BAB 1. Ekstraksi Fitur

A. Tujuan

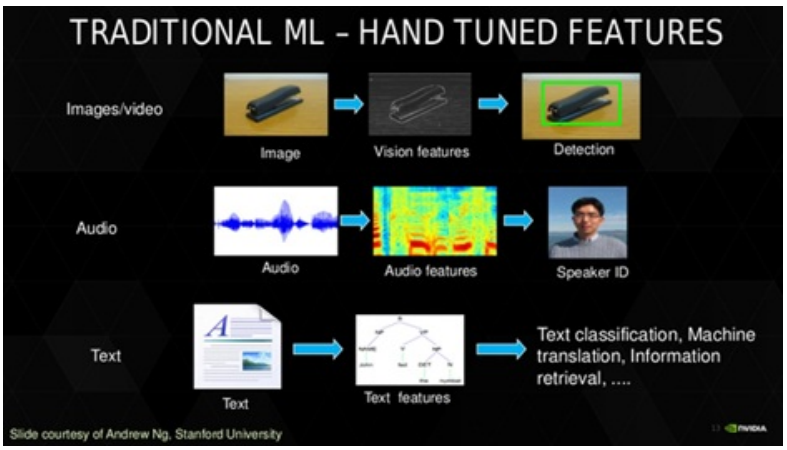
Pada materi ektraksi fitur, mahasiswa diharapkan mampu,

1. memahami konsep ekstraksi fitur pada pembelajaran mesin.
2. melakukan perhitungan manual pada ekstraksi fitur.
3. mengimplementasikan proses ektraksi fitur menggunakan bahasa pemrograman python.

B. Teori

1. Fitur (*Features*)

Dalam pembelajaran mesin (*machine learning*), agar setiap algoritma dapat melakukan pembelajaran dengan baik, diperlukan nilai-nilai yang dikenal dengan nama fitur. Fitur merupakan attribut, karakteristik, ciri, atau sesuatu yang spesial dari sebuah obyek (data) yang direpresentasikan dalam bentuk angka. Terdapat banyak cara untuk mendapatkan fitur dari sebuah data. Hal ini bergantung dengan data apa yang digunakan. Gambar 1.1 merupakan contoh dari berbagai macam fitur yang dapat dihasilkan dari berbagai macam data. Terkadang, algoritma pembelajaran mesin juga sensitif terhadap fitur yang digunakan. Beberapa algoritma akan menghasilkan model pembelajaran mesin yang baik dengan fitur tertentu. Itulah mengapa terdapat satu cabang keilmuan bernama *feature engineering*, dimana keilmuan ini berusaha untuk mendapatkan fitur yang tepat dari sebuah data untuk memaksimalkan hasil dari model pembelajaran mesin.



Gambar 1.1 Contoh ekstraksi fitur dari berbagai macam data

Jumlah fitur yang merepresentasikan data juga mempengaruhi hasil model pembelajaran mesin. Jika fitur yang digunakan hanya memiliki sedikit informasi terkait dengan data, maka model pembelajaran mesin juga tidak akan dapat melaksanakan tugasnya dengan baik. Sebaliknya, jika fitur terlalu banyak mengandung informasi atau fitur mengandung informasi yang tidak relevan, maka kondisi tersebut akan berdampat pada tingginya biaya komputasi yang dikeluarkan untuk melatih model pembelajaran mesin. Selain itu, kondisi ini dapat berujung pada kondisi *overfitting*.

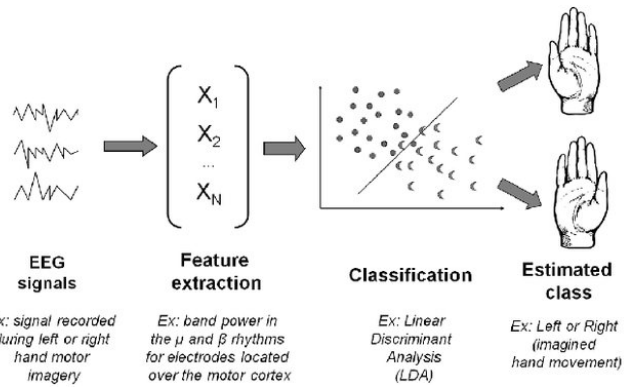
Sebagai contoh, kita akan melakukan predisik harga rumah. Terdapat beberapa karateristik (fitur) yang kita gunuakan, yaitu,

- luas tanah
- luas bangunan
- jumlah kamar tidur
- ukuran toren air

Berdasarkan karakteristik atau fitur tersebut, maka fitur yang dapat digunakan untuk memprediksi harga rumah adalah luas tanah, luas bangunan, dan jumlah kamar Fitur ukuran toren air mungkin tidak akan kita gunakan. Hal ini dikarenakan, ukuran toren air secara umum tidak menjadi parameter utama jika masyarakat hendak membeli sebuah rumah.

2. Ekstraksi Fitur (*Feature Extraction*)

Ekstraksi fitur merupakan sebuah proses pengurangan dimensi dimana data awal yang berupa sekumpulan data mentah kemudian direduksi menjadi kelompok yang lebih mudah dikelola untuk diproses oleh komputer. Tidak hanya itu, pada proses ekstraksi fitur, akan dilakukan pemilihan dan pengkombinasian variabel dari data untuk menjadi fitur, namun tetap dapat merepresentasikan data tersebut.



Gambar 1.2 Contoh ektraksi fitur pada sinyal electroencephalogram (EEG) (Lotte, 2014)

Pada Gambar 1.2 terlihat bahwa data mentah yang digunakan adalah sinyal *electroencephalograph* (EEG). Sinyal EEG merupakan salah satu bentuk informasi dari otak manusia. Sinyal ini memiliki karakteristik berupa gelombang elektromagnetik. Namun pemrosesan sinyal EEG tidak mudah kadang terdapat noise saat perekaman data sehingga dibutuhkan ekstraksi fitur yang dapat merepresentasikan sinyal EEG. Setelah fitur didapat dilakukan proses klasifikasi. Dari proses klasifikasi, dihasilkan prediksi pergerakan tangan ke kanan atau ke kiri.

3. Penyekalaan Fitur (*Feature Scaling*)

Penyekalaan fitur (*feature scaling*) adalah teknik untuk menstandarisasi fitur sehingga berada dalam rentang yang tetap. Hal ini dilakukan karena terdapat kemungkinan bahwa data diambil memiliki variabel dari berbagai macam domain. Variabel data mungkin memiliki satuan yang berbeda (misalnya meter, kaki, kilometer, dan jam) sehingga menyebabkan variabel-variabel tersebut memiliki skala yang berbeda. Perbedaan skala antar variabel dapat meningkatkan kesulitan dalam pemodelan pembelajaran mesin. Contohnya adalah nilai masukan yang besar (ratusan atau ribuan unit) dapat menghasilkan model yang cenderung terhadap nilai bobot yang besar. Model dengan nilai bobot yang besar sering kali tidak stabil, dan menganggap nilai yang lebih kecil sebagai nilai yang lebih rendah. Sehingga terdapat kemungkinan model tersebut memiliki performa yang buruk sehingga mengakibatkan *error* yang lebih tinggi.

Sebagai contoh, Jika suatu algoritma tidak menggunakan metode penyekalaan fitur, algoritma dapat menganggap nilai 3000 meter lebih besar dari 5 km. Tentu saja hal tersebut tidak benar dan dapat mengakibatkan algoritma memberikan prediksi yang salah. Oleh karena itu, penyekalaan fitur digunakan untuk membuat nilai ke besaran yang sama. Dengan demikian diharapkan penyekalaan fitur dapat mengatasi performa yang buruk dari algoritma akibat skala variabel yang berbeda.

Perbedaan skala untuk variabel ini tidak mempengaruhi semua algoritma pembelajaran mesin. Contoh algoritma yang performanya terpengaruh perbedaan skala adalah algoritma-algoritma yang melakukan penjumlahan bobot variabel input seperti *linear regression*, *logistic regression*, and *artificial neural networks*, dan *deep learning*. Begitu juga algoritma yang menggunakan jarak antar data untuk melakukan prediksi seperti K-Nearest Neighbours (KNN) dan Support Vector Machines (SVM). Selain itu, algortima berbasis model non-linier seperti *decision tree* dan *random forest* juga tidak terpengaruh proses penyekalaan. Teknik penyekalaan fitur yang sering digunakan adalah **normalisasi** dan **standarisasi**.

a) Normalisasi

Normalisasi adalah salah satu teknik penyekalaan fitur untuk menghasilkan nilai fitur yang berada dalam rentang baru, yaitu antara 0 dan 1. Persamaan 1 digunakan pada teknik normalisasi.

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \quad (1)$$

x' adalah skala variabel dalam rentang yang baru, x adalah nilai data yang akan dinormalisasi, x_{min} adalah nilai minimal dari variabel, dan x_{max} adalah nilai terbesar dari variabel.

Sebagai contoh, pada sebuah data diketahui nilai minimum adalah -10 dan nilai maksimumnya adalah 30. Jika data yang akan dinormalisasi adalah 18.8. Maka perhitungan normalisasinya adalah sebagai berikut,

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$
$$x' = \frac{18.8 - (-10)}{30 - (-10)}$$
$$x' = \frac{28.8}{40} = 0.72$$

b) Standarisasi

Model pembelajaran mesin dapat bekerja lebih baik ketika dilatih pada data yang terstandarisasi. Standarisasi melibatkan penyekalaan ulang pada distribusi nilai, sehingga didapatkan nilai tengah (*mean*) dari data yang diamati adalah 0 dan nilai standar deviasinya adalah 1. Seperti halnya pada normalisasi, standardisasi berguna untuk model pembelajaran mesin saat data yang digunakan memiliki rentang skala yang jauh berbeda. Persamaan 2 digunakan untuk mendapatkan nilai fitur yang terstandarisasi.

$$x'' = \frac{x - \mu}{\sigma} \quad (2)$$

Dimana μ adalah nilai rata-rata dari fitur dan σ adalah nilai simpangan baku dari fitur.

Sebagai contoh, jika diketahui nilai rata-rata dari sekumpulan data adalah 10.0. Kemudian simpangan bakunya adalah 5.0. Jika sebuah data x memiliki nilai 20.7. Maka dengan menggunakan nilai-nilai tersebut, kita dapat menghitung nilai standarisasinya, yaitu,

$$x'' = \frac{x - \mu}{\sigma}$$

$$x'' = \frac{20.7 - 10}{5} = 2.14$$

4. Ekstrasi Fitur Pada Data Ketegorikal

Dalam pembelajaran mesin terdapat 2 jenis data yang paling banyak digunakan yaitu tipe data kategorikal dan tipe data numerikal. Data numerik melibatkan fitur yang terdiri dari angka, seperti bilangan bulat atau bilangan desimal, sedangkan tipe data kategorikal adalah atribut yang diperlakukan sebagai simbol berbeda atau hanya nama. Data kategorikal digunakan untuk data yang tidak dapat dihitung secara kuantitatif, sehingga tidak dapat menerima operasi matematika seperti penjumlahan atau perkalian. Namun demikian, nilai-nilainya dapat dibedakan antara satu dengan lainnya. Contoh data kategorikal adalah,

- golongan darah manusia (A, AB, B, O)
- konversi nilai huruf pada mata kuliah (A, B+, B, C+, C, D, dan E)
- tingkatan juara pada ajang perlombaan (juara 1, juara 2, juara 3).

Variabel numerik dapat diubah menjadi variabel ordinal dengan membagi rentang variabel numerik menjadi beberapa bin dan menetapkan nilai ke setiap bin. Misalnya, variabel numerik antara 1 sampai 20 dapat dibagi menjadi variabel ordinal dengan 4 label dengan hubungan ordinal: 1-5, 6-10, 11-15, 16-20. Proses Ini disebut **diskritisasi**. Oleh karena itu, dapat didefinisikan bahwa,

- Variabel Nominal: Variabel terdiri dari sekumpulan nilai diskrit terbatas tanpa hubungan antar nilai atau tingkatan secara alamiah. Contohnya adalah data nama kota seperti Jakarta, Bandung, Bali.
- Variabel Ordinal: Variabel terdiri dari sekumpulan nilai diskrit yang terbatas dengan urutan peringkat antar nilai. Contohnya variabel ordinal adalah ketika terdapat urutan rendang, sedang, tinggi.

Beberapa implementasi algoritma pembelajaran mesin mengharuskan semua data harus menjadi data numerikal. Dalam artian bahwa data kategorikal harus diubah ke dalam bentuk numerikal. Ada tiga pendekatan umum yang dapat digunakan untuk melakukan konversi variabel ordinal dan variabel kategorikal menjadi nilai numerikal, yaitu,

- *Ordinal encoding*
- *One-hot encoding*
- *Dummy variable encoding*

a) Ordinal Encoding

Pada metode *ordinal encoding*, setiap kategori yang unik diberi nilai integer. Misalnya, "merah" adalah 1, "hijau" adalah 2, dan "biru" adalah 3. Biasanya nilai integer yang digunakan berawal dai nilai 0. *Ordinal encoding* lebih cocok untuk data bertipe variabel nominal dimana tidak terdapat hubungan atau urutan antar variabel.

Sebagai contoh, terdapat data dengan tipe kategorikal seperti yang tertera pada Tabel 1.1. Data tersebut akan diubah menjadi data dalam bentuk numerik. Langkah pertama yang dilakukan adalah melakukan pengurutan data berdasarkan huruf abjad. Setelah diurutkan maka urutan pertama akan diberi nilai 0, 1, 2, dan seterusnya, sehingga didapatkan hasil pada Tabel 1.2. Implementasi dengan menggunakan library Scikit-learn pada Python telah memiliki *package* untuk melakukan transformasi data kategorikal ke data numerical dengan menggunakan metode ordinal encoder.

Tabel 1.1 Contoh data kategorikal

Kampus Vokasi Indonesia
Politeknik Negeri Malang
Politeknik Elektronika Negeri Surabaya
Politeknik Negeri Jakarta
Politeknik Negeri Semarang

Tabel 1.2 Contoh hasil encoding dengan ordinal encoding

Kampus Vokasi Indonesia	Nilai Ordinal
Politeknik Negeri Malang	**0**
Politeknik Elektronika Negeri Surabaya	**1**
Politeknik Negeri Jakarta	**2**
Politeknik Negeri Semarang	**3**

b) One-Hot Encoding

Metode *one-hot encoding* merepresentasikan nilai kategorikal menggunakan 1 fitur biner untuk setiap nilai yang memungkinkan. Untuk data kategorikal (variabel nominal) dimana tidak ada hubungan urutan peringkat antar nilai penggunaaan *ordinal encoding* tidak memberikan performa yang bagus pada model pembelajaran mesin. Memaksakan hubungan urutan (ordinal) melalui *ordinal encoding* memungkinkan model untuk berasumsi bahwa terdapat urutan antar kategori sehingga mengakibatkan kinerja yang buruk atau hasil yang tidak diharapkan. Dalam hal ini, *one-hot encoding* dapat diterapkan terhadap data yang memiliki tipe ordinal. Tabel 1.3 menunjukkan contoh *one-hot encoding*.

Tabel 1.3 Contoh one-hot encoding

Label	**0**	**1**	**2**	**3**
Politeknik Negeri Malang	1	0	0	0
Politeknik Elektronika Negeri Surabaya	0	1	0	0
Politeknik Negeri Jakarta	0	0	1	0
Politeknik Negeri Semarang	0	0	0	1

Langkah pertama yang dilakukan adalah melakukan pengurutan data berdasarkan huruf abjad. Setelah diurutkan selanjutnya nilai biner akan ditambahkan kepada setiap kategori. *One-hot encoding* akan menambah kolom fitur sesuai dengan nama politeknik yang ada di data. Nilai 1 menunjukkan bahwa pada baris tersebut terdapat data politeknik tersebut sedangkan nilai 0 menunjukkan sebaliknya. Sebagai contoh, Politeknik Negeri Malang akan direpresentasikan dengan [1, 0, 0, 0] dengan "1" untuk nilai biner pertama, kemudian Politeknik Elektronika Negeri Surabaya direpresentasikan dengan [0, 1, 0, 0], dan seterusnya.

c) Dummy Variable Encoding

One-hot encoding membuat satu variabel biner untuk setiap kategori. Terdapat redundansi dalam *one-hot encoding*. Contoh jika [1, 0, 0] mewakili "Politeknik Negeri Malang" dan [0, 1, 0] mewakili "Politeknik Elektronika Negeri Surabaya". Maka, pada *dummy variabel encoding* setiap nilai fitur akan diwakili oleh dua nilai biner, [0, 0]. Tabel 1.4 merupakan contoh penerapan *dummy variable encoding*.

Tabel 1.4 Contoh dummy variable encoding

Label	Nilai Biner	
Politeknik Negeri Malang	0	1
Politeknik Elektronika Negeri Surabaya	0	0
Politeknik Negeri Jakarta	1	0

Dummy variable encoding dapat diimplementasikan menggunakan Scikit-learn dapat dengan menggunakan kelas OneHotEncoder. Gunakan argumen "drop" untuk menunjukkan kategori mana yang akan datang yang diberi nol semua, kategori yang diberi nilai 0 ini disebut *baseline*. Argumen "drop" dapat diberi nilai "first" yang artinya kategori pertama yang akan diberi nilai 0.

C. Praktikum

Pada subbab ini Anda akan mempelajari penerapan teori yang telah disampaikan dengan menggunakan bahasa pemrograman Python.

1. Implementasi Normalisasi

Kode 1-1 merupakan implementasi normalisasi berdasarkan Persamaan 1.

```
def norm_data(data):
    '''
    Melakukan normalisasi data.

    Parameters:
        data (list) : Data yang akan dinormalisasi

    Returns:
        data (list) : Data hasil normalisasi
    '''

    data_max = max(data)
    data_min = min(data)
    data_len = len(data)

    for i in range(0, data_len):
        data[i] = (data[i] - data_min) / (data_max - data_min)

    return data

# Contoh Penggunaan
data = [10, 11, 12, 14, 16]
n_data = norm_data(data) # melakukan normalisasi
print(n_data)
```

Kode 1-1. Implementasi normalisasi manual berdasarkan Persamaan 1

Baris ke-1 hingga ke-19 merupakan fungsi normalisasi dengan input sebuah *list* atau *array*. Perhitungan normalisasi nilai fitur dilakukan pada baris ke-16 hingga baris ke-17. Kode 1 akan menghasilkan hasil seperti pada Gambar

```
[0.0, 0.16666666666666666, 0.3333333333333333, 0.6666666666666666, 1.0]
```

Gambar 1.3 Hasil perintah pada kode 1

Selanjutnya, Kode 1-2 merupakan implementasi normalisasi dengan menggunakan library Scikit-learn untuk kasus data fitur dalam bentuk 2 dimensi (2d list atau 2d array).

```
1  import numpy as np
2  from sklearn.preprocessing import MinMaxScaler
3
4  np.set_printoptions(precision=6) # bulatkan 4 angkat koma
5  np.set_printoptions(suppress=True) # hilangkan nilai e
6
7  # Kita akan membentuk data
8  # Hal ini dikarenakan, scikit-learn hanya menerima input
9  # dalam bentuk n-dimensional array
10 data = [
11     [100, 0.0001],
12     [50, 0.05],
13     [30, 0.003]
14 ]
15
16 # Ubah ke bentuk numpy n-dimensional array
17 data = np.asarray(data)
18 print('Data Asli')
19 print(data)
20
21 # Mendefinisikan obyek MinMaxScaler
22 scaler = MinMaxScaler()
23 # Transformasikan data
24 scaled = scaler.fit_transform(data)
25 print('Data Normalisasi')
26 print(scaled)
```

Kode 1-2 Implementasi normalisasi dengan Scikit-learn

```
Data Asli
[[100.       0.0001]
 [ 50.       0.05  ]
 [ 30.       0.003 ]]
Data Normalisasi
[[1.       0.       ]
 [0.285714 1.       ]
 [0.       0.058116]]
```

Gambar 1.4 Hasil normalisasi dengan Kode 1-2

Fungsi normalisasi pada library Scikit-learn berasal dari kelas ‘MinMaxScaler’, oleh karena itu perlu kita mengimpor kelas tersebut seperti pada baris ke-2. Kemudian, baris ke-22 digunakan untuk mendefinisikan obyek ‘MinMaxScaler’. Proses transformasi normalisasi data dilakukan pada baris ke-24. Hasil dari Kode 1-2 dapat dilihat pada Gambar 1.4.

2. Implementasi Standarisasi

Kode 1-3 merupakan implementasi standarisasi dengan menggunakan library Scikit-learn. Pada Scikit-learn, standarisasi dapat dilakukan dengan menggunakan kelas ‘StandardScaler’. Baris ke-2 digunakan untuk memanggil kelas ‘StandardScaler’. Kemudian, baris ke-22 digunakan untuk mendefinisikan obyek ‘StandardScaler’. Proses transformasi standarisasi data dilakukan pada baris ke-24. Hasil dari Kode 1-3 dapat dilihat pada Gambar 1.5.

```
1  import numpy as np
2  from sklearn.preprocessing import StandardScaler
3
4  np.set_printoptions(precision=6) # bulatkan 4 angkat koma
5  np.set_printoptions(suppress=True) # hilangkan nilai e
6
7  # Kita akan membentuk data
8  # Hal ini dikarenakan, scikit-learn hanya menerima input
9  # dalam bentuk n-dimensional array
10 data = [
11     [100, 0.0001],
12     [50, 0.05],
13     [30, 0.003]
14 ]
15
16 # Ubah ke bentuk numpy n-dimensional array
17 data = np.asarray(data)
18 print('Data Asli')
19 print(data)
20
21 # Mendefinisikan obyek MinMaxScaler
22 scaler = StandardScaler()
23 # Transformasikan data
24 scaled = scaler.fit_transform(data)
25 print('Data Standarisasi')
26 print(scaled)
```

Kode 1-3 Implementasi standarisasi dengan Scikit-learn

```
Data Asli
[[100.       0.0001]
 [ 50.       0.05  ]
 [ 30.       0.003 ]]
Data Standarisasi
[[ 1.358732 -0.76956 ]
 [-0.339683  1.412317]
 [-1.019049 -0.642757]]
```

Gambar 1.5 Hasil standarisasi dengan Kode 1-3

3. Implementasi Ordinal Encoding

Pada percobaan ini kita akan mencoba melakukan *ordinal encoding* sesuai dengan studi kasus pada bagian teori. Kode 1-4 merupakan implementasi dari *ordinal encoding* dengan menggunakan Scikit-learn.

```
1  from sklearn.preprocessing import OrdinalEncoder
2
3  # Inisiasi obyek Ordinal Encoder
4  oe = OrdinalEncoder()
5
6  # Definisikan data
7  # dalam bentuk 2d
8  data = [
9      ['Politeknik Negeri Malang'],
10     ['Politeknik Elektronika Negeri Surabaya'],
11     ['Politeknik Negeri Jakarata'],
12     ['Politeknik Negeri Semarang']
13 ]
14
15 # Transformasi Ordinal Encoder
16 transform_oe = oe.fit_transform(data)
17
18 print('Data Asli')
19 print(data)
20
21 print('Data Transformasi Ordinal Encoder')
22 print(transform_oe)
```

Kode 1-4 Implementasi ordinal encoder dengan Scikit-learn

Baris ke-1 digunakan untuk mengimport kelas ‘OrdinalEncoder’. Selanjutnya, dilakukan iniasi obyek ‘OrdinalEncoder’ pada baris ke-4. Transformasi data menjadi bentuk ordinal encoder dilakukan pada baris ke-16. Hasil dari Kode 1-4 dapat dilihat pada Gambar 1.6.

```
Data Asli
[['Politeknik Negeri Malang'], ['Politeknik Elektronika Negeri Surabaya'], ['Politeknik
Negeri Jakarata'], ['Politeknik Negeri Semarang']]
Data Transformasi Ordinal Encoder
[[2.]
 [0.]
 [1.]
 [3.]]
```

Gambar 1.6 Hasil ordinal encoding berdasarkan Kode 1-4

4. Implementasi One-Hot Encoding

Pada bagian ini, kita akan mempelajari bagaimana cara mengimplementasikan *one-hot encoding* dengan menggunakan Scikit-learn. Kode 1-5 merupakan contoh implmentasi dari *one-hot encoding* dengan menggunakan contoh seperti

pada kasus sebelumnya yaitu nama-nama kampus vokasi. Baris ke-1, kita akan menggunakan kelas 'OneHotEncoder'. Selanjutnya, pada baris ke-4 dilakukan inisiasi obyek 'OneHotEncoder'. Hasil dari 'OneHotEncoder' merupakan sebuah obyek, sehingga pada baris ke-22 perlu dirubah kedalam bentuk array agar dapat lebih mudah dipahami hasilnya. Hasil dari Kode 1-5 dapat dilihat pada Gambar 1.7.

```
1  from sklearn.preprocessing import OneHotEncoder
2
3  # Inisiasi obyek Ordinal Encoder
4  ohe = OneHotEncoder()
5
6  # Definisikan data
7  # dalam bentuk 2d
8  data = [
9      ['Politeknik Negeri Malang'],
10     ['Politeknik Elektronika Negeri Surabaya'],
11     ['Politeknik Negeri Jakarata'],
12     ['Politeknik Negeri Semarang']
13 ]
14
15 # Transformasi Ordinal Encoder
16 transform_ohe = ohe.fit_transform(data)
17
18 print('Data Asli')
19 print(data)
20
21 print('Data Transformasi One-Hot Encoding')
22 print(transform_ohe.toarray())
```

Kode 1-5 Implementasi one-hot encoding dengan Sckit-learn

```
Data Asli
[['Politeknik Negeri Malang'], ['Politeknik Elektronika Negeri Surabaya'], ['Politeknik Negeri Jakarata'], ['Politeknik Negeri Semarang']]
Data Transformasi One-Hot Encoding
[[0. 0. 1. 0.]
 [1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 0. 1.]]
```

Gambar 1.7 Hasil one-hot encoding berdasarkan Kode 1-5

5. Implementasi Dummy Variable Encoding

Pada Scikit-learn, *dummy variable encoding* dapat dilakukan dengan menggunakan kelas yang sama dengan *one-hot encoding* yaitu dengan kelas 'OneHotEncoding'. Namun, pada implementasi dummy variable encoding, perlu di definisikan parameter 'drop' dengan nilai 'first'. Kode 1-6 merupakan contoh implementasi *dummy variable encoding*. Pada baris ke-4, inisiasi obyek 'OneHotEncoding' dilengkapi dengan parameter 'drop=False'. Hasil dari Kode 1-6 dapat lihat pada Gambar 1.8. Jika diperhatikan, hasil encoding hanya memiliki 3 kolom, dan hasil pada 'Politeknik Elektronika Negeri Surabaya'

adalah [0, 0, 0]. Hasil ini membuktikan pada metode *dummy variable encoding* setiap nilai fitur hanya memiliki 1 nilai biner dari total kombinasi $n - 1$, dimana n adalah jumlah dari jenis fitur.

```python
from sklearn.preprocessing import OneHotEncoder

# Inisiasi obyek Ordinal Encoder
de = OneHotEncoder(drop='first')

# Definisikan data
# dalam bentuk 2d
data = [
    ['Politeknik Negeri Malang'],
    ['Politeknik Elektronika Negeri Surabaya'],
    ['Politeknik Negeri Jakarata'],
    ['Politeknik Negeri Semarang']
]

# Transformasi Ordinal Encoder
transform_de = de.fit_transform(data)

print('Data Asli')
print(data)

print('Data Transformasi One-Hot Encoding')
print(transform_de.toarray())
```

Kode 1-6 Implementasi dummy variable encoding dengan Scikit-learn

```
Data Asli
[['Politeknik Negeri Malang'], ['Politeknik Elektronika Negeri Surabaya'], ['Politeknik Negeri Jakarata'], ['Politeknik Negeri Semarang']]
Data Transformasi One-Hot Encoding
[[0. 1. 0.]
 [0. 0. 0.]
 [1. 0. 0.]
 [0. 0. 1.]]
```

Gambar 1.8 Hasil dummy variable encoding berdasarkan Kode 1-6

6. Studi Kasus Ekstrasi Fitur dari Data Teks

Data teks sering kali membutuhkan proses pra pengolahan data sebelum data tersebut dilatih dengan model pembelajaran mesin. Proses pra pengolahan data teks bertujuan untuk membuat dokumen masukan lebih konsisten dan mempermudah representasi teks. Pada pengolahan data teks secara umum berisi tiga proses utama yaitu *tokenizing*, *stopword removal*, dan *stemming*.

- Tahap *tokenizing* adalah tahap pemotongan string berdasarkan tiap kata yang menyusun sebuah kalimat.
- *Stopword removal* mengeliminasi kata – kata yang tidak penting berdasarkan daftar stopword. Contoh stopword adalah “yang”, “dan”,

"di", "dari" dan seterusnya. Gambar 1.9 merupakan cobtoh proses *stopword removal*.

Sebelum	Sesudah
|ppdb| |dengan| |sistem| |zonasi| |di| |wilayah| |dki| |jakarta| |ukurannya| |bukan| |lagi| |jarak| |dari| |rumah| |ke| |sekolah|	|ppdb| |sistem| |zonasi| |wilayah| |dki| |jakarta| |ukurannya| |jarak| |rumah| |sekolah|
|mohon| |di| |jawab| |bapak| |apakah| |sistem| |zonasi| |ppdb| |sma| |jakarta| |akan| |memprioritaskan| |peserta| |yang| |terdekat| |tempat| |tinggalnya| |dengan| |sekolah|	|mohon| |sistem| |zonasi| |ppdb| |sma| |jakarta| |akan| |memprioritaskan| |peserta| |terdekat| |tinggalnya| |sekolah|
|hai| |admin| |mengenai| |sistem| |zonasi| |ppdb| |jalur| |umum| |yang| |menjadi| |pertimbangan| |itu| |radius| |kamu| |atau| |harus| |satu| |regional| |iya| |contohnya| |sekolah| |yang| |bertempat| |di| |jakarta| |pusat| |lebih| |memprioritaskan| |calon| |siswa| |dengan| |kk| |di| |jakarta| |pusat| |juga| |kah| |terima| |kasih|	|admin| |sistem| |zonasi| |ppdb| |jalur| |pertimbangan| |radius| |regional| |contohnya| |sekolah| |bertempat| |jakarta| |pusat| |memprioritaskan| |calon| |siswa| |kk| |jakarta| |pusat| |terima| |kasih|

Gambar 1.9 Contoh proses stopword removal

- Pada tahap *stemming*, dilakukan proses untuk mencari kata dasar. Jadi, setiap kata yang memiliki imbuhan seperti imbuhan awalan dan akhiran, maka akan diambil kata dasarnya saja.
- Selanjutnya, *text representation*. Metode ini merupakan metode yang paling umum untuk memodelkan dokumen dengan cara mengubah setiap kata (*term*) menjadi vektor numerik. Representasi ini disebut "*Bag Of Words*" (BoW) atau "*Vector Space Model*" (VSM). Dalam VSM setiap kata yang terdapat didalam dokumen merupakan representasi dari fitur yang berbeda tanpa dipertimbangkan hubungan semantik antar kata yang ada di dalam dokumen. Selanjutnya setiap kata akan direpresentasikan dengan bobot. Metode pembobotan kata yang paling populer adalah *Term frequency-inverse document frequency* (TF-IDF). Persamaan 3 digunakan untuk memodelkan TF-IDF.

$$tfidf(t_k) = tf * log\left(\frac{N}{df(t_k)}\right) \quad (3)$$

  Dimana $tfidf(t_k)$ merupakan bobot term w didalam dokumen, tf merupakan frekuensi kemunculan term t_k didalam sebuah dokumen, dan df adalah jumlah dokumen yang memiliki kata t_k.

Pada studi kasus ini, kita akan membuat fitur TF-IDF dengan menggunakan data teks sederhana.

1. Siapkan data teks seperti yang ada pada Kode 1-7.

```
corpus = [
    'the house had a tiny little mouse',
    'the cat saw the mouse',
    'the mouse ran away from the house',
    'the cat finally ate the mouse',
    'the end of the mouse story'
]
```

Kode 1-7 Contoh inisasi data teks

2. Lakukan pembobotan dengan menggunakan metode TF-IDF seperti pada Kode 1-8. Pada baris ke-3, digunakan konfigurasi '*stopword*' dengan nilai '*english*'. Konfigurasi dilakukan karena kita menggunakan kata dalam bahasa inggris. Gambar 1.11 merupakan hasil pembobotan.

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Inisiasi obyek TFidfVectorizer
vect = TfidfVectorizer(stop_words='english')

# Pembobotan TF-IDF
resp = vect.fit_transform(corpus)

# Cetak hasil
print(resp)
```

Kode 1-8 Kode pembobotan TF-IDF dengan Scikit-learn

```
  (0, 7)    0.2808823162882302
  (0, 6)    0.5894630806320427
  (0, 11)   0.5894630806320427
  (0, 5)    0.47557510189256375
  (1, 9)    0.7297183669435993
  (1, 2)    0.5887321837696324
  (1, 7)    0.3477147117091919
  (2, 1)    0.5894630806320427
  (2, 8)    0.5894630806320427
  (2, 7)    0.2808823162882302
  (2, 5)    0.47557510189256375
  (3, 0)    0.5894630806320427
  (3, 4)    0.5894630806320427
  (3, 2)    0.47557510189256375
  (3, 7)    0.2808823162882302
  (4, 10)   0.6700917930430479
  (4, 3)    0.6700917930430479
  (4, 7)    0.3193023297639811
```

Gambar 1.10 Hasil pembobotan TF-IDF

Pada Gambar 1.10, kolom 1 merupakan indeks dari dokumen pada *corpus*. Kolom 2 merupakan indeks dari token yang terdapat dalam kalimat. Kolom 3 merupakan bobot dari TF-IDF hasil kalkukasi.

3. Untuk mendapatkan token kata-kata yang didapatkan dari proses stopwords, gunakan perintah seperti pada Kode 1-9. Hasilnya akan seperti pada Gambar 1.11

```
vect.get_feature_names_out()
```

Kode 1-9 Perintah untuk mendapatkan token

```
array(['ate', 'away', 'cat', 'end', 'finally', 'house', 'little', 'mouse',
       'ran', 'saw', 'story', 'tiny'], dtype=object)
```

Gambar 1.11 Token hasil proses stopword

4. Pada proses ini, kita telah selesai melakukan ektraksi fitur teks yang hasilnya disimpan pada variabel ‘resp’.
5. Kode lengkap dapat dilihatp ada Kode 1-10

```
from sklearn.feature_extraction.text import TfidfVectorizer

corpus = [
    'the house had a tiny little mouse',
    'the cat saw the mouse',
    'the mouse ran away from the house',
    'the cat finally ate the mouse',
    'the end of the mouse story'
]

# Inisiasi obyek TFidfVectorizer
vect = TfidfVectorizer(stop_words='english')

# Pembobotan TF-IDF
resp = vect.fit_transform(corpus)

# Cetak hasil
print('Hasil TF-IDF')
print(resp)

# Cetak token hasil stopword
print('Hasil Token')
print(vect.get_feature_names_out())
```

Kode 1-10 Kode lengkap contoh ekstrasi fitur TF-IDF

D. Tugas Praktikum

1. Salin kalimat pada Kode 1-7 dengan tanda baca titik pada setiap kalimatnya dengan menggunakan editor teks.
2. Simpan kalimat tersebut pada file '.txt' dengan nama 'corpus.txt'.
3. Lakukan proses ektraksi fitur TF-IDF dengan menggunakan file 'corpus.txt'.