BAB 2
LANDASAN TEORI

2.1 Peluluhan kata

Menurut kaidah bahasa Indonesia, kata-kata yang berhuruf awal 's', 'p', 't', dan 'k' akan mengalami peluluhan ketika mendapatkan awalam me-. Namun bila merupakan kluster atau gugus konsonan, kata-kata yang berhuruf awal tersebut tidak mengalami peluluhan. Lain halnya ketika berhadapan dengan sesama awalan, awalan me- yang bervariasi dengan men-, meng-, meng-, dan mem tidak meluluhkan [15].

Contoh dari kata luluh ialah *mengantuk*, yang sebelumnya dari kata dasar *kantuk*. Namun, perlu diketahui bahwa kata luluh memiliki peraturan-peraturan khusus, seperti kata yang diawali 's', 'p', 't', 'k' dan huruf keduanya adalah huruf konsonan, maka tidak akan luluh katanya. Contoh dari peraturan tersebut adalah *mengkritik* yang berawal dari kata dasar *kritik*. Adapun kaidah lainnya yang walaupun kata dasarnya diawali dengan huruf 's', 'p', 't', 'k' tidak mengalami peluluhan kata seperti *mengkaji* dari kata dasar *kaji*. Hal ini bertujuan untuk membedakan kata *mengaji* yang memiliki perbedaan makna.

2.2 Text Preprocessing

Sebelum dilakukan pengaplikasian model *machine learning*, perlu diterapkan *text preprocessing* supaya model dapat lebih mudah mengenali data-data yang tersedia. *Text Preprocessing* adalah tahapan mentransformasi teks sebelum dilakukannya analisis dengan mengidentifikasi kata-kata dan frasa yang digunakan, menghapus kata hubung yang tidak relevan, sampai dengan menkonversi menjadi huruf kecil [23] [24]. *Text preprocessing* dibagi menjadi beberapa tahap antara lain,

(1) ***Lowercase conversation*** Tahap ini untuk mengubah huruf kapital menjadi huruf kecil.

(2) ***Handle Negation*** Tahap ini untuk merupakan sekumpulan metode yang digunakan untuk mengidentifikasi, merepresentasikan secara semantis, dan memperhitungkan negasi dalam teks. Biasa dengan simbol "_" yang ditambahkan pada akhiran kata yang berkonotasi negatif.

(3) ***Spelling correction*** Tahap ini untuk mengubah kata-kata yang saltik menjadi kata sebenarnya yang bertujuan untuk mengurangi ukuran *vocabulary*.

(4) ***Expand contractions and abbreviations*** Tahap ini mengubah seluruh kata singkatan seperti ”I’m” menjadi ”I am”. Hal ini bertujuan untuk membuat konotasi akronim tidak saling tumpang tindih.

(5) ***Nonalphabetic character removal*** Tahap ini akan menghilangkan tanda baca, angka, dan kata-kata yang bukan merupakan kata abjad.

(6) ***Emoticon recognition*** Tahap ini akan mengubah emoji menjadi kata yang mendekati dengan arti emoji tersebut.

(7) ***Stop Word Removal*** Tahap ini akan menghilangkan kata yang memiliki arti kata yang kurang/tidak penting.

(8) ***Lemmatizer*** Tahap ini akan mensimplifikasi kata menjadi kata baku dengan *vocabulary* yang tersedia.

(9) ***Stemmer*** Serupa dengan Lemmatizer, tahap ini akan menyimplifikasi kata menjadi kata baku dengan mengurangi kata depan dan imbuhan.

2.3 Damerau-Levenshtein distance

Damerau-Levenshtein distance adalah algoritma yang digunakan untuk mengukur seberapa berbedanya dua string teks dengan memperhitungkan operasi pengeditan dasar seperti penyisipan, penghapusan, penggantian, dan transposisi karakter [25]. Algoritma ini berguna dalam aplikasi pemrosesan bahasa alami, dan koreksi ejaan [21]. Algoritma Damerau-Levenshtein distance dapat didefinisikan pada rumus 2.1 [26].

$$f_{a,b}(i,j) = \min \begin{cases} 0 \quad if i = j = 0 \\ f_{a,b}(i-1,j)+1 \quad \text{if } i > 0 \\ f_{a,b}(i-1,j)+1 \quad \text{if } j > 0 \\ f_{a,b}(i-1,j-1)+1 \, (a_i \text{ and } b_j \text{ are not equal }) \quad \text{if } i,j > 0 \\ f_{a,b}(i-2,j-2)+1 \quad \text{if } i,j > 1 \text{ and } a_{i-1} = b_j \text{ and } a_i = b_{j-1} \end{cases} \tag{2.1}$$

Pertama-tama, fungsi menginisialisasi variabel len_s1 dan len_s2 untuk menyimpan panjang masing-masing string input. Kemudian, dibuat matriks dua dimensi yang akan digunakan untuk menyimpan jarak antara setiap pasangan

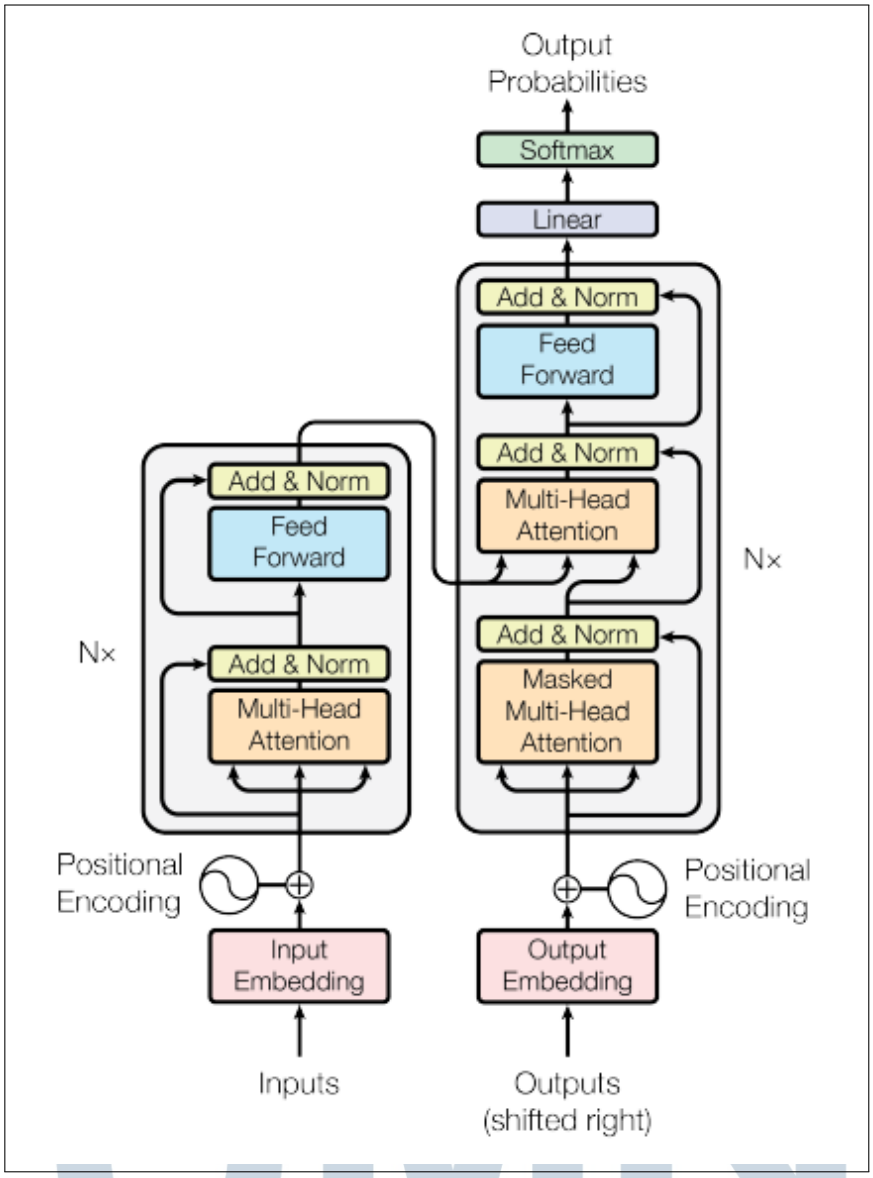
karakter dari kedua string. Inisialisasi dilakukan pada baris pertama dan kolom pertama matriks untuk mewakili jarak dari string kosong ke setiap karakter dalam string input. Selanjutnya, fungsi melakukan iterasi melalui setiap karakter dari kedua string menggunakan *looping*. Pada setiap iterasi, nilai jarak antara dua string diperbarui berdasarkan operasi minimum yang diperlukan. Operasi-operasi ini meliputi *deletion*, *insertion*, dan *subsitution*. Selain itu, algoritma juga mempertimbangkan kemungkinan transposisi karakter, yaitu jika dua karakter berdekatan dari kedua string dapat ditukar posisinya. Setelah semua iterasi selesai, nilai terakhir dari matriks berisi jarak Damerau-Levenshtein antara kedua string, yang kemudian dikembalikan sebagai output dari fungsi. Potongan kode dapat dilihat pada 2.1

```
def damerau_levenshtein_distance(s1, s2):
    len_s1 = len(s1)
    len_s2 = len(s2)
    d = [[0] * (len_s2 + 1) for _ in range(len_s1 + 1)]

    for i in range(len_s1 + 1):
        d[i][0] = i
    for j in range(len_s2 + 1):
        d[0][j] = j

    for i in range(1, len_s1 + 1):
        for j in range(1, len_s2 + 1):
            cost = 0 if s1[i - 1] == s2[j - 1] else 1
            d[i][j] = min(
                d[i - 1][j] + 1,  # deletion
                d[i][j - 1] + 1,  # insertion
                d[i - 1][j - 1] + cost,  # substitution
            )
            #transposition
            if i > 1
                and j > 1
                and s1[i - 1] == s2[j - 2]
                and s1[i - 2] == s2[j - 1]:
                d[i][j] = min(d[i][j], d[i - 2][j - 2] + cost)
    return d[len_s1][len_s2]
```

Kode 2.1 Damerau-Levenshtein Distance

2.4 Transformer

Transformer merupakan sebuah arsitektur model dalam bidang Pemrosesan Bahasa Alami (Natural Language Processing/NLP) yang terbilang revolusioner. Transformer menawarkan pendekatan baru dalam pemodelan sekuen dengan mengeliminasi ketergantungan pada arsitektur sekuenal seperti LSTM (Long Short-Term Memory) atau RNN (Recurrent Neural Network). Arsitektur Transformer memiliki dua komponen utama, yaitu encoder dan decoder, yang berperan dalam menerjemahkan atau menghasilkan teks dalam tugas-tugas terjemahan bahasa atau generasi teks. Salah satu keunggulan utama dari Transformer adalah kemampuannya dalam memahami konteks yang lebih luas dalam teks dengan memperhitungkan seluruh konteks sekaligus melalui mekanisme perhatian yang terkontrol. Hal ini memungkinkan Transformer untuk mengatasi masalah-masalah masuk akal dalam pemrosesan bahasa, seperti mempertahankan informasi penting dalam teks [27]. Arsitektur Transformer dapat dilihat pada gambar 2.1.



Gambar 2.1 Transformer

Sumber: [27]

Dalam arsitektur ini, urutan *input* pertama kali diproses oleh sebuah *encoder* untuk menghasilkan representasi berukuran tetap, yang menangkap informasi dari urutan *input* tersebut. Encoder umumnya terdiri dari beberapa lapisan jaringan saraf, seperti *Recurrent Neural Networks* (RNNs), *Long Short-term Memory Networks* (LSTMs), atau *transformers*, yang memproses urutan input secara hierarkis. Setelah urutan input dienkoding menjadi representasi berukuran tetap, lalu diteruskan ke *decoder*, yang menghasilkan urutan *output* berdasarkan representasi ini. Decoder juga terdiri dari beberapa lapisan jaringan saraf dan menggunakan mekanisme perhatian (*attention*) untuk fokus pada bagian-bagian berbeda dari urutan *input* saat menghasilkan urutan *output* [27].

Input and Output Embedding Input and Output Embedding menggunakan *word embedding* dalam struktur model Transformer. Dalam *word embeddings*, kata-kata dari kosakata ditempatkan

dalam ruang vektor dimana kata-kata yang sering muncul bersama atau memiliki makna yang mirip akan memiliki representasi vektor yang mendekat satu sama lain. Representasi ini memungkinkan model pembelajaran mesin untuk menangkap hubungan semantik dan sintaktik antara kata-kata, yang dapat digunakan untuk berbagai tugas, termasuk NLP, klasifikasi teks, terjemahan mesin, dan lainnya.

2.4.1 *Positional Encoding*

Positional Encoding atau *Positional embeddings* adalah jenis khusus dari *embeddings* yang digunakan dalam model seperti Transformer untuk memperhitungkan urutan relatif dari kata atau token dalam sebuah kalimat atau urutan input. Dalam model Transformer, seperti yang diperkenalkan dalam makalah "Attention Is All You Need" [27], urutan relatif dari kata-kata dalam teks diperhitungkan menggunakan *positional embeddings*. Hal ini dilakukan karena model Transformer tidak memiliki komponen seperti *recurrent layer* yang secara alami memperhitungkan urutan.

2.4.2 *Multi-Head Attention*

Multi-head attention adalah salah satu komponen kunci dalam model Transformer, yang digunakan untuk mengatasi permasalahan keterbatasan perhatian tunggal. *Dalam multi-head attention*, perhatian (*attention*) dihitung beberapa kali secara paralel menggunakan beberapa saluran atau "kepala" (*heads*) yang berbeda. Setiap kepala *attention* mempelajari representasi yang berbeda dari hubungan antara token dalam urutan input. Kemudian, hasil dari setiap kepala attention dikombinasikan kembali untuk menghasilkan representasi akhir. Ini memungkinkan model untuk memperhatikan berbagai jenis hubungan antara kata dalam teks, yang dapat meningkatkan kemampuan model dalam memahami konteks dan struktur dari data masukan.

2.4.3 *Self-Attention*

Dalam konteks model Transformer, *self-attention* digunakan untuk memproses urutan input, seperti kalimat atau urutan token. Ide dasar dari *self-attention* adalah memungkinkan setiap elemen dalam urutan untuk "memperhatikan" atau memperhitungkan hubungannya dengan setiap elemen lainnya dalam urutan tersebut. Proses *self-attention* melibatkan tiga langkah utama:

1. *Attention Scores*: Setiap token dalam urutan diubah menjadi tiga vektor: *query, key*, dan *value*. *Attention Scores* dihitung antara setiap pasangan query-key untuk menentukan seberapa pentingnya token tersebut dalam konteks hubungannya dengan token lainnya.
2. *Weighed Sum*: *Attention Scores* tersebut kemudian dinormalisasi menggunakan fungsi *softmax* untuk menghasilkan bobot yang menunjukkan seberapa banyak perhatian diberikan kepada setiap token dalam urutan.
3. Penggabungan informasi: Setiap nilai (*value*) dikalikan dengan bobot yang sesuai dan dijumlahkan untuk menghasilkan representasi akhir dari *self-attention*.

2.5 *Bidirectional Encoder Representations from Transformers* (BERT)

BERT (Bidirectional Encoder Representations from Transformers) merupakan salah satu model hasil dari transformer. Dikembangkan oleh Google pada tahun 2018, BERT memanfaatkan arsitektur Transformer yang kuat untuk menghasilkan representasi kata yang sangat kontekstual. Keunggulan utama BERT adalah kemampuannya dalam memahami hubungan konteks antara kata dalam sebuah kalimat dengan cara yang lebih baik daripada model *transformer vanilla*. BERT dilatih pada dua tugas bahasa utama, yaitu Masked Language Modeling (MLM) dan Next Sentence Prediction (NSP), yang membantu model memahami konteks dan hubungan antara kata dalam sebuah teks [28] sehingga BERT mampu untuk memberikan rekomendasi kata sesuai dengan konteks kalimat. Hasil dari BERT lebih baik 7.7% GLUE score, 4.6% MultiNLI *accuracy*, dan 1.5% Test F1 dibandingkan dengan *Transformer Vanilla* [28].

2.6 Hugging Face

Hugging Face adalah sebuah perusahaan dan platform open-source yang dikenal atas kontribusinya dalam bidang NLP. Hugging Face memiliki *Transformer library* yang mudah digunakan untuk menggunakan, mengunduh, dan melatih model-model transformer terkini, seperti BERT, GPT, dan lainnya. *Transformers library* dari Hugging Face memungkinkan para peneliti, pengembang, dan praktisi untuk dengan cepat mengakses berbagai model bahasa yang telah dilatih, melakukan fine-tuning, serta melatih model-model baru sesuai dengan kebutuhan spesifik. Platform ini juga menawarkan sumber daya komunitas yang aktif, seperti forum diskusi dan berbagai tutorial, yang memfasilitasi pertukaran pengetahuan dan kolaborasi di antara para pengguna [29].

2.7 Confusion Matrix

Confusion Matrix digunakan dalam pembelajaran mesin untuk menentukan tingkat akurasi dari suatu model [30]. Confusion Matrix terdiri dari *True Positive* (TP), *True Negative* (TN), *False Positive* (FP), *False Negative* (FN). *True Positive* (TP) merupakan jumlah kasus positif yang diprediksi dengan benar. Artinya, kelas sebenarnya adalah positif dan model juga memprediksi sebagai positif. *True Negative* merupakan jumlah kasus negatif yang diprediksi dengan benar. Artinya, kelas sebenarnya adalah negatif dan model juga memprediksi sebagai negatif. *False Positive* merupakan jumlah kasus positif yang diprediksi secara salah. Artinya, kelas sebenarnya adalah negatif, tetapi model memprediksi sebagai positif. *False Negative* merupakan jumlah kasus negatif yang diprediksi secara salah. Artinya, kelas sebenarnya adalah positif, tetapi model memprediksi sebagai negatif. Hasil dari Confusion Matrix bisa mendapatkan nilai *recall*, *accuracy*, *precision*, dan *F-1 Score*. Hasil dari *F-1 Score* dapat merepresentasikan kinerja dari sebuah model pembelajaran mesin [31]. Rumus *recall* (2.2), *accuracy* (2.3), *precision* (2.4), dan *F-1 Score* (2.5) dapat dilihat sebagai berikut.

Tabel *Confusion Matrix*

		p	**n**	**total**
actual value	**p′**	True Positive	False Negative	P′
	n′	False Positive	True Negative	N′
	total	P	N	

$$Recall = \frac{TP}{TP+FN} \tag{2.2}$$

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{2.3}$$

$$Precision = \frac{TP}{TP+FP} \tag{2.4}$$

$$F1Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{2.5}$$