti yang membahas setiap tahapan pada *text preprocessing*. Tentu saja untuk memastikan hasil yang lebih baik dan konsisten semua tahapan harus dilakukan. Untuk memberi gambaran tentang apa yang minimal seharusnya dilakukan, adalah harus dilakukan, sebaiknya dilakukan, dan tergantung tugas.

  - Harus dilakukan meliputi *case folding* (dapat tergantung tugas dalam beberapa kasus)
  - Sebaiknya dilakukan meliputi normalisasi sederhana — misalnya menstandarkan kata yang hampir sama

- Tergantung tugas meliputi normalisasi tingkat lanjut — misalnya mengatasi kata yang tidak biasa, s*topword removal* dan *stemming*

Jadi, untuk mengatasi tugas apapun minimal anda harus melakukan *case folding*. Selanjutnya dapat ditambahkan beberapa normalisasi dasar untuk mendapatkan hasil yang lebih konsisten dan secara bertahap menambahkan tahapan yang lainnya sesuai yang anda inginkan.

Nama : Cynthia Anisa Agatha

NIM : 202420022

Teks Mining adalah sebuah proses pengolahan teks yang bertujuan untuk mengubah struktur atau bentuk teks yang asalnya sulit dimengerti komputer hingga akhirnya mudah dimengerti dan diolah lebih lanjut. Terdapat empat tahap proses pokok dalam *text mining*, yaitu pemrosesan awal terhadap teks (*text preprocessing*), transformasi teks (*text transformation*), pemilihan fitur (*feature selection*), dan penemuan pola (*pattern discovery*) (Eko, 2011).

1. *Text Preprocessing*
   Tahap ini melakukan analisis semantik (kebenaran arti) dan sintaktik (kebenaran susunan) terhadap teks. Tujuan dari pemrosesan awal adalah untuk mempersiapkan teks menjadi data yang akan mengalami pengolahan lebih lanjut. Operasi yang dapat dilakukan pada tahap ini meliputi *part-of-speech* (PoS) *tagging*, menghasilkan *parse tree* untuk tiap-tiap kalimat, dan pembersihan teks.
2. *Text Transformation*
   Transformasi teks atau pembentukan atribut mengacu pada proses untuk mendapatkan representasi dokumen yang diharapkan. Pendekatan representasi dokumen yang lazim digunakan oleh model "*bag of words*" dan model ruang vector (*vector space model*). Transformasi teks sekaligus juga melakukan pengubahan kata-kata ke bentuk dasarnya dan pengurangan dimensi kata di dalam dokumen. Tindakan ini diwujudkan dengan menerapkan stemming dan menghapus stop words.
3. *Feature Selection*
   Pemilihan fitur (kata) merupakan tahap lanjut dari pengurangan dimensi pada proses transformasi teks. Walaupun tahap sebelumnya sudah melakukan penghapusan katakata yang tidak deskriptif (*stopwords*), namun tidak semua kata-kata di dalam dokumen memiliki arti penting. Oleh karena itu, untuk mengurangi dimensi, pemilihan hanya dilakukan terhadap kata-kata yang relevan yang benar-benar merepresentasikan isi dari suatu dokumen. Ide dasar dari pemilihan fitur adalah menghapus kata-kata yang kemunculannya di suatu dokumen terlalu sedikit atau terlalu banyak. Algoritma yang digunakan pada *text mining*, biasanya tidak hanya melakukan perhitungan pada dokumen saja, tetapi juga pada *feature*.
4. *Pattern Discovery*
   Pattern discovery merupakan tahap penting untuk menemukan pola atau pengetahuan (*knowledge*) dari keseluruhan teks. Tindakan yang lazim dilakukan pada tahap ini adalah operasi *text mining*, dan biasanya menggunakan teknik-teknik *data mining*. Dalam penemuan pola ini, proses *text mining* dikombinasikan dengan proses-proses *data mining*. Masukan awal dari proses text mining adalah suatu data teks dan menghasilkan keluaran berupa pola sebagai hasil interpretasi atau evaluasi. Apabila hasil keluaran dari penemuan pola belum sesuai untuk aplikasi, dilanjutkan evaluasi dengan melakukan iterasi ke satu atau beberapa tahap sebelumnya. Sebaliknya, hasil interpretasi merupakan tahap akhir dari proses text mining dan akan disajikan ke pengguna dalam bentuk visual (Eko, 2011).

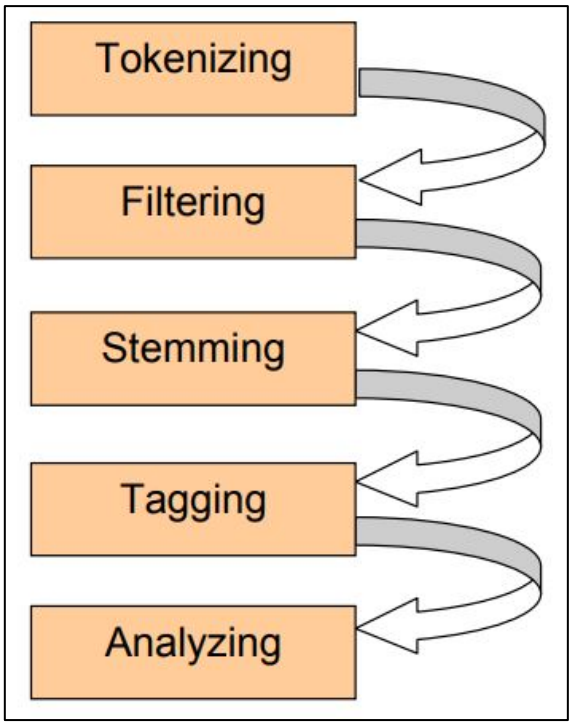
Nama : Efrik Kartono Ahsa
MK : ADVACED DATABASE
NIM : 202420030

DEFINISI TEXT MINING

Text Mining adalah salah satu bidang khusus dalam data mining yang memiliki definisi menambang data berupa teks dimana sumber data biasanya didapatkan dari dokumen dan tujuannya adalah mencari kata-kata yang dapat mewakili isi dari dokumen sehingga dapat dilakukan analisa keterhubungan antar dokumen.

TAHAPAN DALAM TEXT MINING

Tahapan yang dilakukan secara umum adalah :



1. **Tokenizing**
   Tahap *Tokenizing* adalah tahap pemotongan string input berdasarkan tiap kata yang menyusunnya.
   Contoh dari tahap ini adalah sebagai berikut:

Teks Input	Hasil Token
New London Spicer suggested that the system tracks	New London Spicer suggested that the system track

Nama : Efrik Kartono Ahsa
MK : ADVACED DATABASE
NIM : 202420030

2. Filtering

Tahap *Filtering* adalah tahap mengambil kata-kata penting dari hasil token. Bisa menggunakan algoritma stoplist (membuang kata kurang penting) atau *wordlist* (menyimpan kata penting). *Stoplist/stopword* adalah kata-kata yang tidak deskriptif yang dapat dibuang dalam pendekatan *bag-of-words.*
Selanjutnya pada tahapan *Filtering :*

Hasil Token	Hasil Filter
New London Spicer suggested that the system tracks	New London Spicer suggested that system tracks

3. Stemming

Tahapan *Stemming* adalah tahap dimana mencari root kata dari tiap kata hasil dari *filtering*.
Selanjutnya tahap *Stemming* :

Hasil Token	Hasil Filter
New London Spicer suggested that system tracks	New London Spicer suggestion that system track

4. Tagging

Tahap *Tagging* adalah tahap untuk mencari bentuk awal/root dari tiap kata lampau atau kata hasil *Stemming*.

Selanjutnya tahap *Tagging* :

Hasil Token	Hasil Filter
New London Spicer	New London Spicer

suggested that system tracks	suggestion that system track

5. Analyzing

Tahap *Analyzing* merupakan tahap penentuan seberapa jauh keterhubungan antar kata-kata antar dokumen yang ada.

GATOT ARIFIANTO
NIM : 202420029
KELAS MTI 23 A

TUGAS 08
ADVANCED DATABASE TEKS MINING

TEKS MINING

Silahkan cari tutorial lain yang membahas tutorial teks mining, lalu buat ringkasannya dan ditulis di ms.word dan dikumpulkan sebelum batas waktu pengumpulan berakhir.

tutorial :

Salah satu keunggulan python adalah mendukung banyak *open-source library*. Ada banyak *library* python yang dapat digunakan untuk melakukan dan mengimplementasikan masalah dalam NLP.

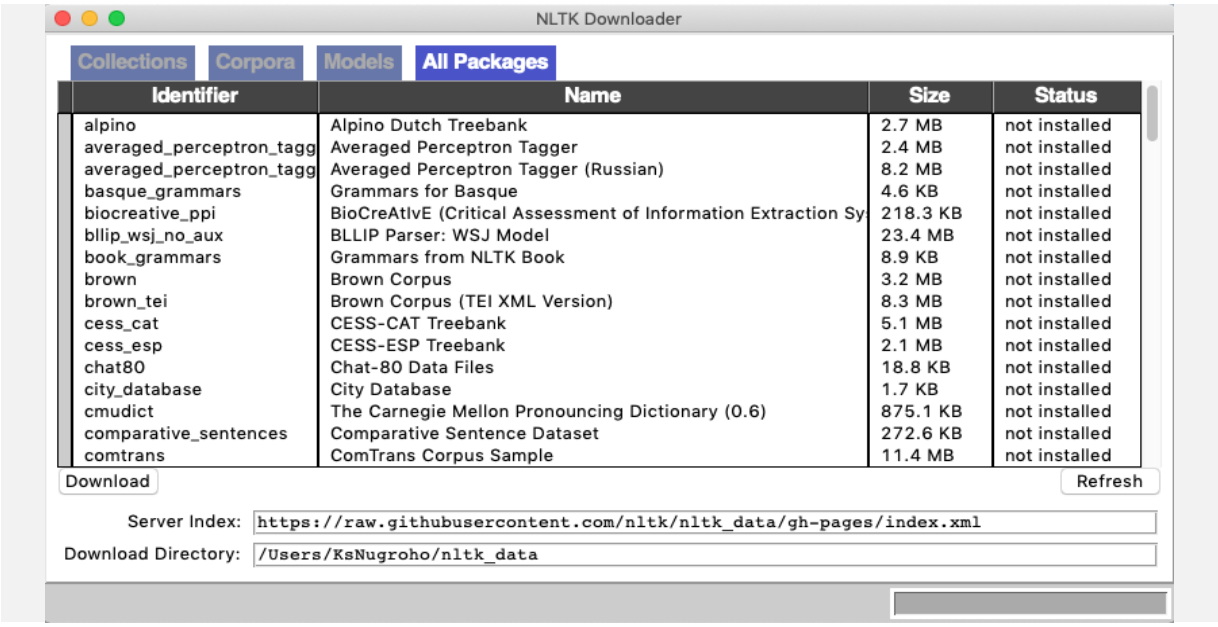
Natural Language Toolkit (NLTK)

Natural Language Toolkit atau disingkat NLTK, adalah *libray* python untuk bekerja dengan permodelan teks. NLTK menyediakan alat yang baik mempersiapkan teks sebelum digunakan pada *machine learning* atau algoritma *deep learning*. Cara termudah untuk menginstall NLTK adalah menggunakan "pip" pada *command line/terminal*.

```
pip install nltk
```

Langkah pertama yang perlu anda lakukan setelah menginstall NLTK adalah mengunduh paket NLTK.

```
Import nltk
nltk.download()
```



NLTK

Python Sastrawi

Python Sastrawi adalah pengembangan dari proyek PHP Sastrawi. Python Sastrawi merupakan library sederhana yang dapat mengubah kata berimbuhan bahasa Indonesia menjadi bentuk dasarnya. Sastrawi juga dapat diinstal melalui "pip".

```
pip install Sastrawi
```

Penggunaan Python Sastrawi sangat sederhana seperti baris kode dibawah ini :

```
# import StemmerFactory class
from Sastrawi.Stemmer.StemmerFactory import
StemmerFactory

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

# stemming process
sentence = 'Perekonomian Indonesia sedang dalam pertumbuhan yang membanggakan'

output   = stemmer.stem(sentence)

print(output)
# ekonomi indonesia sedang dalam tumbuh yang bangga

print(stemmer.stem('Mereka meniru-nirukannya'))
# mereka tiru
```

1. Case folding

Case folding adalah salah satu bentuk *text preprocessing* yang paling sederhana dan efektif meskipun sering diabaikan. Tujuan dari *case folding* untuk mengubah semua huruf dalam dokumen menjadi huruf kecil. Hanya huruf 'a' sampai 'z' yang diterima. Karakter selain huruf dihilangkan dan dianggap *delimiter*. Pada tahap ini tidak menggunakan *external library* apapun, kita bisa memanfaatkan modul yang tersedia di python.

Ada beberapa cara yang dapat digunakan dalam tahap *case folding*, anda dapat menggunakan beberapa atau menggunakan semuanya, tergantung pada tugas yang diberikan.

Mengubah text menjadi lowercase

Salah satu contoh pentingnya penggunaan *lower case* adalah untuk mesin pencarian. Bayangkan anda sedang mencari dokumen yang mengandung "indonesia" namun tidak ada hasil yang muncul karena "indonesia" di indeks sebagai "INDONESIA". Siapa yang harus kita salahkan? U.I. *designer* yang mengatur *user interface* atau *developer* yang mengatur sistem dalam indeks pencarian?

Contoh dibawah menunjukan bagaimana python mengubah teks menjadi *lowercase* :

```
kalimat = "Berikut ini adalah 5 negara dengan pendidikan terbaik di dunia adalah Korea Selatan, Jepang, Singapura, Hong Kong, dan Finlandia."
lower_case = kalimat.lower()
print(lower_case)
# output
# berikut ini adalah 5 negara dengan pendidikan terbaik di dunia adalah korea selatan, jepang, singapura, hong kong, dan finlandia.
```

Menghapus angka

Hapuslah angka jika tidak relevan dengan apa yang akan anda analisa, contohnya seperti nomor rumah, nomor telepon, dll. *Regular expression* (*regex*) dapat digunakan untuk menghapus karakter angka. Python memiliki modulre untuk melakukan hal – hal yang berkaitan dengan *regex*.

Contoh dibawah menunjukan bagaimana python menghapus angka dalam sebuah kalimat :

```
import re # impor modul regular expression
kalimat = "Berikut ini adalah 5 negara dengan pendidikan terbaik di dunia adalah Korea Selatan, Jepang, Singapura, Hong Kong, dan Finlandia."
hasil = re.sub(r"\d+", "", kalimat)
print(hasil)

# ouput
# Berikut ini adalah negara dengan pendidikan terbaik di dunia adalah Korea Selatan, Jepang, Singapura, Hong Kong, dan Finlandia.
```

Menghapus tanda baca

Sama halnya dengan angka, tanda baca dalam kalimat tidak memiliki pengaruh pada *text preprocessing*. Menghapus tanda baca seperti [!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~] dapat dilakukan di pyhton seperti dibawah ini :

```
kalimat = "Ini &adalah [contoh] kalimat? {dengan} tanda. baca?!!"
hasil =
 kalimat.translate(str.maketrans("","",string.punctuation))
print(hasil)

# output
# Ini adalah contoh kalimat dengan tanda baca
```

Menghapus whitepace (karakter kosong)

Untuk menghapus spasi di awal dan akhir, anda dapat menggunakan fungsi strip() pada pyhton. Perhatikan kode dibawah ini :

```
kalimat = " \t ini kalimat contoh\t "
hasil = kalimat.strip()
print(hasil)

# output
# ini kalimat contoh
```

2. Tokenizing

Tokenizing adalah proses pemisahan teks menjadi potongan-potongan yang disebut sebagai token untuk kemudian di analisa. Kata, angka, simbol, tanda baca dan entitas penting lainnya dapat dianggap sebagai token. Didalam NLP, token diartikan sebagai "kata" meskipun *tokenize* juga dapat dilakukan pada paragraf maupun kalimat.

Fungsi split() pada pyhton dapat digunakan untuk memisahkan teks. Perhatikan contoh dibawah ini :

```
kalimat = "rumah idaman adalah rumah yang bersih"
pisah = kalimat.split()
print(pisah)

# output
# ['rumah', 'idaman', 'adalah', 'rumah', 'yang', 'bersih']
```

Sayangnya, tahap *tokenizing* tidak sesederhana itu. Dari *output* kode diatas kita akan mengolah kata "rumah" dan "rumah" sebagai 2 entitas yang berbeda. Permasalahan ini tentunya akan mempengaruhi hasil dari analisa teks itu sendiri.

Untuk mengatasi masalah ini kita dapat menggunakan modul dari NLTK yang sudah diinstal sebelumnya.

Tokenizing kata

Sebuah kalimat atau data dapat dipisah menjadi kata-kata dengan kelas word_tokenize() pada modul NLTK.

```
# impor word_tokenize dari modul nltk
from nltk.tokenize import word_tokenize

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online."

tokens = nltk.tokenize.word_tokenize(kalimat)
```

```
print(tokens)

# ouput
# ['Andi', 'kerap', 'melakukan', 'transaksi', 'rutin', 'secara', 'daring', 'atau', 'online', '.']
```

Dari *output* kode diatas terdapat kemunculan tanda baca titik(.) dan koma (,) serta token "Andi" yang masih menggunakan huruf besar pada awal kata. Hal tersebut nantinya dapat menggangu proses perhitungan dalam penerapan algoritma. Jadi, sebaiknya teks telah melewati tahap *case folding* sebelum di *tokenize* agar menghasilkan hasil yang lebih konsisten.

Kita dapat menambahkan fungsi *case folding* untuk menghilangkan tanda baca dan mengubah *teks* ke dalam bentuk *lowercase*.

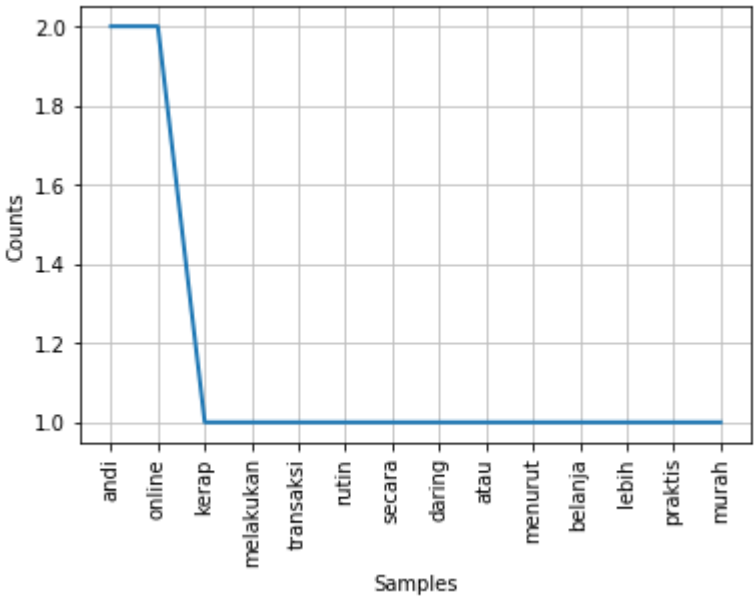
```
kalimat = kalimat.translate(str.maketrans("",string.punctuation)).lower()

# output
# ['andi', 'kerap', 'melakukan', 'transaksi', 'rutin', 'secara', 'daring', 'atau', 'online']
```

Kita juga bisa mendapatkan informasi frekuensi kemunculan setiap token dengan kelas FreqDist() yang sudah tersedia pada modul NLTK.

```
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDist
kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online. Menurut Andi belanja online lebih praktis & murah."
kalimat = kalimat.translate(str.maketrans("",string.punctuation)).lower()
tokens = nltk.tokenize.word_tokenize(kalimat)
kemunculan = nltk.FreqDist(tokens)
print(kemunculan.most_common())


# output
# [('andi', 2), ('online', 2), ('kerap', 1), ('melakukan', 1), ('transaksi', 1), ('rutin', 1), ('secara', 1), ('daring', 1), ('atau', 1), ('menurut', 1), ('belanja', 1), ('lebih', 1), ('praktis', 1), ('murah', 1)]
```

Untuk menggambarkan ke dalam bentuk grafik, kita perlu menginstall library Matplotlib.

```
import matplotlib.pyplot as plt

kemunculan.plot(30,cumulative=False)
plt.show()
```



Grafik frekuensi kemunculan kata pada dokumen

Tokenizing kalimat

Prinsip yang sama dapat diterapkan untuk memisahkan kalimat pada paragraf. Anda dapat menggunkan kelas sent_tokenize() pada modul NLTK. Saya telah menambahkan kalimat pada contoh seperti dibawah ini :

```
# impor sent_tokenize dari modul nltk
from nltk.tokenize import sent_tokenize

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online. Menurut Andi belanja online lebih praktis & murah."

tokens = nltk.tokenize.sent_tokenize(kalimat)
print(tokens)

# ouput
# ['Andi kerap melakukan transaksi rutin secara daring atau online.', 'Menurut Andi belanja online lebih praktis & murah.']
```

3. Filtering (Stopword Removal)

Filtering adalah tahap mengambil kata-kata penting dari hasil token dengan menggunakan algoritma *stoplist* (membuang kata kurang penting) atau *wordlist* (menyimpan kata penting).

Stopword adalah kata umum yang biasanya muncul dalam jumlah besar dan dianggap tidak memiliki makna. Contoh *stopword* dalam bahasa Indonesia adalah “yang”, “dan”, “di”, “dari”, dll. Makna di balik penggunaan *stopword* yaitu dengan menghapus kata-kata yang memiliki informasi rendah dari sebuah teks, kita dapat fokus pada kata-kata penting sebagai gantinya.

Contoh penggunaan *filtering* dapat kita temukan pada konteks mesin pencarian. Jika permintaan pencarian anda adalah "apa itu pengertian manajemen?" tentunya anda ingin sistem pencarian fokus pada memunculkan dokumen dengan topik tentang "pengertian manajemen" di atas dokumen dengan topik "apa itu". Hal ini dapat dilakukan dengan mencegah kata dari daftar *stopword* dianalisa.

Filtering dengan NLTK

```
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online. Menurut Andi belanja online lebih praktis & murah."
kalimat = kalimat.translate(str.maketrans(",",string.punctuation)).lower()

tokens = word_tokenize(kalimat)
listStopword =  set(stopwords.words('indonesian'))

removed = []
for t in tokens:
    if t not in listStopword:
        removed.append(t)

print(removed)

# ouput
# ['andi', 'kerap', 'transaksi', 'rutin', 'daring', 'online', 'andi', 'belanja', 'online', 'praktis', 'murah']
```

Filtering dengan sastrawi

Selain untuk *stemming,* library Sastrawi juga mendukung proses *filtering*. Kita dapat menggunakan **stopWordRemoverFactory** dari modul sastrawi.

Untuk melihat daftar *stopword* yang telah didefinisikan dalam library Sastrawi dapat menggunakan kode berikut :

```
from Sastrawi.StopWordRemover.StopWordRemoverFactory
import StopWordRemoverFactory

factory = StopWordRemoverFactory()
stopwords = factory.get_stop_words()
print(stopwords)
```

Kode diatas akan menampikan *stopword* yang tersedia di library Sastrawi. Prosos *filtering* pada Sastrawi dapat dilihat pada baris kode dibawah :

```
from Sastrawi.StopWordRemover.StopWordRemoverFactory
```

```
import StopWordRemoverFactory
from nltk.tokenize import word_tokenize

factory = StopWordRemoverFactory()
stopword = factory.create_stop_word_remover()

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online. Menurut Andi belanja online lebih praktis & murah."
kalimat = kalimat.translate(str.maketrans(",",string.punctuation)).lower()

stop = stopword.remove(kalimat)
tokens = nltk.tokenize.word_tokenize(stop)
print(tokens)

# output
# ['andi', 'kerap', 'transaksi', 'rutin', 'daring', 'online', 'andi', 'belanja', 'online', 'praktis', 'murah']
```

Kita dapat menambah atau mengurangi kata pada daftar *stopword* sesuai dengan kebutuhan analisa. Pada dasarnya daftar *stopword* pada library Sastrawi tersimpan di dalam list yang anda lihat disini. Jadi sebenarnya kita tinggal mengubah daftar pada list tersebut. Tetapi hal tersebut bisa menjadi permasalahan apabila pada suatu kasus kita diharuskan menambahkan *stopword* secara dinamis.

Library Sastrawi dapat mengatasi permasalahan tersebut, perhatikan kode dibawah ini :

```
from Sastrawi.StopWordRemover.StopWordRemoverFactory
import StopWordRemoverFactory, StopWordRemover, ArrayDictionary
from nltk.tokenize import word_tokenize

stop_factory = StopWordRemoverFactory().get_stop_words() #load defaul stopword
more_stopword = ['daring', 'online'] #menambahkan stopword

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online. Menurut Andi belanja online lebih praktis & murah."
kalimat = kalimat.translate(str.maketrans(",",string.punctuation)).lower()

data = stop_factory + more_stopword #menggabungkan stopword

dictionary = ArrayDictionary(data)
str = StopWordRemover(dictionary)
tokens = nltk.tokenize.word_tokenize(str.remove(kalimat))

print(tokens)

# output
# ['andi', 'kerap', 'transaksi', 'rutin', 'andi', 'belanja', 'praktis', 'murah']
```

Menurut Jim Geovedi pada artikelnya menjelaskan bahwa penyesuaian daftar *stopwords* perlu dilakukan setiap pertama kali melakukan proyek analisa. Memang bukan sesuatu yang melelahkan, tapi jika tidak dilakukan maka ini akan dapat mengakibatkan salah interpretasi terhadap data.

4. Stemming

Stemming adalah proses menghilangkan infleksi kata ke bentuk dasarnya, namun bentuk dasar tersebut tidak berarti sama dengan akar kata (*root word*). Misalnya kata "mendengarkan", "dengarkan", "didengarkan" akan ditransformasi menjadi kata "dengar".

Idenya adalah ketika anda mencari dokumen "cara membuka lemari", anda juga ingin melihat dokumen yang menyebutkan "cara terbuka lemari" atau "cara dibuka lemari" meskipun terdengar tidak enak. Tentunya anda ingin mencocokan semua variasi kata untuk memunculkan dokumen yang paling relevan.

Stemming dengan NLTK (bahasa inggris)

Ada banyak algoritma yang digunakan untuk *stemming*. Salah satu algoritma yang tertua dan paling populer adalah algoritma Porter. Algoritma ini tersedia dalam modul NLTK melalui kelas PorterStemmer().

```
from nltk.stem import PorterStemmer
ps = PorterStemmer()
kata = ["program", "programs", "programer", "programing", "programers"]

for k in kata:
    print (k, " : ", ps.stem(k))

# ouput
# program  :  program
programs  :  program
programer  :  program
programing  :  program
programers  :  program
```

Selain Porter, NLTK juga mendukung algoritma Lancester, WordNet Lemmatizer, dan SnowBall. Sayangnya proses *stemming* Bahasa Indonesia pada modul NLTK belum didukung.

Stemming bahasa indonesia menggunakan Python Sastrawi

Proses *stemming* antara satu bahasa dengan bahasa yang lain tentu berbeda. Contohnya pada teks berbahasa inggris, proses yang diperlukan hanya proses menghilangkan sufiks. Sedangkan pada teks berbahasa Indonesia semua kata imbuhan baik itu sufiks dan prefiks juga dihilangkan.

Untuk melakukan *stemming* bahasa Indonesia kita dapat menggunakan library Python Sastrawi yang sudah kita siapkan di awal. *Library* Sastrawi menerapkan Algoritma Nazief dan Adriani dalam melakukan *stemming* bahasa Indonesia.

```
from Sastrawi.Stemmer.StemmerFactory
import StemmerFactory

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

kalimat = "Andi kerap melakukan transaksi rutin secara daring atau online. Menurut Andi belanja online lebih praktis & murah."

hasil = stemmer.stem(kalimat)

print(hasil)

# ouput
# andi kerap laku transaksi rutin cara daring atau online turut andi belanja online lebih praktis murah
```

SELESAI