BAB 2
LANDASAN TEORI

2.1 Klasifikasi Teks

Klasifikasi teks adalah sebuah teknik dalam *machine learning* yang bertujuan untuk mengkategorikan suatu data teks menjadi sebuah kategori berdasarkan ciri-ciri teks tersebut[8]. Untuk melakukan klasifikasi teks pada data yang besar secara manual akan memakan waktu yang lama dan sulit, serta hasil akurasi klasifikasi teks manual akan dipengaruhi oleh faktor manusia seperti kelelahan dan keahliannya. Maka dari itu *machine learning* digunakan untuk mengotomasikan proses klasifikasi teks agar mendapat hasil yang lebih akurat dan objektif[8].

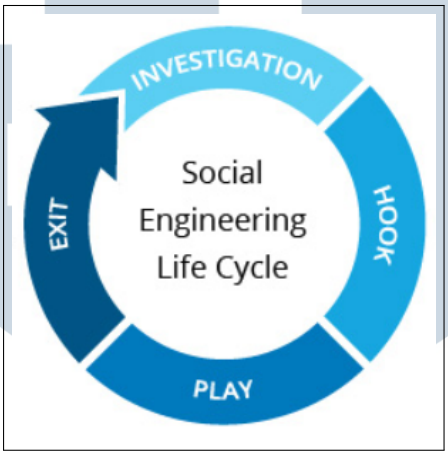
Klasifikasi teks adalah teknologi yang terus berkembang dan memiliki beragam aplikasi di berbagai sektor industri. Sebagai contoh dalam sektor bisnis klasifikasi teks dapat digunakan untuk menganalisis sentimen pelanggan terhadap sebuah produk dan merencanakan kampanye yang sesuai. Dalam sektor media sosial, klasifikasi teks dapat dimanfaatkan dalam memoderasi suatu grup diskusi atau forum dan secara otomatis mendeteksi kata yang sensitif dan mungkin melanggar peraturan komunitas[9]. Selain itu klasifikasi teks juga dapat digunakan untuk mendeteksi pesan spam dan penipuan berdasarkan karakteristik dan pola teks, sehingga pengguna dapat diberikan peringatan dini atau memblokir pesan secara otomatis agar pengguna dapat terhindar dari potensi risiko dan kerugian[10].

Kemampuan klasifikasi teks dalam mengolah dan mengkategorikan data secara otomatis memberikan berbagai keuntungan yang berarti di berbagai sektor. Dengan terus berkembangnya *machine learning*, teknik klasifikasi teks akan terus memainkan peran penting dalam pengambilan informasi, penambangan data, dan pengambilan keputusan. Integrasi algoritma yang semakin canggih, memungkinkan sistem klasifikasi teks untuk mengatasi kekurangan yang ada dan membuka peluang untuk diaplikasikan di industri lainnya[10].

2.2 *Social Engineering*

Social Engineering atau Rekayasa Sosial merupakan suatu teknik manipulasi dengan interaksi manusiawi seperti bujukan dan pengaruh yang memiliki tujuan untuk mengendalikan korban sehingga melakukan suatu tindakan,

baik secara sadar maupun tidak sadar[11]. Teknik ini memanfaatkan kerentanan manusia seperti perasaan, rasa percaya, dan kebiasaan untuk mengambil data atau informasi rahasia. Banyak pelaku kriminal yang menggunakan teknik *social engineering* karena biasanya lebih mudah untuk memanfaatkan rasa percaya korban dibandingkan dengan meretas perangkat lunak korban tersebut. Fokus dari serangan *social engineering* adalah cara seseorang berperilaku, bereaksi, dan berfikir. Karena ketika pelaku sudah mengetahui hal-hal tersebut, maka orang itu dapat dimanipulasi dengan mudah[12]. Berikut adalah ilustrasi siklus hidup *social engineering* dan penjelasannya[13].



Gambar 2.1. Siklus hidup *social engineering*

1. *Investigation*, pelaku memilih korban dan melakukan riset serta menentukan jenis serangan yang cocok
2. *Hook*, di tahap ini pelaku akan mendekati dan berinteraksi dengan korban untuk mendapatkan kepercayaan korbannya
3. *Play*, setelah korban sudah mempercayai pelaku dan pelaku sudah tahu titik lemah korban, pelaku memanipulasi korban untuk mendapatkan data pribadinya.
4. *Exit*, setelah pelaku sudah mendapatkan yang diinginkan, ia akan mengakhiri segala komunikasi dengan korban dan mencari target baru.

Langkah-langkah dalam semua jenis serangan *social engineering* kurang lebih adalah sama, tetapi di kenyataan jenis-jenis serangan *social engineering* makin bertambah dengan modus-modus penipuan yang baru. Berikut adalah beberapa jenis serangan *social engineering*[14].

- *Phishing* merupakan salah satu bentuk *social engineering* yang bertujuan untuk menjerat korban melalui pendekatan manipulatif, di mana pelaku berusaha "memancing" korban agar secara tidak sadar memberikan informasi pribadi atau data penting yang dibutuhkan oleh pelaku [15]. Kejahatan phishing yang dilakukan secara *online* dapat menimbulkan kerugian bagi korban dan termasuk tindakan yang dilarang oleh hukum. Saat ini, platform media sosial sering dimanfaatkan oleh oknum yang tidak bertanggung jawab untuk menjalankan aksi phishing secara online. Pelaku phishing biasanya memanfaatkan berbagai saluran yang terhubung dengan internet, seperti *e-mail*, SMS, dan *website*. Salah satu modus yang umum dilakukan adalah dengan menawarkan hadiah kepada calon korban dan menyisipkan tautan menuju situs palsu. Ketika korban mengakses tautan tersebut atau mengisi data pribadi, informasi tersebut dapat dicuri atau disalahgunakan [16].

- *Baiting* adalah jenis serangan *social engineering* yang menggunakan umpan untuk memancing korban agar menyerahkan informasi mereka atau mengunduh aplikasi berbahaya. Salah satu contoh umpan serangan *baiting* adalah *website* untuk mengunduh *game*, musik, dan *software* secara gratis.

- *Tailgating* merupakan jenis serangan *social engineering* yang tidak dilakukan secara *online*, melainkan secara langsung. Pelaku akan berbaur dengan sekitarnya untuk mendapatkan akses kedalam suatu area yang memiliki barang berharga atau informasi penting.

- *Pretexting* adalah salah satu jenis *social engineering* yang memanfaatkan rasa percaya korban, pelaku *pretexting* akan membuat situasi palsu dimana korban seakan-akan dalam bahaya, dan pelaku akan menyelamatkan korban dari bahaya tersebut dengan mengendalikan perangkat korban secara *remote* atau meminta informasi penting dari korban.

- *Quid pro quo*, dalam jenis serangan ini pelaku menjanjikan jasa atau barang untuk memancing korban mereka agar membuka data sensitif, contohnya adalah telefon *IT support* dan SMS pemenang undian.

2.3 Algoritma *Naïve Bayes*

Algoritma *Naïve Bayes* adalah salah satu algoritma yang paling populer untuk mengklasifikasikan data, popularitas ini dikarenakan karena konsepnya yang

sederhana namun memiliki kecepatan dan akurasi tinggi[17]. Efisiensi ini muncul dari asumsi bahwa setiap atribut data bersifat independen satu dengan yang lainnya berdasarkan kelasnya, meskipun asumsi yang dibuat oleh algoritma ini tidak sepenuhnya benar tidak benar dalam kenyataan, namun sering kali berhasil dalam praktik[17]. Maka dari itu algoritma ini memiliki nama *Naïve*.

Naïve Bayes merupakan algoritma probabilistik *machine learning* yang berdasarkan *Bayes Theorem*, yang digunakan dalam berbagai jenis klasifikasi. *Bayes Theorem* adalah rumus matematika yang membantu menentukan probabilitas suatu peristiwa berdasarkan peristiwa sebelumnya dan data baru[18]. Berikut adalah persamaan dari Algoritma *Naïve Bayes*.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2.1}$$

Keterangan:

1. P($A|B$) adalah probabilitas peristiwa A terjadi ketika peristiwa B terjadi.
2. P($B|A$) adalah probabilitas peristiwa B terjadi ketika peristiwa A terjadi.
3. P(A) adalah probabilitas peristiwa A.
4. P(B) adalah probabilitas peristiwa B.

2.4 TF-IDF *(Term Frequency-Inverse Document Frequency)*

TF-IDF atau *(Term Frequency-Inverse Document Frequency)* adalah salah satu library dari *skicit-learn* di *phyton* yang digunakan untuk melakukan data *preprocessing* dengan mengubah data teks menjadi matriks dan memperoleh tingkat kepentingan kata dalam suatu *dataset* [19]. Cara kerjanya adalah dengan menghitung jumlah kata yang muncul pada *dataset*, lalu dihitung tingkat kepentingan dari kata-kata tersebut[20]. Berikut rumus TF-IDF:

$$\text{TF} - \text{IDF}(t,d) = \text{TF}(t,d) \times \text{IDF}(t) \tag{2.2}$$

$$\text{TF}(t,d) = \frac{\text{Jumlah kata } t \text{ dalam dokumen } d}{\text{Jumlah total kata dokumen } d} \tag{2.3}$$

$$\text{IDF}(t) = \log\left(\frac{\text{Jumlah total dokumen dalam } d}{\text{Jumlah dokumen dimana terdapat kata } t}\right) \tag{2.4}$$

Keterangan:

1. TF-IDF(t, d) adalah bobot kata dari kata d dalam dataset t.
2. TF(t, d) adalah tf (*term-frequency*) atau frekuensi kata d dalam dataset t.
3. IDF(t) adalah banyaknya dokumen atau dataset yang mengandung kata t.

Berikut contoh visualisasi cara TF-IDF mengubah teks menjadi bentuk matriks.

```
    dataset = [
        "Kucing itu sangat lucu dan gemas",
        "Anjing adalah teman yang setia",
        "Kucing dan anjing adalah hewan peliharaan",
    ]

```

Kode 2.1: Text sebelum diubah dengan TF-IDF

Pada gambar 2.1 adalah teks sebelum dilakukan proses TF-IDF dan dibawah adalah tabel 2.1 dan 2.2 yang menampilkan teks setelah diubah ke bentuk matriks oleh TF-IDF.

	adalah	anjing	dan	gemas	hewan	itu	kucing
0	0.0000	0.0000	0.3349	0.4404	0.0000	0.4404	0.3349
1	0.3730	0.3730	0.0000	0.0000	0.0000	0.0000	0.0000
2	0.3662	0.3662	0.3662	0.0000	0.4815	0.0000	0.3662

Tabel 2.1. Teks setelah proses TF-IDF

	lucu	peliharaan	sangat	setia	teman	yang
0	0.4404	0.0000	0.4404	0.0000	0.0000	0.0000
1	0.0000	0.0000	0.0000	0.4905	0.4905	0.4905
2	0.0000	0.4815	0.0000	0.0000	0.0000	0.0000

Tabel 2.2. Teks setelah proses TF-IDF

Ketika teks sudah diubah menjadi bentuk matriks oleh *TFIDF vectorizer*, setiap kolom matriks adalah kata unik yang ditemukan dalam dataset dan diurutkan berdasarkan alfabet. Setiap baris matriks adalah indeks teks dari dataset dan angka di dalam sel adalah nilai TF-IDF kata tersebut. Contohnya kata ”setia” yang ada di teks indeks 1 memiliki nilai 0.4905, tetapi tidak muncul di teks indeks 0 dan 2, selain itu kata ”kucing” muncul di teks indeks ke 0 dan 2 dan memiliki nilai TF-IDF lebih rendah yaitu 0.3349 dan 0.3662 yang menandakan bahwa kata ”kucing” bersifat lebih umum di keseluruhan dokumen.

2.5 *Count Vectorizer*

Count Vectorizer adalah salah satu library dari *skicit-learn* di *phyton* yang digunakan untuk melakukan data *preprocessing* dengan cara mengubah teks di dalam suatu *dataset* menjadi bentuk matriks [21]. *Count Vectorizer* hanya menghitung kemunculan setiap kata dalam dokumen, beda halnya dengan TF-IDF yang juga melakukan pembobotan setiap kata selain menghitung jumlahnya. Maka kekurangan dari *Count Vectorizer* adalah ketika ada kata yang jarang muncul maka memiliki bobot rendah, padahal pada beberapa kasus kata yang jarang muncul bisa memiliki peran penting [22].

Berikut contoh visualisasi cara *Count Vectorizer* mengubah teks menjadi bentuk matriks.

```
    dataset = [
        "Kucing itu sangat lucu dan gemas",
        "Anjing adalah teman yang setia",
        "Kucing dan anjing adalah hewan peliharaan",
    ]

```

Kode 2.2: Text sebelum diubah dengan *Count Vectorizer*

Pada gambar 2.2 adalah teks sebelum dilakukan *Count Vectorizer* dan dibawah adalah tabel 2.3 dan 2.4 yang menampilkan teks setelah diubah ke bentuk matriks oleh 2.2.

	adalah	anjing	dan	gemas	hewan	itu	kucing
0	0	0	1	1	0	1	1
1	1	1	0	0	0	0	0
2	1	1	1	0	1	0	1

Tabel 2.3. Teks setelah proses *Count Vectorizer*

	lucu	peliharaan	sangat	setia	teman	yang
0	1	0	1	0	0	0
1	0	0	0	1	1	1
2	0	1	0	0	0	0

Tabel 2.4. Teks setelah proses *Count Vectorizer*

Ketika teks sudah diubah menjadi bentuk matriks oleh *count vectorizer*, semua kata sudah diubah menjadi huruf kecil, lalu setiap kolom matriks adalah kata unik yang ditemukan dalam *dataset* dan diurutkan berdasarkan alfabet. Setiap

baris matriks adalah indeks teks dari *dataset* dan angka di dalam sel adalah jumlah kata dalam teks tersebut[23].

Di dalam *count vectorizer* kata di dalam kolom pada tabel 2.3 dan 2.4 tidak disimpan dalam bentuk teks, melainkan diubah menjadi index, contohnya kolom yang berisi kata ”adalah” menjadi indeks 0, ”anjing” menjadi indeks 1, dan seterusnya[23]. Berikut adalah tabel vektorisasi teks setelah diberikan indeks.

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	1	1	0	1	1	1	0	1	0	0	0
1	1	1	0	0	0	0	0	0	0	0	1	1	1
2	1	1	1	0	1	0	1	0	1	0	0	0	0

Tabel 2.5. Teks setelah proses *Count Vectorizer* dengan indeks kata

2.6 *GridSearchCV*

GridSearchCV atau *Grid Search Cross-Validation* adalah salah satu fungsi dari library *skicit-learn* di *phyton* yang digunakan untuk melakukan *hyperparameter tuning* yaitu mencari *hyperparameter* yang terbaik dari serangkaian *hyperparameter* tertentu untuk sebuah model[24] dan mengevaluasi setiap kombinasi dengan *cross-validation*. Performa model sangat dipengaruhi oleh pemilihan nilai *hyperparameter*-nya. Karena tidak ada cara pasti untuk menentukan nilai optimal *hyperparameter* sejak awal, maka biasanya perlu mencoba berbagai kombinasi *hyperparameter* yang memungkinkan. Proses ini akan sangat memakan waktu dan sumber daya jika dilakukan secara manual. Oleh karena itu, *GridSearchCV* dapat di manfaatkan untuk mengotomatiskan pencarian kombinasi hiperparameter terbaik secara efisien[25].

```
1    GridSearchCV(
2        estimator=,     # A sklearn model
3        param_grid=,    # A dictionary of parameter names and
   values
4        cv=,            # An integer that represents the number of
    k-folds
5        scoring=,       # The performance measure (such as r2,
   precision)
6        n_jobs=,        # The number of jobs to run in parallel
7        verbose=        # Verbosity (0-3, with higher being more)
8    )
9
```

Kode 2.3: Fungsi *GridSearchCV*

Potongan kode 2.3 menampilkan fungsi *GridSearchCV* dan parameter-parameter yang dapat diisi ketika memanggil fungsi ini, berikut penjelasannya [26]:

- `estimator` berisi model atau *estimator* yang mau dilakukan *hyperparameter tuning*.
- `param_grid` berisi *dictionary* atau *dictionary list* yang merupakan *key-value pair* dimana *key* adalah nama *hyperparameter* dan *value* adalah nilai-nilai yang mau dicoba. Contohnya 'alpha': [0.01, 0.1, 1.0, 10.0].
- `cv` berisi angka yang menentukan jumlah *cross-validation*, jika tidak diisi memiliki nilai *default* 5.
- `scoring` berisi metrik evaluasi yang mau di gunakan, jika tidak diisi memiliki nilai *default None*.
- `n_jobs` berisi jumlah proses yang mau dijalankan secara bersamaan, jika tidak diisi memiliki nilai *default None*.
- `verbose` berisi angka yang menentukan jumlah informasi yang ditampilkan. Nilai 1 akan menampilkan waktu eksekusi untuk setiap iterasi. Jika menggunakan nilai 2, selain waktu, skor dari masing-masing iterasi juga ditampilkan. Sementara itu, nilai 3 akan menampilkan *fold* dan parameter yang diuji. Jika tidak diisi memiliki nilai *default* 0.