

# **PENERAPAN**

## **KONSEP MACHINE LEARNING & DEEP LEARNING**

**PENULIS**

ARI MUZAKIR, S.KOM., M.CS

PROF. DR. KUSWORO ADI, S.SI., M.T

DR. RETNO KUSUMANINGRUM, S.SI., M.KOM

# **Penerapan Konsep *Machine Learning & Deep Learning***

(Pendekatan Ekspansi Semantik Untuk Klasifikasi Ujaran Kebencian)

Penulis

**Ari Muzakir, S.Kom., M.Cs**

**Prof. Dr. Kusworo Adi, S.Si., M.T**

**Dr. Retno Kusumaningrum, S.Si., M.Kom**

# **PENERAPAN KONSEP MACHINE LEARNING DAN DEEP LEARNING**

(Pendekatan Ekspansi Semantik Untuk Klasifikasi Ujaran Kebencian)

Oleh:

Ari Muzakir, S.Kom., M.Cs

Prof. Dr. Kusworo Adi, S.Si., M.T

Dr. Retno Kusumaningrum, S.Si., M.Kom



2024

**PENERAPAN KONSEP *MACHINE LEARNING*  
DAN *DEEP LEARNING*  
(Pendekatan Ekspansi Semantik Untuk Klasifikasi Ujaran  
Kebencian)**

Oleh:

Ari Muzakir, S.Kom., M.Cs

Prof. Dr. Kusworo Adi, S.Si., M.T

Dr. Retno Kusumaningrum, S.Si., M.Kom

Uk. 15,5cm x 23cm (xii + 150 hlm)

ISBN: 978-623-417-282-9

diterbitkan oleh



Anggota APPTI 003.151.1.3.2022

Anggota IKAPI 246/Anggota Luar Biasa/JTE/2022

Cetakan Pertama, Februari 2024

Dicetak oleh UNDIP Press

Isi diluar tanggung jawab percetakan

Hak Cipta dilindungi undang-undang

Dilarang mencetak dan menerbitkan sebagian atau seluruh isi buku ini dengan cara  
dan bentuk apapun tanpa seizin penulis dan penerbit

# KATA PENGANTAR

Dalam era digital yang semakin maju, internet telah menjadi salah satu media utama komunikasi global, memungkinkan koneksi global dan pertukaran informasi yang cepat. Sehingga peluang untuk mempelajari pengolahan bahasa alami (Natural Language Processing atau NLP) semakin terbuka lebar. Dengan akses yang luas ke data teks dari berbagai sumber internet, termasuk media sosial, artikel berita, dan lainnya, peluang untuk mengembangkan algoritma NLP yang lebih canggih dan efektif semakin meningkat.

Buku ini akan membimbing pembaca melalui serangkaian langkah-langkah dan teknik yang diperlukan untuk mengembangkan sistem klasifikasi teks ujaran kebencian yang efektif dan responsif dalam ekspansi semantik berbahasa Indonesia. Pembahasannya mencakup proses mulai dari akuisisi data hingga penggunaan model *pretrained*, serta memperkenalkan teknik-teknik canggih seperti augmentasi data atau *back-translation*, ekspansi kata, disambiguasi kata, dan *semantic similarity*. Lebih lanjut, studi kasus tentang pembangunan model kami sajikan secara singkat disertai tahapan yang diperlukan.

Dalam buku ini, algoritma *machine Learning* dan *deep learning* dijelaskan dengan pendekatan matematis yang sederhana, tetapi dapat diikuti. Pembaca disarankan memiliki pemahaman dasar tentang statistik, kalkulus, aljabar, dan pengenalan kecerdasan buatan. Penjelasan dalam buku ini bersifat deskriptif dan dilengkapi dengan langkah-langkah serta contoh yang membantu pemahaman. Walaupun tidak sempurna, buku ini diharapkan dapat menginspirasi pembaca untuk mengimplementasikan ilmu ini dalam proyek nyata, termasuk studi kasus deteksi ujaran kebencian. Teruslah berinovasi dan berkolaborasi untuk memperbaiki lingkungan online.

Semarang, November 2023

Penulis

# DAFTAR ISI

|   |             |
|---|-------------|
| <b>KATA PENGANTAR.....</b>  | <b>iii</b>  |
| <b>DAFTAR ISI.....</b>  | <b>iv</b>   |
| <b>DAFTAR GAMBAR.....</b>   | <b>vii</b>  |
| <b>DAFTAR TABEL .....</b>   | <b>viii</b> |
| <b>BAB I PENGENALAN DASAR .....</b>                                 | <b>1</b>    |
| 1.1 Kecerdasan Buatan .....   | 2           |
| 1.2 Konsep Belajar .....  | 4           |
| 1.3 Supervised Learning.....  | 6           |
| 1.4 Unsupervised Learning.....                                      | 8           |
| 1.5 Semi-supervised Learning.....                                   | 9           |
| 1.6 Reinforcement Learning.....                                     | 10          |
| <b>BAB II TEXT MINING.....</b>                                      | <b>13</b>   |
| 2.1 Data .....  | 14          |
| 2.2 Data Preprocessing .....  | 15          |
| 2.2.1 Case Folding .....  | 17          |
| 2.2.2 Data Cleansing.....   | 18          |
| 2.2.3 Normalization .....   | 20          |
| 2.2.4 Tokenization .....  | 21          |
| 2.2.5 Stop Words Removal .....                                      | 22          |
| 2.2.6 Stemming.....   | 24          |
| <b>BAB III REPRESENTASI TEKS .....</b>                              | <b>27</b>   |
| 3.1 Term Frequency and Inverse Document Frequency (TF-IDF)<br>..... | 28          |
| 3.2 Word2Vec .....  | 30          |
| 3.2.1 Continous Bag of Words (CBOW).....                            | 31          |
| 3.2.2 Skip-Gram Model .....   | 34          |
| 3.3 <i>Global Vectors for Word Representation (GloVe)</i> .....     | 36          |

|   |           |
|---|-----------|
| 3.4 FastText .....  | 40        |
| 3.5 <i>Embeddings from Language Model (ELMo)</i> .....                          | 42        |
| 3.6 <i>Bidirectional Encoder Representations from Transformers (BERT)</i> ..... | 45        |
| 3.6.1 Definisi .....  | 45        |
| 3.6.2 Representasi <i>Input</i> BERT .....                                      | 50        |
| 3.6.3 <i>Pre-training</i> BERT .....  | 51        |
| 3.6.4 <i>Fine-tuning</i> BERT .....   | 54        |
| 3.6.5 <i>Pre-trained</i> BERT .....   | 55        |
| <b>BAB IV ALGORITMA KLASIFIKASI .....</b>                                       | <b>57</b> |
| 4.1 <i>Machine Learning</i> .....   | 57        |
| 4.1.1 Naive Bayes (NB).....   | 58        |
| 4.1.2 Random Forest Decision Tree (RFDT) .....                                  | 67        |
| 4.1.3 Support Vector Machine (SVM) .....  | 73        |
| 4.2 Deep Learning .....   | 74        |
| 4.2.1 Bidirectional Long Short Term Memory Network (BiLSTM) .....               | 75        |
| 4.2.2 Recurrent Neural Network (RNN).....                                       | 80        |
| 4.2.3 Convolutional Neural Network (CNN) .....                                  | 83        |
| 4.2.4 Bidirectional Gated Recurrent Unit (BiGRU) .....                          | 86        |
| <b>BAB V METRIK EVALUASI .....</b>  | <b>89</b> |
| 5.1 Confusion Matrix .....  | 89        |
| 5.2 <i>Micro-Averaged F1-measure</i> .....                                      | 93        |
| 5.3 Area Under the ROC Curve (AUC – ROC).....                                   | 94        |
| <b>BAB VI STUDI KASUS: EKSPANSI SEMANTIK .....</b>                              | <b>97</b> |
| 6.1 Back-Translation .....  | 98        |
| 6.2 Disambiguasi Kata .....   | 101       |
| 6.3 Ekspansi Teks.....  | 105       |
| 6.4 Kemiripan Semantik.....   | 114       |

|  |            |
|--|------------|
| <b>BAB VII STUDI KASUS: KLASIFIKASI UJARAN KEBENCIAN .....</b> | <b>120</b> |
| 7.1 Data .....   | 121        |
| 7.2 Data Preprocessing .....                                   | 122        |
| 7.3 Pembagian Data.....  | 125        |
| 7.4 Pembentukan Model Deteksi Ujaran Kebencian.....            | 126        |
| 7.4.1 Pemahaman tentang <i>Pre-trained</i> Model .....         | 127        |
| 7.4.2 Memuat Model <i>Embedding</i> BERT dan Konfigurasi ...   | 127        |
| 7.4.3 <i>Fine-tuning</i> (Opsional) .....                      | 129        |
| 7.4.4 Pelatihan Model .....                                    | 129        |
| 7.4.5 Penyimpanan Model .....                                  | 133        |
| 7.4.6 Evaluasi Model .....                                     | 134        |
| <b>BAB VIII PENUTUP .....</b>                                  | <b>139</b> |
| <b>DAFTAR PUSTAKA .....</b>                                    | <b>141</b> |
| <b>BIODATA PENULIS.....</b>                                    | <b>149</b> |

# DAFTAR GAMBAR

|  |     |
|--|-----|
| <b>Gambar 2. 1</b> Tahapan Data Preprocessing .....  | 17  |
| <b>Gambar 3. 1</b> Arsitektur Model <i>Continuous Bag of Words</i> .....   | 32  |
| <b>Gambar 3. 2</b> Arsitektur Model Skip-Gram.....   | 35  |
| <b>Gambar 3. 3</b> <i>Encoder</i> (kiri) dan <i>Decoder</i> (kanan).....   | 48  |
| <b>Gambar 3. 4</b> Proses prapelatihan pada BERT .....   | 50  |
| <b>Gambar 3. 5</b> Representasi Input pada BERT.....   | 51  |
| <b>Gambar 4. 1</b> Struktur Decision Tree.....   | 68  |
| <b>Gambar 4. 2</b> Decision Tree pada Pemilihan Ponsel .....   | 70  |
| <b>Gambar 4. 3</b> Gambaran kinerja algoritma <i>traditional machine Learning</i> dan <i>deep learning algorithm</i> ..... | 74  |
| <b>Gambar 4. 4</b> Arsitektur Umum dari LSTM .....   | 77  |
| <b>Gambar 4. 5</b> Arsitektur Umum BiLSTM .....  | 78  |
| <b>Gambar 4. 6</b> Bentuk konseptual paling sederhana RNN.....   | 80  |
| <b>Gambar 4. 7</b> Konsep Recurrent Neural Network .....   | 81  |
| <b>Gambar 4. 8</b> Konsep feed forward pada RNN .....  | 82  |
| <b>Gambar 4. 9</b> Konsep backpropagation through time.....  | 83  |
| <b>Gambar 4. 10</b> Arsitektur CNN untuk klasifikasi teks.....   | 84  |
| <b>Gambar 4. 11</b> Struktur unit GRU .....  | 87  |
| <b>Gambar 4. 12</b> Struktur diagram BiGRU .....   | 88  |
| <b>Gambar 5. 1</b> Hasil Kurva AUC-ROC.....  | 96  |
| <b>Gambar 6. 1</b> Proses Back-Translation .....   | 98  |
| <b>Gambar 6. 2</b> Proses Disambiguasi Kata .....  | 102 |
| <b>Gambar 6. 3</b> Proses Ekspansi Teks.....   | 106 |
| <b>Gambar 6. 4</b> Proses Semantic Similarity .....  | 114 |
| <b>Gambar 7. 1</b> Alir Proses Klasifikasi Ujaran Kebencian .....  | 121 |
| <b>Gambar 7. 2</b> Data dalam Format CSV.....  | 125 |
| <b>Gambar 7. 3</b> Pembagian Data Latih dan Data Uji .....   | 126 |
| <b>Gambar 7. 4</b> Pilihan Model <i>Pre-trained</i> IndoBERT .....   | 127 |
| <b>Gambar 7. 5</b> Konfigurasi parameter untuk Fine-Tunning .....  | 129 |
| <b>Gambar 7. 6</b> Contoh Hasil Model Dari Proses Pelatihan .....  | 134 |
| <b>Gambar 7. 7</b> Alir Pengujian dan Evaluasi Model.....  | 136 |

## DAFTAR TABEL

|   |     |
|---|-----|
| <b>Tabel 2. 1</b> Contoh Case Folding .....                   | 17  |
| <b>Tabel 2. 2</b> Contoh Data Cleansing.....                  | 19  |
| <b>Tabel 2. 3</b> Contoh Normalization .....                  | 20  |
| <b>Tabel 2. 4</b> Contoh Tokenization .....                   | 22  |
| <b>Tabel 2. 5</b> Contoh Stop-word Removal.....               | 23  |
| <b>Tabel 2. 6</b> Contoh Proses Stemming .....                | 25  |
| <b>Tabel 3. 1</b> Perbandingan Model Skip-Gram dan CBOW ..... | 36  |
| <b>Tabel 3. 2</b> Contoh Matriks Kemunculan Kata.....         | 38  |
| <b>Tabel 3. 3</b> Contoh Ulasan .....                         | 44  |
| <b>Tabel 4. 1</b> Data Cuaca .....                            | 61  |
| <b>Tabel 5. 1</b> Confusion Matrix .....                      | 89  |
| <b>Tabel 5. 2</b> Contoh Confusion Matrix.....                | 92  |
| <b>Tabel 7. 1</b> Contoh Studi Kasus Confusion Matrix .....   | 136 |

# **BAB I**

## **PENGENALAN DASAR**

Selamat datang dalam bab pengenalan dasar tentang *machine Learning* dan deep learning. Mungkin istilah-istilah ini sudah tidak asing bagi pembaca, baik dari pengalaman kuliah di jurusan Sistem Informasi, Statistik, Matematika, Teknik Informatika atau Ilmu Komputer, maupun dari bidang lain. Terkadang, ada kebingungan mengenai hubungan antara *machine Learning* dan deep learning. Penting untuk dicatat bahwa deep learning adalah salah satu cabang dari *machine Learning*. Dalam bab ini, kami akan membahas konsep dasar dan prinsip utama yang menjadi dasar dari kedua bidang ini.

Dalam bab ini, kami akan merinci dasar-dasar *machine Learning*, termasuk pemahaman tentang konsep inti dan bagaimana algoritma *machine Learning* dapat digunakan untuk memahami dan memproses data. Selain itu, kami juga akan menjelaskan bahwa deep learning, yang saat ini menjadi fokus utama dalam *machine Learning*, merupakan metode yang menggunakan jaringan saraf tiruan berlapis dalam proses pembelajarannya. Hal ini memungkinkan deep learning untuk mengatasi masalah kompleks, seperti pengenalan gambar atau bahasa alami.

Kami akan membahas konsep dan prinsip dasar *machine Learning*, yang menjadi fondasi bagi pemahaman lebih lanjut tentang deep learning. Ini adalah langkah awal yang penting untuk memahami bagaimana mesin dapat belajar dari data dan membuat prediksi serta keputusan cerdas. Dengan memahami prinsip-prinsip ini, pembaca akan siap untuk menjelajahi konsep deep learning yang lebih mendalam dalam bab-bab berikutnya.

## **1.1 Kecerdasan Buatan**

Kecerdasan buatan adalah bidang yang berkembang pesat dalam teknologi dan komputer. Ini adalah dasar dari pemahaman yang lebih baik tentang bagaimana komputer dapat belajar dan berperilaku mirip dengan manusia. Di subbab ini, kita akan menjelaskan konsep dasar dan sejarah singkat kecerdasan buatan, serta mengapa ini menjadi topik yang begitu penting dalam pengembangan sistem deteksi ujaran kebencian.

Kecerdasan buatan adalah bidang ilmu yang berfokus pada pengembangan algoritma dan sistem komputer yang mampu melakukan tugas-tugas yang memerlukan pemahaman dan belajar. Dalam bidang keilmuan kecerdasan buatan, tujuan utamanya adalah menciptakan agen komputasi yang memiliki kemampuan untuk mengeksekusi tugas-tugas yang pada umumnya memerlukan kecerdasan manusia. Namun, penting untuk memahami bahwa dalam konteks ini, yang dimaksudkan adalah aproksimasi kecerdasan manusia. Meskipun hewan-hewan juga memiliki tingkat kecerdasan yang bervariasi, fokus utama kecerdasan buatan adalah mendekati tingkat kecerdasan manusia.

Kecerdasan manusia adalah konsep yang sangat kompleks dan sulit untuk didefinisikan secara tuntas. Ini melibatkan banyak aspek, termasuk nalar (logika), kemampuan berbahasa, pemahaman seni, dan banyak lagi. Karena kecerdasan manusia memiliki banyak dimensi yang berbeda, pendekatan yang sering digunakan adalah memecahnya menjadi sub-bidang yang lebih spesifik, mengikuti prinsip "divide and conquer". Dengan mendekati masalah ini dalam sub-bidang yang lebih kecil, kita dapat mengambil langkah konkret untuk mencapai pemahaman lebih dalam tentang kecerdasan manusia.

Perlu dicatat bahwa meskipun kecerdasan buatan telah mencapai pencapaian yang mengesankan, para peneliti hingga saat ini masih belum sepenuhnya memahami apa yang membuat manusia

menjadi cerdas, dan bagaimana manusia dapat mencapai tingkat kecerdasan ini. Oleh karena itu, ilmu kecerdasan buatan adalah disiplin ilmu yang sangat interdisipliner, menggabungkan pengetahuan dari berbagai bidang, termasuk psikologi, linguistik, ilmu komputer, biologi, dan banyak lagi. Kolaborasi lintas disiplin menjadi kunci dalam upaya untuk memahami dan mendekati kecerdasan manusia. Sebagai contoh, mari kita lihat beberapa ilustrasi berikut:

**Ilustrasi 1: Algoritma Pencarian dan Deteksi Hoaks**

Bayangkan Anda adalah pengguna media sosial. Ketika Anda menjelajahi berita di aliran sosial Anda, algoritma kecerdasan buatan bekerja di latar belakang. Misalnya, ketika Anda membaca sebuah artikel berjudul "Percobaan Vaksin COVID-19 Terbukti Berbahaya," algoritma tersebut memeriksa artikel tersebut dan mencocokkannya dengan data dari sumber tepercaya. Algoritma ini menggunakan pembelajaran mesin dan analisis teks untuk mengidentifikasi potensi hoaks atau informasi yang salah.

**Ilustrasi 2: Filter Spam di Email**

Ketika Anda membuka kotak surat elektronik Anda, algoritma kecerdasan buatan juga berperan dalam menangkal email spam. Algoritma ini memindai setiap pesan yang masuk dan menilai kemungkinan pesan tersebut adalah spam atau bukan. Dengan demikian, algoritma ini membantu Anda menjaga kotak masuk Anda dari berbagai jenis pesan yang mencoba menyebarluaskan hoaks atau tautan berbahaya.

**Ilustrasi 3: Rekognisi Gambar dan Video**

Di era media sosial, berbagi gambar dan video adalah hal yang umum. Algoritma kecerdasan buatan digunakan dalam aplikasi pengenalan gambar dan video. Misalnya, ketika Anda membagikan gambar atau video yang tampak mencurigakan atau mengandung informasi palsu, algoritma ini dapat memberikan tanda peringatan atau bahkan mengidentifikasi elemen-elemen yang meragukan dalam konten tersebut.

#### **Ilustrasi 4: Chatbot Pemantau Informasi**

Beberapa situs berita dan organisasi menggunakan chatbot dengan kecerdasan buatan untuk membantu pengguna memeriksa informasi. Misalnya, jika Anda memiliki pertanyaan tentang suatu berita atau artikel yang Anda temui, Anda dapat mengajukan pertanyaan kepada chatbot tersebut. Chatbot akan menggunakan kemampuan pemahaman bahasa alaminya untuk memberikan informasi tentang apakah berita tersebut sudah terverifikasi atau belum.

Dalam semua ilustrasi ini, algoritma kecerdasan buatan bertindak sebagai perangkat pertahanan penting dalam mengidentifikasi hoaks dan menyaring informasi yang tidak benar. Algoritma tersebut membantu dalam memastikan bahwa pengguna memiliki akses ke informasi yang akurat dan dapat menghindari penyebaran berita palsu yang dapat merugikan masyarakat secara luas.

## **1.2 Konsep Belajar**

Bayangkan situasi di mana Anda berada di suatu negara asing, di mana Anda tidak akrab dengan norma-norma yang berlaku di sana. Bagaimana Anda akan berusaha agar dapat menjadi individu yang "normal" dalam lingkungan tersebut? Tentunya, Anda perlu belajar. Anda harus mengamati perilaku dan tindakan orang-orang di negara tersebut, sehingga seiring berjalaninya waktu, Anda dapat memahami norma-norma yang berlaku. Proses belajar adalah cara kita memperoleh pengetahuan dan keterampilan; melalui pengamatan, latihan, dan penyesuaian perilaku berdasarkan pengalaman.

Namun, dalam dunia *machine Learning*, yang belajar bukanlah makhluk hidup, melainkan mesin. Definisi konsep belajar dalam *machine Learning* memiliki latar belakang yang agak berbeda. Dalam konteks ini, belajar diartikan sebagai perubahan

tingkah laku mesin berdasarkan pengalaman atau data yang diberikan. Ini bertujuan agar mesin dapat menjadi lebih baik dalam tugas yang diberikan. Dalam *machine Learning*, belajar adalah proses di mana mesin menyesuaikan konfigurasi parameter, yang mengendalikan perilaku mesin, berdasarkan data yang diberikan oleh lingkungan atau input yang diberikan. Dengan kata lain, mesin belajar dengan meningkatkan kinerjanya berdasarkan pengalaman yang diperoleh dari data dan lingkungan sekitarnya, mirip dengan cara manusia belajar dari pengamatan dan pengalaman di lingkungan baru.

Konsep belajar adalah inti dari *machine Learning*, yang merupakan salah satu cabang utama dalam kecerdasan buatan. Dalam *machine Learning*, belajar merujuk pada kemampuan komputer atau mesin untuk mengidentifikasi pola dalam data, mengekstrak pengetahuan dari pengalaman, dan meningkatkan kinerjanya seiring berjalannya waktu. Proses belajar ini mirip dengan cara manusia belajar, meskipun terjadi dalam konteks yang lebih matematis dan algoritmik.

Ketika mesin belajar, data menjadi kunci. Data tersebut dapat berupa berbagai informasi yang digunakan sebagai input, seperti gambar, teks, atau angka. Mesin menggunakan algoritma untuk menganalisis data ini, mengidentifikasi pola, dan membuat prediksi atau pengambilan keputusan. Semakin banyak data yang diberikan kepada mesin, semakin baik mesin dapat belajar dan meningkatkan performanya. Dengan kata lain, semakin banyak pengalaman yang dimiliki mesin, semakin baik kemampuannya dalam memahami dan menangani tugas-tugas yang kompleks.

Hubungan antara konsep belajar dalam *machine Learning* dan manusia adalah bahwa *machine Learning* merupakan upaya untuk meniru atau memodelkan cara manusia belajar. Ini adalah salah satu cara di mana komputer dapat menjadi cerdas dan mandiri dalam menghadapi berbagai tugas, mulai dari pengenalan wajah

hingga prediksi harga saham. Proses belajar dalam *machine Learning* mencerminkan upaya untuk menerapkan intuisi dan kecerdasan manusia ke dalam algoritma komputer, sehingga mesin dapat memahami dan berinteraksi dengan dunia sekitarnya dengan lebih baik.

### 1.3 Supervised Learning

Supervised learning adalah salah satu paradigma utama dalam *machine Learning*, di mana proses belajarannya bergantung pada serangkaian contoh input-output yang benar, yang bertindak sebagai supervisor atau pelatih bagi algoritma. Dalam supervised learning, tujuan utama adalah melatih mesin untuk mencapai output yang diinginkan, seperti kelas atau label yang telah diketahui berdasarkan contoh yang diberikan. Sebagai contoh, dalam kasus pengenalan gambar, mesin dapat dilatih dengan berbagai gambar dan label yang mengidentifikasi objek dalam gambar tersebut, sehingga mesin dapat memprediksi objek dalam gambar baru yang belum pernah dilihat sebelumnya.

Supervised learning sering digunakan dalam berbagai aplikasi di mana data historis digunakan untuk memprediksi kemungkinan peristiwa di masa depan. Sebagai ilustrasi, perusahaan kartu kredit dapat menggunakan supervised learning untuk mendeteksi potensi penipuan dalam transaksi. Dalam hal ini, mesin belajar dari data transaksi yang sah dan transaksi yang curang untuk mengenali pola-pola yang mengindikasikan potensi penipuan. Demikian pula, perusahaan asuransi dapat menggunakan supervised learning untuk memprediksi klaim yang kemungkinan akan diajukan oleh pelanggan berdasarkan data historis klaim yang telah diajukan.

Algoritma supervised learning sendiri dapat dibagi menjadi beberapa jenis tergantung pada tujuan utamanya. Dua jenis utama adalah klasifikasi dan prediksi numerik (regresi). Dalam klasifikasi,

algoritma digunakan untuk mengkategorikan data ke dalam kelas atau label yang telah ditentukan sebelumnya. Contoh algoritma klasifikasi termasuk Logistic Regression (LR), Decision Trees (DT), Random Forest (RF), K-Nearest Neighbors (KNN), Support Vector Machines (SVM), dan Neural Networks (NNs). Di sisi lain, prediksi numerik atau regresi digunakan untuk memprediksi nilai numerik atau kontinu. Algoritma regresi meliputi Linear Regression, Decision Trees, Neural Networks, SVM, dan lainnya.

Dalam kedua kasus ini, supervised learning memungkinkan mesin untuk memahami dan memanfaatkan pola-pola yang ada dalam data historis untuk membuat prediksi yang berguna dalam berbagai aplikasi, dari klasifikasi email spam hingga prediksi harga saham. Misalnya, kita dapat menggunakan algoritma regresi untuk memprediksi harga rumah berdasarkan berbagai atribut, seperti luas tanah, jumlah kamar, lokasi, dan sebagainya. Dalam hal ini, kita memiliki data historis harga rumah beserta atribut-atributnya. Algoritma regresi, seperti Linear Regression atau Random Forest, akan mempelajari hubungan antara atribut-atribut ini dan harga rumah. Dengan demikian, ketika kita memiliki data atribut untuk rumah yang akan dijual, algoritma ini dapat memprediksi harga rumah tersebut.

Sebagai contoh lain dalam aplikasi bisnis, misalnya sebuah bank menggunakan supervised learning untuk mendeteksi potensi penipuan dalam transaksi kartu kredit. Mereka memiliki data historis transaksi yang mencakup transaksi yang sah dan transaksi penipuan. Algoritma supervised learning, seperti Neural Networks atau Decision Trees, dilatih menggunakan data ini. Algoritma tersebut mempelajari pola-pola yang mungkin mengindikasikan penipuan dalam transaksi, seperti pola penggunaan kartu di lokasi yang tidak biasa atau pola pembelian yang tidak biasa. Ketika transaksi baru dilakukan, algoritma ini dapat memeriksa apakah ada indikasi penipuan dan memberikan peringatan jika diperlukan.

## 1.4 Unsupervised Learning

Unsupervised learning adalah paradigma *machine Learning* yang berbeda dari supervised learning. Dalam unsupervised learning, algoritma diberikan sejumlah sampel input, tetapi perbedaannya adalah bahwa sampel tersebut tidak memiliki label atau output yang benar yang sesuai. Dengan kata lain, tidak ada supervisor yang memberikan petunjuk tentang kinerja atau output yang diharapkan. Ini membuat unsupervised learning sangat berguna dalam situasi di mana kita ingin mengeksplorasi data untuk menemukan struktur internalnya tanpa panduan eksternal.

Salah satu tujuan utama dari teknik pembelajaran unsupervised adalah untuk menggali struktur dalam data. Dalam konteks ini, salah satu contoh paling umum adalah pengelompokan atau klustering. Klustering adalah proses mengidentifikasi kesamaan dalam data, mengelompokkan data ke dalam kelompok yang mirip berdasarkan karakteristik atau atribut tertentu. Sebagai contoh, pertimbangkan sebuah toko e-commerce yang ingin memahami preferensi pelanggan mereka. Mereka mungkin menggunakan algoritma klustering seperti K-Means Clustering untuk mengelompokkan pelanggan ke dalam segmen berdasarkan pola pembelian mereka. Ini memungkinkan toko untuk menyusun strategi pemasaran yang lebih sesuai dan mengidentifikasi peluang untuk meningkatkan penjualan.

Tidak hanya klustering, dalam unsupervised learning juga ada jenis lain yang disebut Association, yang digunakan untuk mengidentifikasi asosiasi atau hubungan antara item dalam data. Sebagai contoh, pertimbangkan sebuah basis data transaksi toko ritel. Algoritma Association Rules dapat digunakan untuk mengidentifikasi hubungan antara produk yang sering dibeli bersama, seperti kenyataan bahwa ketika pelanggan membeli roti, mereka cenderung juga membeli mentega. Ini memungkinkan toko

untuk menyesuaikan penataan produk atau menyusun penawaran khusus yang lebih relevan untuk pelanggan mereka.

Unsupervised learning memberikan kebebasan eksplorasi data yang lebih luas, di mana algoritma berusaha untuk mengungkap struktur dan hubungan internal dalam data tanpa arahan eksternal. Itu adalah alat yang sangat berguna dalam analisis data, pemrosesan bahasa alami, pengelompokan, serta berbagai aplikasi lain di mana struktur data perlu diungkapkan tanpa panduan eksternal yang jelas.

## 1.5 Semi-supervised Learning

Semi-supervised learning adalah suatu paradigma dalam *machine Learning* yang memungkinkan algoritma untuk memanfaatkan data yang memiliki label serta data yang tidak memiliki label selama proses pembelajaran. Dalam hal ini, "berlabel" berarti bahwa setiap data dalam kumpulan data memiliki informasi yang menunjukkan output yang benar atau label yang sesuai. Sementara itu, "tidak berlabel" berarti bahwa data tersebut tidak memiliki label atau output yang sesuai.

Keunikian dari semi-supervised learning adalah bahwa algoritma mampu menggabungkan informasi yang diperoleh dari data berlabel dengan data yang lebih melimpah yang tidak berlabel. Hal ini memungkinkan sistem untuk belajar dengan efisiensi yang lebih tinggi daripada metode yang sepenuhnya bergantung pada data berlabel. Dalam kasus di mana data berlabel sangat terbatas, semisupervised learning menjadi sangat berguna untuk memaksimalkan hasil pembelajaran.

Sebagai contoh dunia nyata, pertimbangkan pengenalan wajah seseorang melalui kamera webcam atau ponsel pintar. Dalam pengenalan wajah, kita dapat memiliki sejumlah data gambar wajah yang telah diberi label dengan nama orang yang sesuai (data berlabel). Namun, ada begitu banyak varian wajah dan situasi yang

berbeda sehingga memungkinkan untuk mengumpulkan sejumlah besar data gambar wajah yang tidak memiliki label (data tidak berlabel).

Dalam hal ini, algoritma semi-supervised learning dapat memanfaatkan data berlabel yang terbatas untuk mengenali pola-pola umum dalam wajah manusia dan kemudian menggabungkan pengetahuan tersebut dengan data wajah yang tidak berlabel. Dengan cara ini, sistem dapat menjadi lebih efisien dalam mengidentifikasi dan mengklasifikasikan wajah orang yang mungkin belum terlihat sebelumnya.

## 1.6 Reinforcement Learning

Reinforcement learning (pembelajaran penguatan) adalah paradigma dalam *machine Learning* yang memiliki dua komponen utama, yaitu agen (agent) dan lingkungan (environment). Dalam algoritma ini, agen memiliki tugas untuk belajar sendiri bagaimana ia harus bertindak di hadapan lingkungannya untuk mencapai tujuannya. Yang membedakan reinforcement learning dari metode lain adalah bahwa tidak ada kumpulan data yang diberikan sebelumnya, seperti pada supervised learning atau unsupervised learning.

Tujuan utama dari reinforcement learning adalah untuk menggunakan informasi yang diperoleh melalui interaksi dengan lingkungan guna mengambil tindakan yang akan memaksimalkan hasil (reward) dan meminimalkan risiko. Algoritma ini melibatkan proses belajar berulang di mana agen terus-menerus memperbarui strategi atau keputusan yang diambil.

Proses reinforcement learning terjadi dalam beberapa tahap. Pertama, agen mengamati data input dari lingkungan. Setelah itu, agen membuat keputusan atau tindakan tertentu sebagai respons terhadap data input tersebut. Setelah tindakan diambil, agen

menerima reward atau penguatan dari lingkungan sebagai umpan balik atas tindakan yang diambil. Selanjutnya, agen akan kembali mengamati data input dan memproses pengambilan keputusan. Proses ini berulang terus dengan tambahan penguatan dari lingkungan, yang membantu agen untuk memperbaiki dan mengoptimalkan strateginya.

Contoh umum dari reinforcement learning adalah penggunaan algoritma ini dalam pengembangan agen cerdas dalam permainan video. Dalam kasus ini, agen (misalnya, agen yang mengendalikan karakter dalam permainan) berinteraksi dengan lingkungan (permainan itu sendiri) dan belajar dari hasil tindakan yang diambilnya. Agen memaksimalkan skor atau reward dengan mencoba berbagai tindakan dan memperbaiki strateginya seiring berjalannya waktu. Dengan demikian, reinforcement learning memungkinkan pembelajaran interaktif yang terus menerus dan pengoptimalan tindakan untuk mencapai tujuan tertentu dalam konteks yang dinamis.



## **BAB II**

### **TEXT MINING**

Penambangan teks, juga dikenal sebagai Text Mining, adalah suatu proses yang melibatkan analisis mendalam dan intensif terhadap kumpulan dokumen, di mana pengguna berinteraksi dengan data teks seiring berjalananya waktu menggunakan berbagai alat analisis. Tujuan utama dari penambangan teks adalah untuk mengekstrak informasi berharga dari sumber data yang terdiri dari dokumen-dokumen. Proses ini dilakukan dengan cara mengidentifikasi dan mengeksplorasi pola-pola menarik yang tersembunyi dalam teks.

Dalam konteks text mining, sumber data yang menjadi fokus adalah koleksi dokumen yang dapat mencakup segala jenis materi tertulis, seperti laporan, artikel, tweet, email, dan banyak lagi. Tantangannya terletak pada fakta bahwa data ini terstruktur dalam format teks yang tidak teratur dan tidak terformat secara seragam. Oleh karena itu, penambangan teks berbeda secara fundamental dengan sistem penambangan data tradisional yang cenderung berfokus pada data yang telah diformalkan dalam basis data.

Text mining menghadirkan peluang penting untuk mengeksplorasi data teks yang sangat beragam dan tidak terstruktur, yang dapat menghasilkan wawasan berharga. Dalam bab ini, kita akan membahas komponen-komponen penting dari text mining, seperti text preprocessing, yang mencakup case folding, data cleansing, normalization, tokenization, stop word removal, dan stemming, serta berbagai alat analisis yang digunakan untuk menggali informasi berharga dari kumpulan dokumen tersebut.

## 2.1 Data

Data memiliki peran yang semakin penting dalam era saat ini, di mana informasi tersebar begitu meluas. Mengelola dan memproses data secara manual telah menjadi suatu tugas yang semakin tidak praktis mengingat volume data yang sangat besar. Berbagai proses pemrosesan data telah menjadi lebih penting daripada sebelumnya, seperti kategorisasi, peringkasan dokumen, ekstraksi informasi, dan rekomendasi produk berdasarkan data transaksi. Semua ini mencerminkan pentingnya data dalam pemahaman, pengambilan keputusan, dan kemajuan teknologi saat ini.

*Machine Learning*, salah satu cabang dari kecerdasan buatan, memiliki dua tujuan utama. Pertama, memungkinkan kita untuk memprediksi peristiwa di masa depan yang belum diamati sebelumnya. Kedua, membantu dalam penemuan pengetahuan atau struktur yang belum diketahui sebelumnya. Ini adalah dua aspek yang sangat terkait. Sebagai contoh, kita dapat menggunakan pengetahuan yang sudah ada untuk memprediksi bahwa cara menggunakan pulpen mirip dengan cara menggunakan pensil, meskipun kita belum pernah menggunakan pulpen sebelumnya. Di sisi lain, knowledge discovery membantu kita menemukan bahwa aturan atau pola yang mendasari penggunaan pulpen dan pensil adalah sama, meskipun kita belum pernah menggunakan pulpen sebelumnya.

Data memainkan peran kunci dalam mencapai tujuan ini. Kami mengumpulkan data dalam bentuk sampel, dan kemudian menggunakan data ini untuk membuat model. Model ini kemudian digunakan untuk menggeneralisasikan aturan atau pola dalam data, sehingga kita dapat menggunakannya untuk mendapatkan informasi yang berharga atau membuat keputusan yang lebih baik. Dengan kata lain, data adalah bahan mentah yang menjadi dasar bagi model *machine Learning* untuk mengungkap pengetahuan yang ada dalam

data tersebut. Dalam dunia yang semakin didorong oleh data, pemahaman dan penerapan data yang efektif menjadi kunci untuk memahami dunia sekitar kita dan membuat keputusan yang lebih cerdas.

## 2.2 Data Preprocessing

Text preprocessing atau data preprocessing adalah cabang dari ilmu komputer yang berkaitan dengan analisis, manipulasi, dan pemahaman teks dalam bahasa manusia. Ini adalah bidang yang sangat penting dalam era digital, di mana jumlah data teks yang dihasilkan setiap hari semakin meningkat. Text preprocessing memungkinkan komputer untuk berinteraksi dengan teks dalam cara yang bermanfaat, seperti pemahaman isi teks, klasifikasi dokumen, ekstraksi informasi, penerjemahan bahasa, analisis sentimen, dan banyak lagi.

Pemrosesan bahasa alami (Natural Language Processing atau NLP) adalah subdomain dari pemrosesan teks yang berfokus pada pemahaman bahasa manusia dalam bentuk yang dapat diproses oleh komputer. NLP melibatkan berbagai tugas yang mencakup pemahaman sintaksis dan semantik, penerjemahan bahasa, pemrosesan percakapan, analisis sentimen, dan banyak lagi. NLP memungkinkan komputer untuk mengenali dan memproses bahasa manusia secara lebih alami, yang membuka pintu untuk aplikasi seperti asisten virtual, chatbot, analisis teks otomatis, dan masih banyak lagi.

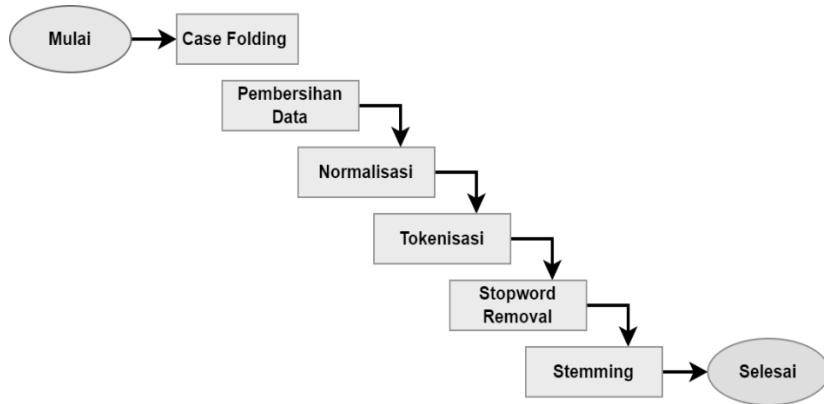
Penting untuk diingat bahwa text preprocessing adalah fondasi utama bagi NLP. Data teks harus dianalisis, dibersihkan, dan diproses sebelum dapat dijalani oleh algoritma NLP. Dalam hal ini, pemrosesan teks mencakup langkah-langkah seperti tokenisasi (pemisahan teks menjadi kata-kata atau frasa), normalisasi (mengubah kata menjadi bentuk dasarnya), penghilangan tanda

baca, dan sebagainya. Semua ini bertujuan untuk mengoptimalkan data teks agar dapat dipahami dan diproses dengan lebih efisien oleh algoritma NLP.

Text preprocessing dan NLP memiliki peran yang krusial dalam berbagai aplikasi di dunia nyata, seperti dalam pencarian online, analisis berita, deteksi ujaran kebencian, pemodelan bahasa, dan banyak lagi. Dengan kemajuan teknologi, kemampuan pemrosesan teks dan NLP terus berkembang, membantu kita untuk lebih memahami dan berinteraksi dengan bahasa manusia dalam bentuk digital.

Tahapan dalam text preprocessing seperti pada Gambar 2.1 umumnya yaitu case folding, data cleansing, normalization, tokenization, dan stemming. *Case folding* adalah langkah yang mengubah semua huruf kapital dalam kata menjadi huruf kecil, termasuk kata-kata yang muncul di awal kalimat, judul, dan kata-kata yang secara konsisten ditulis dengan huruf kapital (Manning, 2009). Data cleansing adalah tahapan yang dilakukan untuk mengeliminasi elemen kata yang tidak relevan dengan tujuan mengurangi gangguan atau noise pada dataset (Riyaddulloh dan Romadhony, 2021). Normalisasi teks adalah konsep yang merujuk pada proses mengubah format teks agar sesuai dengan tujuan tertentu. Penggunaan normalisasi teks menjadi sangat signifikan, karena dalam teks atau bahan bacaan seringkali terdapat beragam permasalahan, mulai dari penggunaan kata yang ambigu, variasi bahasa, hingga perubahan makna kalimat dari arti aslinya (Riyaddulloh dan Romadhony, 2021). Tokenisasi adalah langkah di mana sebuah string input dibagi menjadi unit kata berdasarkan batasannya (Luthfiarta dkk., 2013). Dengan lebih rinci, proses ini mengidentifikasi dan memisahkan setiap karakter dalam sebuah kalimat menjadi unit-unit kata yang terpisah. Tujuan utama dari Tokenisasi adalah untuk mengeksplorasi kata-kata yang terdapat dalam suatu kalimat (Verma dkk., 2014). Stemming merupakan

bagian integral dalam sistem *Information Retrieval* yang mengubah kata-kata dalam sebuah dokumen ke bentuk akarnya dengan menerapkan aturan-aturan tertentu (Tala, 2003).



**Gambar 2. 1** Tahapan Data Preprocessing

### 2.2.1 Case Folding

Case folding adalah langkah yang mengubah semua huruf kapital dalam kata menjadi huruf kecil, termasuk kata-kata yang muncul di awal kalimat, judul, dan kata-kata yang secara konsisten ditulis dengan huruf kapital (Manning, 2009). Proses ini sangat penting mengingat ketidak konsistennan dalam penggunaan huruf kapital yang sering terjadi pada tweet yang dibuat oleh pengguna Twitter. Contoh dalam case folding adalah seperti pada Tabel 2.1 yang menunjukkan bentuk teks asli sebelum proses dan setelah proses case folding.

**Tabel 2. 1** Contoh Case Folding

| No. | Teks Asli  | Hasil Case Folding   |
|-----|--|--|
| 1   | Proses Case Folding sangat<br>Penting dalam Analisis Teks. | proses case folding sangat<br>penting dalam analisis teks. |

|   |   |   |
|---|---|---|
| 2 | Data Cleansing Merupakan Tahapan Penting dalam DATA Mining. | data cleansing merupakan tahapan penting dalam data mining. |
| 3 | Semua File akan Di-Download pada Tanggal 20 JANUARI 2023.   | semua file akan di-download pada tanggal 20 januari 2023.   |

Fungsi yang digunakan untuk melakukan *case folding* adalah fungsi *lower()* yang sudah disediakan oleh library Python. Tujuan utama dari proses case folding adalah untuk meningkatkan akurasi sistem. Selain itu, penerapan case folding untuk menciptakan keseragaman dalam penggunaan huruf kapital pada data teks yang seringkali tidak memiliki konsistensi dalam penggunaannya (Prihatini, 2016). Dengan menerapkan case folding, dokumen-dokumen teks yang awalnya dapat memiliki beragam variasi dalam penggunaan huruf kapital akan menjadi lebih seragam, mempermudah proses analisis dan pengolahan data dalam konteks text mining.

### 2.2.2 Data Cleansing

Data cleansing atau pembersihan data adalah tahapan yang sangat penting dalam analisis data, terutama dalam konteks data teks. Tujuan utamanya adalah untuk mengeliminasi elemen kata yang tidak relevan, yang sering disebut sebagai gangguan atau noise, yang ada dalam dataset (Riyaddulloh dan Romadhony, 2021). Proses pembersihan data ini mencakup berbagai aspek, termasuk penghapusan tautan, nama pengguna (@username), tanda pagar (#), simbol-simbol, emotikon, tanda baca, dan angka. Tujuan dari tindakan ini adalah untuk menyederhanakan data teks sehingga hanya elemen-elemen yang benar-benar relevan dengan tujuan analisis yang tersisa.

Dalam konteks penelitian atau analisis data, proses pembersihan data membawa manfaat yang signifikan. Dengan membersihkan dataset dari elemen-elemen yang tidak relevan, analis dapat fokus pada informasi yang benar-benar penting. Seperti contoh pada Tabel 2.2 yang menunjukkan hasil pembersihan data. Hal ini memungkinkan analisis yang lebih tepat dan akurat terhadap data teks yang tersedia.

**Tabel 2. 2** Contoh Data Cleansing

| No. | Tweet Asli  | Hasil Data Cleansing  |
|-----|---|---|
| 1   | Hari ini adalah 10 November, #HariPahlawan. Selamat hari pahlawan, @pahlawan45!                                       | Hari ini adalah 10 November, Selamat hari pahlawan!             |
| 2   | Gw lupa, tp kayanya besok bakal rilis nih 😊. Can't wait!! #excited  | Gw lupa, tp kayanya besok bakal rilis nih. Can't wait!! excited |
| 3   | URL untuk informasi lebih lanjut: <a href="https://www.contohlink.com/info123">https://www.contohlink.com/info123</a> | URL untuk informasi lebih lanjut:                               |

Selain itu, data cleansing juga membantu meningkatkan kualitas data secara keseluruhan. Data yang telah dibersihkan memiliki tingkat ketepatan dan kebersihan yang lebih tinggi, memungkinkan hasil analisis yang lebih andal. Dengan kata lain, proses ini menghilangkan gangguan yang dapat mempengaruhi hasil analisis, sehingga memastikan bahwa kesimpulan yang diambil dari data teks benar-benar relevan dan berguna.

Data cleansing memfasilitasi proses selanjutnya dalam penelitian atau analisis yang sedang dilakukan. Dengan dataset yang telah dibersihkan, penelitian dapat berjalan lebih lancar dan efisien, karena peneliti tidak perlu lagi menghadapi gangguan dari elemen-elemen yang tidak relevan. Sebagai hasilnya, waktu dan sumber daya dapat digunakan dengan lebih efektif untuk menghasilkan hasil yang lebih berharga dalam analisis data teks.

### **2.2.3 Normalization**

Normalisasi teks adalah suatu konsep yang merujuk pada proses yang bertujuan untuk mengubah format teks agar sesuai dengan tujuan tertentu. Dalam dunia teks dan bahan bacaan, seringkali kita menghadapi beragam permasalahan, seperti kata-kata yang ambigu, variasi dalam penggunaan bahasa, dan bahkan perubahan makna kalimat dari arti aslinya (Riyaddulloh dan Romadhony, 2021). Oleh karena itu, penggunaan normalisasi teks menjadi sangat penting dalam konteks ini. Mari kita perhatikan contoh pada Tabel 2.3 berikut, hasil normalisasi kata akan sangat membantu dalam memahami konteks makna.

**Tabel 2. 3** Contoh Normalization

| No. | Teks Asli             | Hasil Normalisasi       |
|-----|-----------------------|-------------------------|
| 1   | Sy suka memakan nasi. | Aku senang makan nasi.  |
| 2   | Gimana kabar lo?      | Bagaimana kabarmu?      |
| 3   | Tgl 15 Januari 2023   | Tanggal 15 Januari 2023 |

Proses normalisasi teks memerlukan strategi dan teknik yang disesuaikan dengan kasus-kasus tertentu. Proses dapat mencakup penggantian kata-kata yang ambigu dengan sinonim yang lebih jelas, penyesuaian ejaan agar sesuai dengan aturan, atau bahkan pemilihan bahasa baku dalam beberapa situasi. Hasil dari normalisasi teks adalah teks yang lebih mudah dipahami, sehingga dapat digunakan secara lebih efektif dalam berbagai konteks, termasuk dalam analisis teks (Luthfiarta dkk., 2013). Ini memungkinkan data yang telah dinormalisasi menjadi lebih relevan dan berguna dalam berbagai keperluan.

Dalam dunia pemrosesan bahasa alami dan analisis teks, normalisasi teks menjadi langkah awal yang sangat penting. Tujuannya adalah untuk memastikan bahwa data yang akan digunakan dalam analisis memiliki kualitas yang baik dan sesuai

dengan tujuan penelitian atau aplikasi yang diinginkan. Tanpa normalisasi, data teks mungkin mengandung noise atau gangguan yang dapat mengganggu hasil analisis dan pengambilan keputusan. Oleh karena itu, normalisasi teks menjadi fondasi yang kuat dalam pengolahan teks yang berkualitas tinggi.

#### **2.2.4 Tokenization**

Tokenisasi adalah langkah penting dalam pemrosesan teks di mana sebuah string input dibagi menjadi unit-unit kata berdasarkan aturan tertentu (Luthfiarta dkk., 2013). Proses ini secara rinci melibatkan identifikasi dan pemisahan setiap karakter dalam sebuah kalimat untuk menghasilkan unit-unit kata yang terpisah. Tujuan utama dari tokenisasi adalah untuk membongkar kalimat menjadi bagian-bagian komponen, yaitu kata-kata. Dengan cara ini, kita dapat mengeksplorasi setiap kata dalam suatu kalimat dengan lebih terstruktur, yang merupakan fondasi penting dalam analisis teks (Verma dkk., 2014).

Tokenisasi memainkan peran kunci dalam pemrosesan teks karena mengubah teks kontinu menjadi unit-unit terstruktur, yang memungkinkan komputer untuk lebih mudah memahami dan memproses informasi dalam teks (Prihatini, 2016). Dengan kata lain, tokenisasi menciptakan dasar yang kuat untuk melanjutkan langkah-langkah preprocessing selanjutnya dalam analisis teks dengan lebih mudah dan efisien. Ini membantu dalam mengurai teks menjadi elemen-elemen yang dapat dihitung, dianalisis, dan diterapkan dalam berbagai konteks. Pada proses ini, umumnya menggunakan fungsi *word\_tokenize()* yang disediakan oleh library NLTK. Contoh tahap tokenisasi dapat dilihat pada Tabel 2.4.

**Tabel 2. 4** Contoh Tokenization

| No. | Teks Asli   | Hasil Tokenization  |
|-----|---|---|
| 1   | Aku suka sekali makan nasi goreng setiap hari.              | “Aku”, “suka”, “sekali”, “makan”, “nasi”, “goreng”, “setiap”, “hari”                  |
| 2   | Ketika langit berubah menjadi mendung dan hujan mulai turun | “Ketika”, “langit”, “berubah”, “menjadi”, “mendung”, “dan”, “hujan”, “mulai”, “turun” |
| 3   | Setiap tanggal 10 November adalah hari pahlawan             | “Setiap”, “tanggal”, “10”, “November”, “adalah”, “hari”, “pahlawan”                   |

Penting untuk dicatat bahwa hasil dari tokenisasi tidak hanya berguna dalam pemahaman teks, tetapi juga dalam berbagai jenis analisis teks. Misalnya, hasil tokenisasi dapat digunakan dalam penghitungan kata, yang merupakan langkah penting dalam analisis teks untuk memahami distribusi kata-kata dalam dokumen. Selain itu, tokenisasi juga memungkinkan analisis sentimen dengan mengidentifikasi kata-kata kunci dalam teks yang dapat mengungkapkan perasaan atau pendapat, serta dapat digunakan dalam klasifikasi dokumen untuk mengelompokkan teks-teks ke dalam kategori yang sesuai. Dengan demikian, tokenisasi adalah langkah awal yang sangat penting dalam analisis teks yang memungkinkan pemrosesan teks yang efektif dan bermanfaat.

## 2.2.5 Stop Words Removal

Stop word removal atau yang sering disebut sebagai stop word removal, adalah langkah penting dalam pemrosesan teks yang bertujuan untuk meningkatkan efisiensi dan relevansi analisis teks. Stop words adalah kata-kata umum yang sering muncul dalam teks, seperti "dan," "atau," "di," "yang," dan sejenisnya. Meskipun stop words penting dalam bahasa untuk membentuk struktur tata bahasa, mereka sering tidak memberikan nilai informasi yang signifikan

dalam analisis teks. Oleh karena itu, menghapus stop words dari teks adalah langkah kunci dalam mempersiapkan data teks untuk analisis lebih lanjut. Penghapusan Stop-words merupakan proses yang dilakukan untuk menghapus kata-kata yang tidak memiliki arti. Tahap ini akan menggunakan library stopwords() Bahasa Indonesia yang disediakan oleh NLTK dan ditambah dengan kamus yang dibuat oleh Tala<sup>1</sup> sebanyak 758 kata. Contoh stop-words yang terdapat pada kamus Tala pada Tabel 2.5.

**Tabel 2. 5** Contoh Stop-word Removal

| No. | Teks Asli  | Teks Setelah <i>Stop-word Removal</i> |
|-----|--|---------------------------------------|
| 1   | Penggunaan stop words <b>dalam</b> analisis teks.  | Penggunaan stop words analisis teks.  |
| 2   | Analisis teks <b>merupakan</b> langkah penting.    | Analisis teks langkah penting.        |
| 3   | Kualitas data teks <b>sangat</b> menentukan hasil. | Kualitas data teks menentukan hasil.  |

Dalam contoh di atas, kita telah menghapus stop words seperti "dalam," "merupakan," "penting," "sangat," dan "yang" dari teks asli. Hasil penghapusan stop words adalah teks yang lebih fokus pada kata-kata kunci dan informasi yang lebih relevan dalam analisis teks. Penghapusan stop words dilakukan dengan mengidentifikasi dan menghilangkan kata-kata yang termasuk dalam daftar stop words dari teks. Hasilnya adalah teks yang lebih bersih, dengan fokus pada kata-kata kunci yang memiliki makna yang lebih penting dalam konteks analisis tertentu. Ini dapat meningkatkan akurasi

---

<sup>1</sup> <https://github.com/masdevid/ID-Stopwords>

analisis teks, mengurangi noise, dan membuat proses analisis lebih efisien.

Meskipun penghapusan stop words seringkali berguna dalam banyak kasus analisis teks, perlu diingat bahwa daftar stop words dapat bervariasi tergantung pada bahasa dan konteks analisis. Beberapa kata yang dianggap stop words dalam satu analisis mungkin memiliki nilai informasi yang tinggi dalam analisis lain. Oleh karena itu, pemilihan daftar stop words perlu disesuaikan dengan tujuan analisis tertentu. Selain itu, dalam beberapa konteks, penghapusan stop words mungkin tidak diperlukan, tergantung pada sifat data teks dan jenis analisis yang dilakukan.

Stop-word Removal adalah langkah awal yang penting dalam preprocessing teks, yang membantu menyederhanakan teks, meningkatkan relevansi, dan membuat analisis teks lebih efisien. Hal ini membantu peneliti dan praktisi dalam berbagai disiplin untuk memahami dan mengekstrak makna yang lebih mendalam dari data teks yang tersedia.

## 2.2.6 Stemming

*Stemming* adalah komponen integral dalam sistem Information Retrieval yang memiliki peran krusial dalam mengubah kata-kata dalam dokumen menjadi bentuk akarnya dengan menerapkan aturan-aturan tertentu (Tala, 2003). Salah satu *library* yang sering digunakan untuk melakukan stemming dalam bahasa Indonesia adalah Sastrawi, yang menggunakan algoritma Nazief dan Adriani. Penggunaan Sastrawi telah terbukti unggul dalam mengkonversi kata-kata ke bentuk dasarnya (*root*) (Soyusiawaty dkk., 2020). Kelebihan utama Sastrawi, yang menggunakan algoritma Nazief & Adriani, terletak pada pemilihan kamus kata dasar yang diadaptasi dari Kateglo dengan sedikit modifikasi.

Tujuan utama dari proses stemming adalah untuk menghilangkan variasi infleksi dan derivasi dari suatu kata sehingga menghasilkan bentuk dasar yang lebih umum (Jivani, 2011). Dalam tahap ini, kata-kata yang memiliki afiks akan diproses untuk menghapus afiksnya, sehingga hanya tersisa bentuk dasar. Proses stemming ini sangat penting dalam menyederhanakan analisis teks dengan mengurangi variasi kata yang mungkin muncul dalam teks yang sama. Pada tahap ini, *stemming* dilakukan dengan menggunakan *library Sastrawi*<sup>2</sup>. Contoh tahap *stemming* yang dapat dilihat pada Tabel 2.6.

**Tabel 2.6** Contoh Proses Stemming

| No. | Teks Asli   | Hasil <i>Stemming</i>  |
|-----|---|--|
| 1   | intinya kalian terlalu brengsek dan bangsat <b>menghakimi</b> sebuah <b>kepercayaan</b> | [“inti”, ‘kalian”, “terlalu”, “brengsek”, “dan”, “bangsat”, ,” <b>hakim</b> ”, “sebuah”, ” <b>percayaan</b> ”] |
| 2   | Ketika langit <b>berubah menjadi</b> mendung dan hujan mulai turun                      | [“Ketika”, “langit”, “ubah”, “jadi”, “mendung”, “dan”, “hujan”, “mulai”, “turun”]                              |
| 3   | Seorang Nelayan <b>menangkap</b> ikan <b>memakai</b> jaring                             | “Seorang”, “Nelayan”, “ <b>tangkap</b> ”, “ikan”, “ <b>pakai</b> ”, “jaring”                                   |

Dengan kata dasar yang dihasilkan melalui stemming, pemrosesan teks seperti pencarian kata kunci atau pengklasifikasian teks menjadi lebih efisien. Misalnya, ketika mencari kata kunci "berlari," stemming akan mengubahnya menjadi kata dasar "lari," sehingga hasil pencarian akan mencakup semua bentuk kata yang

---

<sup>2</sup> <https://pypi.org/project/Sastrawi/>

relevan. Selain itu, stemming juga memainkan peran penting dalam menghilangkan redundansi kata-kata, yaitu kata-kata yang memiliki akar yang sama, yang dapat mengganggu analisis teks yang lebih cermat. Dengan kata lain, stemming adalah alat penting dalam pemrosesan teks yang membantu menghasilkan representasi teks yang lebih terstruktur dan efisien dalam berbagai konteks analisis.

## **BAB III**

### **REPRESENTASI TEKS**

Representasi teks dalam pemrosesan bahasa alami (NLP) memungkinkan komputer untuk efisien memahami dan menganalisis teks. Dalam NLP, teks merupakan sumber data berharga yang harus diubah menjadi format numerik agar dapat dimengerti oleh algoritma komputer. Berbagai teknik representasi teks, seperti TF-IDF yang memberi bobot kata berdasarkan frekuensi dan keunikannya, serta metode seperti Word2Vec, GloVe, dan FastText yang mengonversi kata menjadi vektor numerik untuk menggambarkan hubungan semantik, digunakan dalam analisis teks. Di samping itu, BERT, model bahasa transformer yang bidireksional, telah menjadi terkenal karena menghasilkan representasi kata kontekstual dan informatif. Pemilihan metode representasi teks yang tepat sangat penting dalam analisis teks, dengan pertimbangan kelebihan dan kekurangannya yang harus sesuai dengan tujuan dan karakteristik data teks yang dihadapi. Representasi teks terus berkembang dan berperan sentral dalam membantu komputer memahami bahasa manusia dalam berbagai aplikasi NLP.

Dalam konteks analisis teks, pemilihan metode representasi teks yang tepat sangat penting karena dapat memengaruhi hasil analisis dan pemahaman teks. Setiap metode memiliki kelebihan dan kekurangan, dan pemilihan metode yang sesuai harus disesuaikan dengan tujuan dan karakteristik data teks yang dihadapi. Pada bab ini, kita akan berkenalan dengan apa itu representasi teks yang menjadi salah satu elemen kunci dalam NLP dan memainkan peran utama dalam membantu komputer memahami bahasa manusia.

### 3.1 Term Frequency and Inverse Document Frequency (TF-IDF)

TF-IDF (*Term Frequency-Inverse Document Frequency*) adalah salah satu metode representasi teks yang digunakan untuk mengukur pentingnya sebuah kata dalam suatu dokumen atau korpus teks. Metode ini digunakan secara luas dalam pemrosesan bahasa alami (NLP) untuk menganalisis dan mengekstrak informasi dari dokumen teks.

*Term Frequency (TF)*: Tahap pertama dari TF-IDF adalah mengukur sejauh mana kata tertentu muncul dalam dokumen. Ini dilakukan dengan menghitung frekuensi kata tersebut dalam dokumen. Frekuensi ini mewakili sejauh mana kata tersebut dominan dalam dokumen tersebut. Formula TF sederhana (Persamaan 3.1) adalah:

$$TF_{w,d} = \frac{n_{w,d}}{N_d} \quad (3.1)$$

Di mana:

$TF_{w,d}$  adalah Term Frequency dari kata w dalam dokumen d.

$n_{w,d}$  adalah jumlah kemunculan kata w dalam dokumen d.

$N_d$  adalah jumlah total kata dalam dokumen d.

Sebagai contoh, pertimbangkan dokumen berikut: "Analisis teks adalah bagian penting dari pemrosesan bahasa alami." Jika kita ingin menghitung TF kata "analisis" dalam dokumen ini, kita melihat bahwa kata tersebut muncul sekali dalam dokumen yang memiliki total 8 kata, sehingga TF kata "analisis" adalah 1/8.

*Inverse Document Frequency (IDF)*: Tahap kedua adalah mengukur sejauh mana kata tersebut unik dalam konteks seluruh korpus teks. Kata-kata umum yang muncul di banyak dokumen dapat memiliki TF yang tinggi, tetapi mungkin tidak informatif. Oleh karena itu, kita ingin memberikan bobot lebih pada kata-kata

yang unik. IDF mengukur seberapa sering kata w muncul dalam seluruh dokumen korpus. Berikut Persamaan 3.2 untuk IDF:

$$IDF_w = \log\left(\frac{N}{n_w}\right) \quad (3.2)$$

Di mana:

$IDF_w$  adalah Inverse Document Frequency dari kata w.

N adalah jumlah total dokumen dalam korpus.

$n_w$  adalah jumlah dokumen yang mengandung kata w.

Dalam contoh di atas, jika kita ingin menghitung IDF kata "analisis," kita perlu mengetahui berapa banyak dokumen dalam korpus yang mengandung kata "analisis."

*TF-IDF Score:* Setelah menghitung TF dan IDF, TF-IDF Score adalah hasil perkalian TF dengan IDF. Ini menghasilkan bobot yang mengukur seberapa pentingnya kata tersebut dalam dokumen tertentu dalam konteks korpus. Berikut Persamaan 3.3 untuk TF-IDF:

$$TF - IDF_{w,d} = TF_{w,d} * IDF_w \quad (3.3)$$

Jadi, jika kita ingin menghitung TF-IDF Score kata "analisis" dalam dokumen dengan TF sebelumnya (1/8) dan IDF yang sudah dihitung, kita akan mengalikan keduanya.

Hasil akhir dari penggunaan TF-IDF adalah vektor yang mewakili dokumen dengan bobot kata-kata yang mencerminkan pentingnya kata-kata tersebut dalam dokumen tersebut dan dalam konteks seluruh korpus teks. Bobot ini dapat digunakan dalam berbagai tugas analisis teks, seperti pengelompokan dokumen, ekstraksi informasi, dan analisis sentimen, untuk menilai relevansi kata-kata dalam dokumen terhadap topik atau tujuan tertentu.

### **3.2 Word2Vec**

Word2Vec adalah salah satu teknik embedding kata yang memetakan kata-kata dalam korpus teks ke dalam vektor numerik. Teknik ini dikembangkan oleh Tomas Mikolov dan timnya pada tahun 2013 dan telah menjadi salah satu metode paling populer untuk pemahaman kata-kata dalam pemrosesan bahasa alami (NLP) (Mikolov dkk., 2013). Word2Vec memungkinkan representasi vektor kata yang menggambarkan hubungan semantik antara kata-kata dalam korpus.

#### **Tahapan Proses Word2Vec**

1. Pemilihan Korpus: Tahap pertama dalam penggunaan Word2Vec adalah memilih korpus teks yang akan digunakan untuk pelatihan model. Korpus ini dapat berupa koleksi dokumen, artikel, atau bahkan seluruh internet, tergantung pada keperluan.
2. Tokenisasi dan Pembersihan Teks: Korpus teks perlu diuraikan menjadi token (unit kata) dan dihapus dari karakter khusus, tanda baca, dan token yang tidak relevan. Tahap ini membantu dalam memproses teks dengan benar sebelum digunakan dalam pembuatan model Word2Vec.
3. Pembuatan Model Word2Vec: Setelah teks dikumpulkan dan dibersihkan, langkah selanjutnya adalah pembuatan model Word2Vec. Ada dua jenis model Word2Vec: Continuous Bag of Words (CBOW) dan Skip-gram. CBOW mencoba memprediksi kata target berdasarkan kata-kata konteks di sekitarnya, sedangkan Skip-gram mencoba memprediksi kata-kata konteks berdasarkan kata target. Model ini memerlukan jumlah vektor yang direpresentasikan, yang juga disebut dimensi vektor. Dimensi vektor biasanya sekitar 100 hingga 300.

4. Pelatihan Model: Model Word2Vec diberi teks yang telah dibersihkan dan tokenisasi dari korpus teks. Selama pelatihan, model akan belajar mengasosiasikan kata-kata yang sering muncul bersama dalam konteks yang berbeda. Dengan demikian, kata-kata yang sering muncul bersamaan dalam berbagai konteks akan memiliki representasi vektor yang mirip.

### Contoh Penggunaan Word2Vec

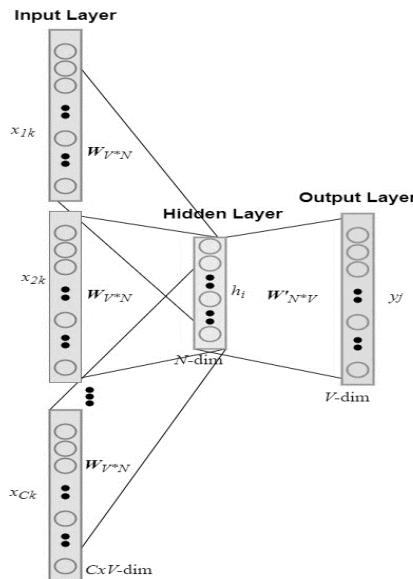
Misalkan kita memiliki kalimat: "Anjing adalah hewan peliharaan yang setia." Dalam pelatihan model Word2Vec, model akan memperhatikan kata "anjing" dan kata-kata lain yang muncul dalam konteks yang berbeda, seperti "hewan," "peliharaan," dan "setia." Hasilnya adalah representasi vektor kata-kata yang mencerminkan hubungan semantik. Dengan model Word2Vec, kita dapat melakukan operasi vektor, seperti menghitung "anjing" - "hewan" + "mobil" untuk menemukan kata yang mirip dengan "mobil" dalam konteks "hewan peliharaan yang setia."

Word2Vec telah menjadi alat yang sangat bermanfaat dalam NLP untuk berbagai tugas, termasuk pengelompokan dokumen, analisis sentimen, pencarian informasi, dan pemahaman bahasa manusia. Representasi vektor kata-kata yang dihasilkan oleh Word2Vec memungkinkan komputer untuk memahami dan menganalisis bahasa manusia dengan lebih baik.

#### 3.2.1 Continous Bag of Words (CBOW)

*Continuous Bag of Words* (CBOW) adalah salah satu model dalam Word2Vec yang digunakan untuk menghasilkan representasi vektor kata dalam pemrosesan bahasa alami (NLP). Model CBOW dirancang untuk memprediksi kata target (*target word*) berdasarkan kata-kata konteks (*context words*) yang mengelilinginya. CBOW

berusaha memahami hubungan antara kata-kata yang sering muncul bersama dalam berbagai konteks. Mari kita lihat arsitektur dari CBOW pada Gambar 3.1 berikut. Pada gambar tersebut memperlihatkan input layer berupa teks kemudian dilanjutkan dengan penambahan hidden layer dari setiap teks yang kemudian menghasilkan output akhir.



**Gambar 3. 1** Arsitektur Model *Continuous Bag of Words* (CBOW)

### Tahapan Proses *Continuous Bag of Words* (CBOW)

1. Pemilihan Korpus: Seperti pada Word2Vec, tahap awal adalah memilih korpus teks yang akan digunakan untuk pelatihan model CBOW. Korpus ini dapat berupa berbagai jenis dokumen yang relevan dengan tugas analisis yang diinginkan.
2. Tokenisasi dan Pembersihan Teks: Korpus teks perlu diuraikan menjadi token (unit kata) dan dibersihkan dari karakter khusus, tanda baca, dan token yang tidak relevan.

Tahap ini membantu dalam memproses teks dengan benar sebelum digunakan dalam pembuatan model CBOW.

3. Pembuatan Model CBOW: Selanjutnya, kita membuat model CBOW. Model ini berfokus pada memprediksi kata target (target word) berdasarkan kata-kata konteks (context words) yang mengelilinginya. Contoh dari kalimat "Anjing adalah hewan peliharaan yang setia" adalah sebagai berikut:

Kata target: "*hewan*"

Kata-kata konteks: ["*Anjing*", "*adalah*", "*peliharaan*", "*yang"]*

Model CBOW akan belajar mengasosiasikan kata target ("hewan") dengan kata-kata konteks ("Anjing", "adalah", "peliharaan", "yang") dalam berbagai konteks dan perbedaan posisi.

4. Pelatihan Model: Model CBOW diberi teks yang telah dibersihkan dan diuraikan dari korpus teks. Selama pelatihan, model akan belajar mengasosiasikan kata-kata konteks dengan kata target. Proses pelatihan ini melibatkan optimasi parameter model, seperti pembobotan vektor kata.

### **Contoh Continuous Bag of Words (CBOW)**

Misalkan model CBOW telah dilatih dan kita ingin menggunakan model ini untuk memahami hubungan antara kata-kata dalam kalimat "Anjing adalah hewan peliharaan yang setia." Jika kita memberikan kata-kata konteks ("Anjing", "adalah", "peliharaan", "yang") sebagai input ke model CBOW, model tersebut akan memberikan perkiraan kata target yang mungkin adalah "hewan." Dengan kata lain, model CBOW telah memahami bahwa kata-kata konteks ini sering muncul bersama dalam berbagai konteks dan memiliki hubungan semantik dengan kata target "hewan."

*Continuous Bag of Words* (CBOW) adalah salah satu pendekatan yang berguna untuk memahami dan merepresentasikan kata-kata dalam teks. Meskipun model ini tidak memperhitungkan urutan kata dalam kalimat (sehingga tidak dapat menangkap informasi urutan), itu berguna untuk tugas analisis teks yang lebih umum, seperti klasifikasi dokumen dan analisis sentimen.

### 3.2.2 Skip-Gram Model

Skip-Gram adalah model lain dalam Word2Vec yang berbeda dari *Continuous Bag of Words* (CBOW). Sementara CBOW berfokus pada memprediksi kata target berdasarkan kata-kata konteks di sekitarnya, Skip-Gram berusaha memprediksi kata-kata konteks berdasarkan kata target. Arsitektur model dapat dilihat pada Gambar 3.2. Skip-Gram lebih cocok untuk memahami hubungan antara kata-kata yang mungkin memiliki variasi konteks yang lebih luas. Berikut adalah tahapan proses Skip-Gram:

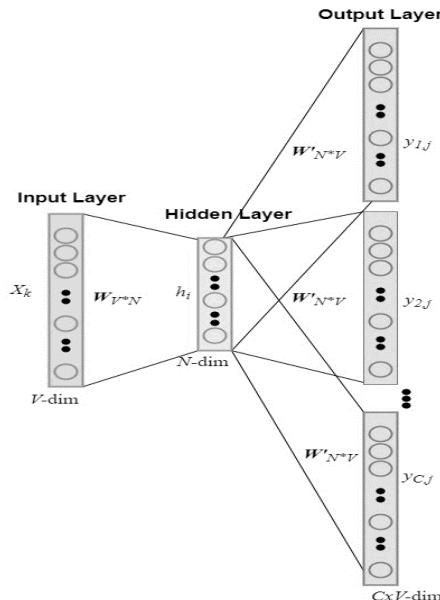
1. Pemilihan Korpus: Seperti dalam Word2Vec, tahap pertama adalah memilih korpus teks yang akan digunakan untuk pelatihan model Skip-Gram.
2. Tokenisasi dan Pembersihan Teks: Korpus teks perlu diuraikan menjadi token (unit kata) dan dibersihkan dari karakter khusus, tanda baca, dan token yang tidak relevan.
3. Pembuatan Model Skip-Gram: Model Skip-Gram berfokus pada memprediksi kata-kata konteks berdasarkan kata target. Contoh dari kalimat "Anjing adalah hewan peliharaan yang setia" adalah sebagai berikut:

Kata target: "hewan"

Kata-kata konteks yang mungkin: ["Anjing", "adalah", "peliharaan", "yang"]

Model Skip-Gram akan belajar mengasosiasikan kata-kata konteks ("Anjing", "adalah", "peliharaan", "yang") dengan

- kata target ("*hewan*") dalam berbagai konteks dan perbedaan posisi.
- Pelatihan Model: Model Skip-Gram diberi teks yang telah dibersihkan dan diuraikan dari korpus teks. Selama pelatihan, model akan belajar mengasosiasikan kata target dengan kata-kata konteks yang mungkin muncul bersamanya dalam berbagai konteks.



**Gambar 3. 2** Arsitektur Model Skip-Gram

### Contoh Skip-Gram

Misalkan model Skip-Gram telah dilatih dan kita ingin menggunakan model ini untuk memahami hubungan antara kata-kata dalam kalimat "*Anjing adalah hewan peliharaan yang setia.*" Jika kita memberikan kata target ("*hewan*") sebagai input ke model Skip-Gram, model tersebut akan memberikan kata-kata konteks yang mungkin, seperti "*Anjing*", "*adalah*", "*peliharaan*", "*yang*" dan sebagainya. Dengan kata lain, model Skip-Gram telah memahami

bahwa kata target "hewan" sering muncul bersamaan dengan kata-kata konteks ini dalam berbagai konteks dan variasi posisi.

Skip-Gram lebih berguna ketika kita ingin memahami hubungan antara kata-kata dalam konteks yang lebih luas dan ketika kita ingin memperoleh representasi vektor kata yang mewakili makna kata dengan cara yang lebih kontekstual. Model ini berguna untuk tugas seperti word analogy, pemahaman bahasa manusia, dan pemodelan semantik kata. Berikut perbandingan dari 2 model Word2Vec antara Skip-Gram dan CBOW pada Tabel 3.1.

**Tabel 3. 1 Perbandingan Model Skip-Gram dan CBOW**

| No. | Aspek Perbandingan    | <i>Continuous Bag of Words</i> (CBOW)                                  | Skip-Gram   |
|-----|-----------------------|--|---|
| 1   | Pendekatan            | Memprediksi kata target berdasarkan kata konteks.                      | Memprediksi kata konteks berdasarkan kata target.                           |
| 2   | Ukuran Data Pelatihan | Lebih kecil, karena menggunakan satu kata target.                      | Lebih besar, karena melibatkan banyak pasangan kata target-konteks.         |
| 3   | Ukuran Vektor Kata    | Lebih stabil, karena menggabungkan informasi dari banyak kata konteks. | Lebih fleksibel, karena mengatasi kata-kata konteks yang sangat bervariasi. |
| 4   | Aplikasi Umum         | Klasifikasi dokumen, analisis sentimen, pemrosesan bahasa alami umum.  | Pemodelan semantik kata, word analogy, pemahaman kontekstual hubungan kata. |

### **3.3 Global Vectors for Word Representation (GloVe)**

*Global Vectors for Word Representation* (GloVe) adalah metode embedding kata yang memodelkan hubungan antara kata-

kata dalam teks dengan matriks kemunculan kata (*co-occurrence matrix*) dan vektor kata yang merepresentasikan kata-kata tersebut dalam ruang vektor. GloVe memanfaatkan statistik kemunculan kata dalam teks untuk memahami makna dan hubungan antara kata-kata. GloVe mengasumsikan bahwa hubungan antara kata-kata dalam korpus teks dapat dijelaskan melalui perbedaan vektor antara kata target dan kata konteks. Persamaan matematika dasar GloVe dapat dilihat pada Persamaan 3.4 sebagai berikut:

$$F(w_i, w_j, \widetilde{w_k}) = \frac{P_{ik}}{P_{jk}} \quad (3.4)$$

Di mana:

$F(w_i, w_j, \widetilde{w_k})$  adalah fungsi yang mengukur hubungan antara kata target  $w_i$ , kata konteks  $w_j$ , dan kata-kata lain  $\widetilde{w_k}$ .

$P_{ik}$  adalah probabilitas kemunculan bersama kata target  $w_i$  dan kata konteks  $\widetilde{w_k}$ .

$P_{jk}$  adalah probabilitas kemunculan bersama kata konteks  $w_j$ , dan kata konteks  $\widetilde{w_k}$ .

Fungsi ini mencoba mengekspresikan hubungan antara kata target dan kata konteks dengan mengukur rasio probabilitas kemunculan bersama kata-kata tersebut.

## Tahapan Proses GloVe

1. Pemilihan Korpus: Seperti dalam teknik pemrosesan teks lainnya, tahap pertama adalah memilih korpus teks yang akan digunakan untuk pelatihan model GloVe. Korpus ini dapat berupa berbagai jenis dokumen, artikel, atau teks lainnya yang relevan dengan tugas analisis yang diinginkan.
2. Konstruksi Matriks Kemunculan Kata: Untuk memahami hubungan antara kata-kata dalam korpus, GloVe membangun matriks kemunculan kata, yang disebut juga

sebagai co-occurrence matrix. Matriks ini mengukur sejauh mana kata-kata muncul bersama dalam konteks yang berbeda. Setiap entri dalam matriks mencerminkan jumlah kemunculan bersama antara dua kata dalam suatu jendela konteks.

3. Perhitungan Matriks Kemunculan Kata: Matriks kemunculan kata dapat dihitung menggunakan berbagai metode, seperti pendekatan berbasis jendela konteks atau pendekatan berbasis dokumen. Sebagai contoh, mari kita lihat pada ilustrasi Tabel 3.2. Jika kita memiliki kalimat "Anjing adalah hewan peliharaan yang setia" dan kita menggunakan jendela konteks yang mencakup dua kata di sekitar kata target, maka matriks kemunculan kata akan mencatat hubungan antara kata-kata dalam jendela konteks.

**Tabel 3. 2** Contoh Matriks Kemunculan Kata

|            | Anjing | adalah | hewan | peliharaan | yang | setia |
|------------|--------|--------|-------|------------|------|-------|
| Anjing     | 0      | 1      | 1     | 1          | 1    | 1     |
| adalah     | 1      | 0      | 1     | 1          | 1    | 1     |
| hewan      | 1      | 1      | 0     | 1          | 1    | 1     |
| peliharaan | 1      | 1      | 1     | 0          | 1    | 1     |
| yang       | 1      | 1      | 1     | 1          | 0    | 1     |
| setia      | 1      | 1      | 1     | 1          | 1    | 0     |

4. Optimasi Model GloVe: Setelah matriks kemunculan kata dibuat, langkah selanjutnya adalah melakukan optimasi untuk menghasilkan vektor kata yang merepresentasikan kata-kata dalam ruang vektor. Ini melibatkan teknik-teknik optimasi seperti gradient descent untuk mengubah vektor kata sehingga mereka mencerminkan hubungan yang sesuai dengan matriks kemunculan kata.
5. Vektor Kata Akhir: Hasil akhir dari pelatihan GloVe adalah vektor kata yang merepresentasikan kata-kata dalam ruang vektor berdasarkan hubungan kemunculan kata dalam

korpus teks. Vektor kata ini dapat digunakan dalam berbagai tugas analisis teks, seperti pengelompokan dokumen, analisis sentimen, pencarian informasi, dan pemahaman semantik kata.

Mari lihat contoh sederhana implementasi GloVe untuk memahami hubungan antara kata-kata dalam kalimat "Anjing adalah hewan peliharaan yang setia."

1. Membangun Matriks Kemunculan Kata (*Co-occurrence Matrix*): Untuk membangun matriks kemunculan kata, kita menggunakan jendela konteks dengan panjang tertentu. Misalnya, kita menggunakan jendela konteks dengan panjang 2. Matriks ini akan mencatat hubungan kemunculan bersama kata-kata dalam jendela konteks. Hasilnya bisa seperti yang ditunjukkan dalam tahapan sebelumnya.
2. Perhitungan Vektor Kata (*Word Vectors*): Setelah matriks kemunculan kata dibuat, kita ingin menghitung vektor kata yang merepresentasikan kata-kata tersebut. Ini melibatkan proses optimasi yang menggunakan persamaan GloVe untuk mengubah vektor kata sehingga cocok dengan hubungan kemunculan kata dalam matriks.
3. Menggunakan Vektor Kata: Setelah vektor kata dihitung, kita dapat menggunakananya dalam berbagai tugas NLP. Misalnya, kita ingin memahami hubungan antara kata "anjing" dan "setia" dalam konteks kata "hewan." Kita dapat menghitung hubungan antara vektor kata "anjing" dan "setia" dengan menggunakan perbedaan vektor mereka. Jika vektor "anjing" - vektor "hewan" + vektor "setia" menghasilkan vektor yang mendekati vektor "hewan," ini menunjukkan hubungan semantik antara kata-kata tersebut.

GloVe adalah metode yang kuat untuk embedding kata yang telah digunakan dalam banyak aplikasi NLP. Representasi vektor kata yang dihasilkan oleh GloVe membantu komputer memahami makna dan hubungan antara kata-kata dalam konteks teks, dan ini sangat bermanfaat dalam tugas-tugas seperti pengelompokan dokumen, analisis sentimen, dan pemodelan semantik kata. GloVe telah menjadi salah satu teknik embedding kata yang populer dalam pemrosesan bahasa alami karena kemampuannya untuk menangkap hubungan semantik dan sintaktik antara kata-kata dalam konteks. Dengan menggunakan matriks kemunculan kata, GloVe membantu menghasilkan representasi vektor kata yang informatif dan bermanfaat untuk berbagai tugas NLP.

### **3.4 FastText**

FastText adalah metode pemrosesan bahasa alami (NLP) yang dikembangkan oleh Facebook AI Research yang digunakan untuk melakukan embedding kata dan teks. Salah satu ciri khas utama dari fastText adalah kemampuannya dalam memahami kata-kata yang jarang muncul atau bahkan kata-kata yang tidak ada dalam korpus teks pelatihan. Ini membuatnya menjadi salah satu metode yang sangat berguna dalam berbagai tugas NLP, termasuk klasifikasi teks, klasifikasi teks berbahasa asing, dan bahkan pemrosesan bahasa dengan dataset yang lebih kecil.

#### **Tahapan Proses FastText**

1. Pemilihan Korpus: Seperti dalam sebagian besar metode pemrosesan teks, langkah pertama adalah memilih korpus teks yang akan digunakan untuk melatih model fastText. Korpus ini dapat berupa berbagai jenis dokumen atau teks yang relevan dengan tugas analisis yang diinginkan.

2. Tokenisasi dan Pembentukan N-gram: fastText menggunakan tokenisasi untuk membagi teks menjadi unit kata atau kata-kata parsial, yang disebut N-gram. N-gram adalah sekuens kata yang terdiri dari N token, dan ini membantu fastText memahami hubungan makna antara kata-kata yang berdekatan dalam teks. Sebagai contoh, jika kita menggunakan N-gram dengan N=2, kalimat "Anjing adalah hewan peliharaan" akan menjadi: ["Anjing", "adalah", "hewan", "peliharaan", "Anjing adalah", "adalah hewan", "hewan peliharaan"].
3. Pelatihan Model fastText: fastText mempelajari representasi kata dan teks dalam korpus dengan cara yang agak berbeda dari Word2Vec. Model fastText memprediksi kata-kata dalam suatu konteks berdasarkan kata-kata yang ada dalam teks. Misalnya, jika kita memiliki kalimat "Anjing adalah hewan peliharaan yang setia," dan kita ingin memprediksi kata "setia," fastText akan mempertimbangkan kata-kata sekitarnya, seperti "hewan," "peliharaan," "adalah," dan "Anjing."
4. Vektor Kata Akhir: Hasil dari pelatihan fastText adalah representasi vektor kata yang mewakili setiap kata dalam korpus. Dengan vektor ini, kita dapat mengukur kemiripan antara kata-kata, melakukan analisis sentimen, atau bahkan melakukan klasifikasi teks dengan lebih baik.

Mari kita lihat contoh sederhana implementasi fastText untuk menghasilkan vektor kata:

1. Pemilihan Korpus: Kita menggunakan korpus yang berisi berbagai ulasan produk online.
2. Tokenisasi dan Pembentukan N-gram: Kita melakukan tokenisasi pada ulasan-ulasan tersebut dan membentuk N-gram dengan N=2.

3. Pelatihan Model fastText: Model fastText diberi teks dari ulasan produk sebagai input. Misalkan kita ingin memprediksi kata "bagus" dalam konteks ulasan produk. Model fastText akan mempertimbangkan kata-kata sekitarnya dalam kalimat seperti "sangat," "kualitas," "sangat bagus," dan "produk."
4. Vektor Kata Akhir: Hasil dari pelatihan fastText adalah vektor kata yang merepresentasikan setiap kata dalam korpus. Misalnya, vektor "bagus" akan merepresentasikan makna kata "bagus" dalam konteks ulasan produk tersebut.

FastText adalah salah satu metode yang populer dalam pemrosesan bahasa alami karena kemampuannya dalam memahami kata-kata yang jarang muncul atau bahkan tidak ada dalam korpus pelatihan. Representasi vektor kata yang dihasilkan fastText sangat berguna dalam berbagai tugas analisis teks, terutama ketika kita memiliki dataset yang relatif kecil atau dataset berbahasa asing.

### **3.5 *Embeddings from Language Model (ELMo)***

*Embeddings from Language Model (ELMo)* adalah metode embedding kata kontekstual yang memanfaatkan model bahasa berbasis jaringan saraf untuk menghasilkan vektor kata yang memperhitungkan konteks di sekitar kata tersebut. ELMo dikenal karena kemampuannya mengatasi permasalahan ambiguitas kata dan variasi makna dalam bahasa.

#### **Tahapan Proses ELMo**

1. Pemilihan Korpus dan Pelatihan Model Bahasa Dasar (*Base Model*): Langkah pertama dalam penggunaan ELMo adalah melatih model bahasa dasar (*base model*) menggunakan korpus teks yang besar. Model bahasa ini adalah jaringan

- saraf yang mampu mempelajari representasi kata dalam berbagai konteks.
2. Pemberian Bobot (*Weighted Sum*): Setiap kata dalam kalimat diberikan bobot yang mempengaruhi kontribusinya terhadap pemahaman konteks kalimat. Ini dilakukan dengan menghitung jumlah tertimbang (*weighted sum*) dari representasi kata yang dihasilkan oleh model bahasa dasar. Representasi kata adalah vektor numerik yang mencerminkan makna kata dalam berbagai konteks. Bobot ini memungkinkan ELMo untuk membedakan kontribusi kata-kata dalam kalimat.
  3. Penggabungan Representasi Kata: Representasi kata yang memiliki bobot digabungkan menjadi representasi kata kontekstual. Representasi ini mencerminkan makna kata dalam konteks spesifik kalimat. Penggabungan ini dapat dilakukan dengan Persamaan 3.5 sebagai berikut:
- $$ELMo_i = ELMo_i^{task} = \gamma \sum_j s_{ij} h_j \quad (3.5)$$
- Dimana :
- $ELMo_i$  adalah Representasi kata kontekstual dari kata ke-i dalam kalimat.
- $ELMo_i^{task}$  adalah Representasi kata yang digunakan dalam tugas tertentu (misalnya, analisis sentimen).
- $\gamma$  adalah Skalar yang mengkalibrasi vektor hasil penggabungan.
- $s_{ij}$  adalah Bobot untuk kata ke-i yang mempengaruhi kontribusinya terhadap representasi kata kontekstual.
- $h_j$  adalah Representasi kata dasar (base word representation) dari kata ke-j dalam kalimat.
4. Penggunaan Vektor Kata Kontekstual: Representasi kata kontekstual yang dihasilkan oleh ELMo dapat digunakan dalam berbagai tugas analisis teks, seperti klasifikasi teks

atau pencarian informasi. Dalam tugas analisis sentimen, misalnya, representasi kata kontekstual membantu dalam memahami makna kata dalam konteks kalimat dan menghasilkan hasil yang lebih akurat.

### Contoh Implementasi ELMo

Untuk memberikan contoh perhitungan dalam tugas klasifikasi sentimen menggunakan ELMo, mari asumsikan kita memiliki model ELMo yang telah dilatih sebelumnya dan kita ingin mengklasifikasikan tiga ulasan produk berikut sebagai positif, negatif, atau netral berdasarkan representasi kata kontekstual yang dihasilkan oleh ELMo:

**Tabel 3.3** Contoh Ulasan

| No. | Ulasan   |
|-----|--|
| 1   | Produk ini sangat bagus. Saya sangat puas dengan kualitasnya |
| 2   | Sangat kecewa dengan produk ini. Tidak sesuai ekspektasi     |
| 3   | Produk ini standar. Tidak terlalu bagus atau buruk           |

Berdasarkan Tabel 3.3 diatas, mari kita asumsikan bahwa model klasifikasi menggunakan representasi kata kontekstual dari ELMo sebagai fitur untuk mengklasifikasikan sentimen. Model ini dapat digunakan untuk memberikan nilai sentimen pada setiap ulasan, dan nilai ini kemudian dapat diinterpretasikan sebagai sentimen positif, negatif, atau netral. Misalkan model klasifikasi telah memberikan skor sentimen pada setiap ulasan dengan rentang skala -1 hingga 1, di mana -1 adalah sentimen negatif, 0 adalah sentimen netral, dan 1 adalah sentimen positif. Hasil perhitungan sentimen mungkin terlihat seperti ini:

- Ulasan 1:
  - Representasi kata kontekstual dari ELMo: [0.9, 0.8, 0.7, 0.6, 0.85]

- Skor Sentimen: 0.85 (Positif)
- Ulasan 2:
  - Representasi kata kontekstual dari ELMo: [-0.9, -0.8, -0.7, -0.6, -0.85]
  - Skor Sentimen: -0.85 (Negatif)
- Ulasan 3:
  - Representasi kata kontekstual dari ELMo: [-0.1, 0.1, 0.05, -0.05, 0.0]
  - Skor Sentimen: 0.0 (Netral)

Dalam contoh di atas, model klasifikasi menggunakan representasi kata kontekstual yang dihasilkan oleh ELMo untuk memberikan skor sentimen pada setiap ulasan. Ulasan pertama dianggap positif dengan skor 0.85, ulasan kedua dianggap negatif dengan skor -0.85, dan ulasan ketiga dianggap netral dengan skor 0.0. Skor sentimen ini dapat digunakan untuk mengklasifikasikan ulasan produk ke dalam kategori sentimen yang sesuai.

ELMo telah digunakan secara luas dalam berbagai tugas NLP dan telah membantu mengatasi permasalahan yang sulit dalam pemrosesan bahasa alami. Dengan memperhitungkan konteks kata, ELMo dapat menghasilkan vektor kata yang lebih kaya dan informatif, yang berguna dalam pemodelan bahasa manusia dan berbagai aplikasi NLP.

### ***3.6 Bidirectional Encoder Representations from Transformers (BERT)***

#### **3.6.1 Definisi**

BERT (*BiDirectional Encoder Representations from Transformers*) merupakan sebuah makalah penelitian yang diterbitkan oleh tim Google AI (Devlin dkk., 2019). Dibandingkan dengan arsitektur NLP sebelumnya, BERT menonjol dengan

kesederhanaan konsepnya namun memiliki kekuatan empiris yang luar biasa. BERT berhasil mencapai hasil yang luar biasa pada 11 tugas pemrosesan bahasa alami (Natural Language Processing - NLP).

Keunggulan utama BERT dibandingkan dengan model bahasa (*Language Models - LM*) standar lainnya terletak pada penerapan pelatihan dua arah (bidireksional) yang mendalam terhadap urutan teks. Ini berarti BERT mempertimbangkan konteks baik dari sisi kiri maupun kanan saat proses pelatihan. Sebaliknya, model LM lain, seperti OpenAI GPT, hanya dapat mempertimbangkan token sebelumnya dalam lapisan perhatiannya, yang disebut sebagai pendekatan searah. Pembatasan ini terbukti kurang optimal dalam tugas yang melibatkan pemahaman tingkat kalimat, seperti tugas parafase, atau dalam tugas tingkat token, seperti pengenalan entitas atau menjawab pertanyaan, di mana penggabungan konteks dari kedua arah menjadi sangat penting.

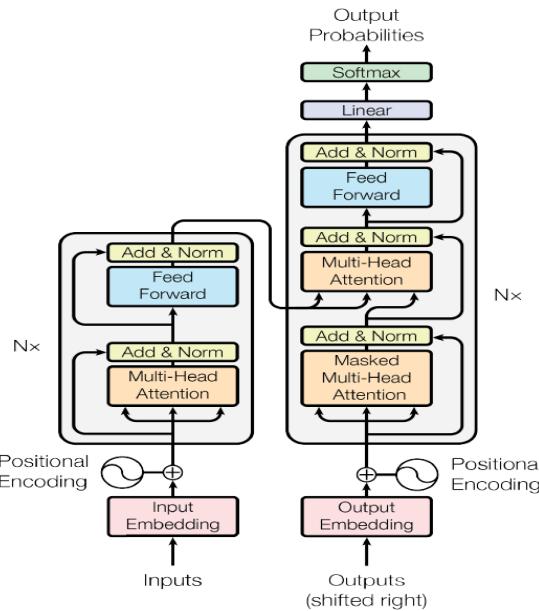
Jadi, apa itu BERT? BERT memperbaiki masalah *embedding* kata yang statis dalam model LM sebelumnya, seperti Glove, di mana kata-kata seperti "benar" memiliki *embedding*nya yang sama tanpa memperhitungkan konteks dalam kalimat. Misalnya, apakah "benar" berarti "benar" atau "arah yang benar"? ELMo mencoba mengatasi masalah ini dengan menggabungkan konteks kiri-ke-kanan dan kanan-ke-kiri untuk menghasilkan representasi kata. Namun, representasi tersebut tidak memanfaatkan konteks dari kedua arah secara bersamaan. BERT, dengan lapisan perhatiannya, berhasil mengungguli semua model sebelumnya.

Untuk memahami perbedaan arsitektur model pra-pelatihan, BERT menggunakan Transformator dua arah, sementara OpenAI GPT menggunakan Transformator yang hanya memproses dari kiri ke kanan. ELMo, di sisi lain, menggunakan rangkaian LSTM yang diperlakukan secara independen untuk menghasilkan fitur yang digunakan dalam tugas akhir. BERT adalah satu-satunya di antara

ketiganya yang mengkondisikan representasi bersamaan pada konteks dari kedua arah di semua lapisan.

Arsitektur dasar Transformer terdiri dari dua komponen utama: Encoder dan Decoder. Bagian Encoder digunakan untuk membaca urutan input dan memprosesnya. Dalam kasus BERT, hanya bagian Encoder yang relevan karena kita fokus pada tugas Model Bahasa (LM). BERT menggunakan arsitektur Encoder Transformer untuk menghasilkan perhatian dua arah yang independen terhadap urutan input. Ini memungkinkan BERT untuk membaca seluruh kalimat secara bersamaan, dan lapisan perhatiannya dapat memahami konteks kata dari seluruh kata di sekitar, baik kiri maupun kanan. Ini merupakan salah satu alasan mengapa BERT sangat efektif dalam pemahaman bahasa manusia oleh mesin.

*Pre-training* BERT dilakukan pada dataset yang tidak memiliki label, sehingga proses ini bersifat tidak diawasi. Model BERT mengikuti dua langkah kunci dalam pengembangannya. Pertama, model menggunakan sejumlah besar data tanpa label untuk mempelajari representasi bahasa dalam proses yang tidak memerlukan pengawasan, yang dikenal sebagai tahap pra-pelatihan. Setelah itu, model yang telah menjalani pra-pelatihan tersebut dapat disesuaikan dengan lebih baik melalui proses yang diawasi, menggunakan sejumlah kecil data yang telah diberi label untuk melakukan berbagai tugas pengawasan. Gambar 3.3 berikut menunjukkan proses *encoder* dan *decoder*.



**Gambar 3.3** Encoder (kiri) dan Decoder (kanan) (Vaswani dkk., 2017)

Gambar 3.3 menunjukkan proses *encoder* dan *decoder* yang dapat dijelaskan sebagai berikut.

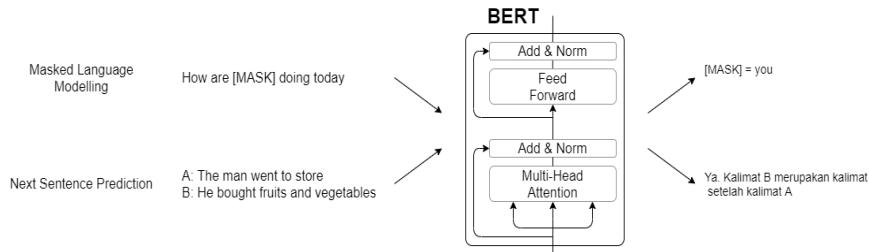
- Setiap kata input yang memasuki *encoder* diubah menjadi sebuah list vector menggunakan embeddings. Oleh karena lapisan *self-attention* tidak membedakan urutan kata-kata pada sebuah kalimat, positional encoding ditambahkan untuk menunjukkan posisi dari tiap kata. Tiap vektor dari input kata memiliki ukuran 512. Proses ini hanya terjadi *encoder* yang berada paling bawah, sehingga *encoder* lainnya akan menerima output dari *encoder* yang pertama.
- Vektor input melewati dua lapisan yang ada pada tiap *encoder* yaitu lapisan *self-attention* dan feed-forward neural network. Pada lapisan *self-attention* dibuat tiga vektor dari masing-masing input vektor yaitu *Query*, *Key*, dan *Value vector*. Ketiga vektor ini dibuat dengan mengalikan *embedding*. Dimensi dari tiap vektor adalah 64. Setelah itu,

nilai *self-attention* dari tiap kata dihitung dengan mengalikan *query vector* dan *key vector*. Selanjutnya, nilai *self-attention* dibagi 8 karena 8 adalah akar kuadrat dari dimensi tiap vektor yaitu 64. Nilai *self-attention* juga dihitung dengan *softmax* sehingga tiap *value vector* akan dikali dengan nilai dari *softmax*. Akhirnya *value vector* dijumlahkan dan menjadi output dari lapisan *self-attention*. Output dari lapisan *self-attention* kemudian masuk ke *feed-forward* untuk tiap posisi

- c. Setelah setiap proses pada *encoder* selesai, output dari *encoder* yaitu *vector key* dan *vector value* kemudian memasuki *decoder*. Tiap input dan output dari lapisan *self-attention* dan *feed-forward neural network* di *encoder* dan *decoder* diproses oleh lapisan *add & norm* yang berisi struktur residual dan lapisan normalisasi. Proses yang terjadi pada *decoder* sama dengan *encoder* akan tetapi di antara lapisan *self-attention* dan *feed-forward neural network* terdapat lapisan *attention* yang membantu *decoder* untuk fokus pada bagian-bagian dari kata yang relevan. Lapisan *self-attention* di *decoder* hanya boleh untuk menghadiri posisi sebelumnya dari output. Output dari tiap langkah dimasukkan ke dalam *decoder* terus menerus dan hasil dari *decoder* sama seperti hasil dari *encoder*. Akhirnya, output dari tumpukan *decoder* menghasilkan sebuah *vector* dengan nilai *float*. Untuk mengubahnya menjadi sebuah kata-kata, lapisan tambahan berupa *fully connected layer* dibutuhkan beserta lapisan *softmax*.

Keunggulan utama dalam performa model BERT terletak pada dua aspek utama. Pertama, BERT mengandalkan dua pendekatan pelatihan awal yang inovatif, yaitu *Masked Language Model* (MLM) dan *Next Sentence Prediction* (NSP). Kedua, model ini memanfaatkan sumber daya yang melimpah, termasuk volume data yang besar dan kemampuan komputasi tinggi, untuk

melatihnya. Selain itu, BERT menggunakan WordPiece embeddings yang memiliki kosa kata sekitar 30.000 token yang digunakan dalam proses pelatihan, seperti yang dapat dilihat pada Gambar 3.4 sebagai ilustrasi dari proses pra-pelatihan model BERT.



**Gambar 3. 4** Proses prapelatihan pada BERT

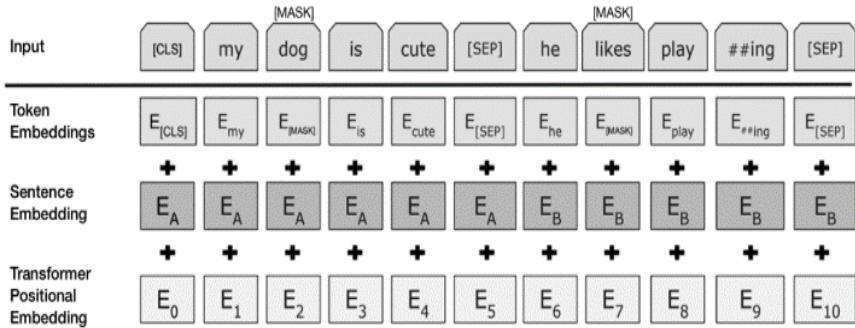
### 3.6.2 Representasi *Input* BERT

Model BERT memiliki batasan panjang maksimum untuk urutan kata-kata input, yang terbatas hingga 512 token, dan menggunakan metode tokenisasi dengan tokenizer WordPiece. Terdapat dua mode dalam model BERT, yaitu mode '*cased*' yang mempertahankan casing asli teks selama tokenisasi, dan mode '*uncased*' yang mengubah seluruh teks menjadi huruf kecil sebelum tokenisasi (Deepa, 2021).

Setiap representasi input dalam model BERT memiliki tiga lapisan *embedding*:

1. *Token embeddings*: Setiap token input diberikan representasi vektor dengan dimensi yang tinggi berdasarkan id token sesuai dengan kosakata BERT.
2. *Sentence embeddings*: Lapisan ini menunjukkan apakah token tersebut berasal dari kalimat pertama atau kedua jika terdapat lebih dari dua kalimat dalam input, untuk membedakan asal-usul token tersebut.

3. *Positional embedding*: Ini memberikan informasi tentang posisi kata dalam urutan. Konsep dan implementasi dari positional embedding telah dijelaskan dalam model Transformer. Model BERT mempelajari lapisan positional embedding selama proses pra-pelatihan.



**Gambar 3.5** Representasi Input pada BERT (Devlin dkk., 2018)

Pada Gambar 3.5, terlihat representasi input dari model BERT. Dalam rangka melatih model bahasa ini, beberapa penyesuaian perlu dilakukan pada model BERT agar dapat digunakan untuk *fine-tuning*, yang merupakan fase pelatihan khusus. Selama *fine-tuning*, *hyperparameter* dapat diatur sedemikian rupa untuk mencapai hasil yang optimal. Mekanisme self-attention memainkan peran penting dalam proses *fine-tuning* ini, karena fitur Transformer dalam model BERT memungkinkan model untuk menyelesaikan berbagai tugas dengan cara yang intuitif, yaitu dengan menganalisis dan memproses setiap kata dalam input dan menghasilkan output yang sesuai.

### 3.6.3 *Pre-training* BERT

Proses *pre-training* pada BERT melibatkan *encoder* dalam model Transformer yang memeriksa seluruh rangkaian kata dalam teks untuk memperoleh pemahaman konteks kata berdasarkan

semua kata yang terkandung dalam teks tersebut. Selama tahap *pre-training* ini, dua tugas prediksi tanpa supervisi dilakukan, yaitu *Masked Language Model* (MLM) dan Prediksi Kalimat Berikutnya *Next Sentence Prediction* (NSP) (Devlin dkk., 2019).

1. *Masked Language Model* (MLM), adalah salah satu tugas *pre-training* yang memungkinkan model BERT memahami konteks kata secara holistik. Dalam MLM, teks dikonversi menjadi token, dan representasi token digunakan sebagai input dan output selama pelatihan. Dalam tahap pelatihan ini, 15% dari token dalam input diubah menjadi token [MASK] yang harus diprediksi kembali oleh model. Prediksi kata yang benar dilakukan berdasarkan konteks yang diberikan oleh kata-kata sebelum dan sesudah token [MASK] dalam urutan kata. Meskipun MLM memungkinkan model *pre-training* dalam dua arah, ada kelemahan, yaitu token [MASK] tidak digunakan selama tahap *fine-tuning*, menciptakan ketidakcocokan antara kedua tahap ini.
2. *Next Sentence Prediction* (NSP), adalah tugas *pre-training* lainnya yang memungkinkan BERT memahami hubungan antara dua kalimat. Dalam tahap pelatihan NSP, 50% dari input terdiri dari pasangan kalimat di mana kalimat kedua adalah kalimat berikutnya dalam dokumen asli. Sementara 50% sisanya adalah kalimat kedua yang dipilih secara acak dari korpus. Model BERT dilatih untuk memprediksi apakah kalimat kedua mengikuti kalimat pertama atau tidak.

Dengan memasukkan MLM dan NSP dalam tahap *pre-training*, BERT dapat mengembangkan pemahaman konteks yang dalam dan luas dari teks, dan kemudian model tersebut dapat disesuaikan (*fine-tuned*) untuk tugas pemrosesan bahasa alami yang lebih spesifik. Proses ini telah menjadi dasar bagi banyak prestasi

terkini dalam NLP, memungkinkan model BERT untuk mencapai hasil canggih dalam berbagai tugas pemrosesan bahasa alami.

Arsitektur model BERT didasarkan pada konsep Transformer yang digunakan dalam pemrosesan bahasa alami. Ini adalah model yang berjalan pada dua arah, yang berarti ia mampu memahami konteks kata dalam teks dari kedua sisi (kiri ke kanan dan sebaliknya). Arsitektur BERT sangat berperan dalam kemajuan pemrosesan bahasa alami karena kemampuannya untuk mengatasi sejumlah besar tugas pemrosesan bahasa alami (*Natural Language Processing*, NLP).

Terdapat dua varian utama dari arsitektur BERT yang telah dikembangkan:

1. **BERT<sub>BASE</sub>:** Model BERT ini memiliki jumlah blok Transformer yang lebih sedikit dan ukuran lapisan tersembunyi yang lebih kecil dibandingkan dengan varian BERT yang lebih besar. BERT<sub>BASE</sub> memiliki 12 blok Transformer, 12 kepala perhatian, dan 768 ukuran lapisan tersembunyi. Meskipun ukurannya lebih kecil, BERT<sub>BASE</sub> tetap sangat kuat dan sering digunakan dalam berbagai tugas NLP.
2. **BERT<sub>LARGE</sub>:** Varian BERT yang lebih besar, dikenal sebagai BERT<sub>LARGE</sub>, memiliki 24 blok Transformer, 16 kepala perhatian, dan ukuran lapisan tersembunyi sebesar 1024. BERT<sub>LARGE</sub> dirancang untuk menangani tugas-tugas yang lebih kompleks dan biasanya memberikan hasil yang sangat canggih pada berbagai tugas NLP.

Ketika model BERT sedang dijalankan, selama proses penyempurnaan model, parameter lapisan seperti blok Transformer, kepala perhatian, dan lapisan tersembunyi dapat diatur dengan hati-hati agar sesuai dengan kebutuhan tugas tertentu. Ini memungkinkan penggunaan BERT dalam berbagai tugas pemrosesan bahasa alami

yang melibatkan pemahaman, klasifikasi, tanya-jawab, dan sebagainya.

Penting untuk dicatat bahwa arsitektur BERT adalah salah satu tonggak penting dalam perkembangan NLP, dan kemampuannya untuk memproses teks dalam dua arah dan diakses dengan beberapa varian ukuran telah menjadi landasan bagi banyak aplikasi pemrosesan bahasa alami yang telah kita lihat dan gunakan sehari-hari.

### 3.6.4 *Fine-tuning* BERT

*Fine-tuning* pada model BERT memungkinkan penyesuaian model *pre-trained* untuk tugas-tugas tertentu dengan mudah. Ini disebabkan oleh mekanisme self-attention yang memungkinkan BERT menghasilkan model yang dapat diterapkan pada berbagai jenis tugas pemrosesan bahasa alami. Proses *fine-tuning* berlangsung lebih cepat dibandingkan dengan pelatihan model dari awal, karena model BERT telah memahami berbagai fitur bahasa yang kompleks selama tahap *pre-training*. Dengan menggunakan fitur *pre-trained* yang dimiliki BERT, *fine-tuning* dapat dilakukan dengan data pelatihan yang lebih kecil dan memerlukan penyesuaian tugas yang minimal untuk mencapai kinerja yang baik. (Devlin dkk., 2019) juga memberikan rekomendasi *hyperparameter* yang memungkinkan mencapai kinerja optimal dalam tugas-tugas tertentu. Rekomendasi *Hyperparameter Fine-tuning*:

1. *Batch size*: *Batch size* adalah jumlah contoh pelatihan yang digunakan dalam satu iterasi. Rekomendasi yang diberikan adalah sekitar 16 atau 32. Pemilihan *batch size* yang tepat dapat memengaruhi kecepatan dan kualitas pelatihan.
2. *Learning rate* (Adam): *Learning rate* adalah parameter pelatihan yang mengatur seberapa besar perubahan bobot model dalam setiap iterasi. Devlin et al. merekomendasikan

nilai *learning rate* sekitar  $5e-5$  ( $5 \times 10^{-5}$ ),  $3e-5$  ( $3 \times 10^{-5}$ ), atau  $2e-5$  ( $2 \times 10^{-5}$ ). Pemilihan *learning rate* yang tepat mempengaruhi seberapa cepat dan seberapa baik model fine-tuned akan berkonvergensi.

3. *Epoch*: *Epoch* mengindikasikan satu siklus di mana model *machine Learning* melihat seluruh data pelatihan. Rekomendasi untuk jumlah *epoch* adalah sekitar 2, 3, atau 4. Jumlah *epoch* yang tepat dapat memastikan bahwa model mencapai tingkat kinerja yang optimal tanpa overfitting atau underfitting.

Dengan menggunakan rekomendasi *hyperparameter fine-tuning* ini, model BERT dapat diatur dengan baik untuk tugas-tugas pemrosesan bahasa alami yang berbeda dan menghasilkan hasil yang canggih dengan menggunakan data pelatihan yang lebih terbatas dibandingkan dengan pelatihan model dari awal.

### 3.6.5 *Pre-trained* BERT

Model *pre-trained* merujuk kepada model yang telah melewati tahap pelatihan awal menggunakan dataset berskala besar dan kemudian disesuaikan untuk mengeksekusi tugas-tugas pemrosesan bahasa alami pada dataset yang berbeda (Azhar dan Khodra, 2020). Dalam penelitian ini, kami akan memanfaatkan model *pre-trained* BERT yang memiliki spesifikasi tertentu dan melakukan *fine-tuning* dengan dataset yang kami punya untuk mengatasi permasalahan penelitian kami. Model BERT yang akan kami gunakan adalah IndoBERT dan IndoBERTweet (Koto dkk., 2020).

IndoBERT dan IndoBERTweet merupakan model *pre-trained* berskala besar yang berperan sebagai pelopor dalam kategori model BERT yang dikhususkan untuk bahasa Indonesia dengan fokus pada platform Twitter. Model ini merupakan perkembangan

dari model BERT berbahasa Indonesia yang telah menjalani pelatihan secara monolingual dan diperkaya dengan kosakata yang bersifat domain-specific.

Pembentukan model ini melibatkan penggunaan 26 juta data tweet berbahasa Indonesia yang terkumpul dari Desember 2019 hingga Desember 2020. Data tersebut mencakup 60 kata kunci yang berkaitan dengan empat topik utama, yaitu ekonomi, kesehatan, pendidikan, dan pemerintahan. Struktur encoder pada model ini terdiri dari 12 lapisan tersembunyi, dengan ukuran embedding tersembunyi setinggi 768, dan dilengkapi dengan 12 attention heads (Koto dkk., 2021).

## **BAB IV**

# **ALGORITMA KLASIFIKASI**

Penggunaan algoritma klasifikasi semakin penting dalam era digital yang terus berkembang pesat. Algoritma ini memainkan peran sentral dalam analisis dan pengelompokan teks, memungkinkan mesin untuk mengidentifikasi pola dan karakteristik khusus dalam dokumen teks.

Algoritma klasifikasi khususnya berbasis pada teks adalah bagian integral dari berbagai aplikasi, mulai dari pemrosesan bahasa alami hingga analisis sentimen, klasifikasi dokumen, dan pemahaman teks otomatis. Mereka memungkinkan komputer untuk mengelompokkan teks ke dalam kategori atau kelas yang sesuai, memungkinkan pengambilan keputusan otomatis dan pengelolaan informasi yang lebih efisien.

Dalam bab ini, kita akan mendalami algoritma klasifikasi teks, mencari tahu bagaimana mereka beroperasi, jenis data teks yang dapat diolah, serta peran kunci mereka dalam berbagai aplikasi praktis. Selain itu, kita akan mengeksplorasi perkembangan terkini dalam bidang ini, serta tantangan yang mungkin dihadapi dalam menerapkan algoritma klasifikasi teks dalam berbagai domain.

### **4.1 Machine Learning**

Dalam era perkembangan teknologi kecerdasan buatan atau *artificial intelligence* (AI) yang pesat saat ini, masih sedikit orang yang menyadari bahwa kecerdasan buatan memiliki beberapa cabang, di antaranya adalah *machine Learning* atau pembelajaran mesin. *Machine Learning* (ML), salah satu bagian penting dari AI, menarik perhatian karena kemampuannya belajar seperti manusia.

Dalam konteks kecerdasan buatan secara umum, terdapat tujuh cabang utama, yaitu *machine Learning*, *natural language processing*, *expert system*, *vision*, *speech*, *planning*, dan *robotics*. Pemisahan ini bertujuan untuk mempersempit fokus dalam pengembangan dan pembelajaran AI, karena kecerdasan buatan memiliki cakupan yang sangat luas. Dalam bab ini, kita akan lebih terfokus pada cabang kecerdasan buatan, yaitu *machine Learning* (ML), beserta algoritmanya. ML adalah teknologi yang mampu memproses data yang ada dan menjalankan tugas-tugas tertentu berdasarkan apa yang telah dipelajarinya. Sebelum kita memahami lebih lanjut tentang *machine Learning*, mari kita definisikan konsepnya terlebih dahulu.

*Machine Learning* telah memiliki peran yang signifikan dalam membantu manusia di berbagai bidang, bahkan saat ini, kita dapat dengan mudah menemui penerapannya dalam kehidupan sehari-hari. Misalnya, ketika kita menggunakan fitur face unlock untuk membuka smartphone kita atau ketika kita menjelajahi internet atau media sosial dan seringkali melihat iklan yang disesuaikan dengan preferensi pribadi kita. Iklan-iklan tersebut juga dibuat berdasarkan hasil pemrosesan *machine Learning*.

#### 4.1.1 Naive Bayes (NB)

Naive Bayes adalah sebuah teknik klasifikasi statistik yang berdasarkan pada Teorema Bayes. Algoritma ini merupakan salah satu metode dalam pembelajaran mesin yang paling sederhana dan populer. Klasifikasi Naive Bayes dikenal karena kecepatan dan keandalannya yang tinggi, serta mampu memberikan hasil yang akurat dalam berbagai kasus. Algoritma ini terutama efisien saat digunakan pada dataset yang besar, menjadikannya pilihan yang populer dalam berbagai aplikasi, seperti klasifikasi teks, klasifikasi email spam, dan pengenalan pola.

Salah satu asumsi utama yang menjadi dasar dari algoritma Naive Bayes adalah asumsi "naif" bahwa pengaruh dari suatu fitur terhadap kelas yang akan diprediksi adalah independen dari fitur lainnya. Dengan kata lain, meskipun fitur-fitur seperti pendapatan, riwayat pinjaman sebelumnya, usia, dan lokasi pemohon pinjaman sebenarnya dapat saling berkaitan dalam dunia nyata, Naive Bayes mengabaikan hubungan-hubungan ini dan menganggap setiap fitur sebagai entitas yang berdiri sendiri. Meskipun asumsi ini terkadang bisa tidak realistik dan memberikan keuntungan dalam hal perhitungan dan memungkinkan algoritma ini untuk beroperasi dengan cepat.

Asumsi independensi kondisional kelas, yang menjadi dasar dari Naive Bayes, berarti bahwa algoritma menganggap bahwa setiap fitur hanya bergantung pada kelas yang akan diprediksi dan tidak bergantung satu sama lain. Meskipun ini adalah asumsi yang sangat sederhana, dalam banyak kasus, algoritma Naive Bayes tetap memberikan hasil yang baik. Dalam praktiknya, Naive Bayes digunakan dalam berbagai aplikasi di mana kecepatan dan akurasi dalam klasifikasi menjadi sangat penting, meskipun dengan asumsi-asumsi yang "naif" tersebut. Proses perhitungan sederhana untuk algoritma naïve bayes dapat kita lihat pada Persamaan 4.1 berikut.

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)} \quad (4.1)$$

Dimana:

$P(h)$ : Merupakan probabilitas hipotesis  $h$  menjadi benar, tanpa mempertimbangkan data. Ini biasa disebut sebagai probabilitas sebelumnya dari  $h$ .

$P(D)$ : Adalah probabilitas data, tanpa mempertimbangkan hipotesis. Ini dikenal sebagai probabilitas sebelumnya.

$P(h | D)$ : Merupakan probabilitas hipotesis  $h$  diberikan data  $D$ . Hal ini dikenal sebagai probabilitas posterior, yang menggambarkan sejauh mana hipotesis  $h$  menjadi benar setelah melihat data  $D$ .

$P(D | h)$ : Adalah probabilitas data  $D$  dengan asumsi bahwa hipotesis  $h$  adalah benar. Ini juga dikenal sebagai probabilitas posterior, dan memberikan gambaran tentang seberapa sesuai data  $D$  dengan hipotesis  $h$  yang telah diajukan.

Sekarang, mari kita memahami cara kerja Naif Bayes melalui contoh. Diberikan contoh kondisi cuaca dan bermain olahraga. Anda perlu menghitung probabilitas bermain olahraga. Sekarang, Anda perlu mengklasifikasikan apakah pemain akan bermain atau tidak, berdasarkan kondisi cuaca. Klasifikasi Naive Bayes melibatkan perhitungan probabilitas suatu peristiwa dalam langkah-langkah berikut:

1. Langkah 1: Menghitung probabilitas sebelumnya (*prior probability*) untuk label kelas yang telah diberikan.
2. Langkah 2: Menemukan probabilitas peluang (*likelihood*) dengan menghubungkan setiap atribut dengan setiap kelas yang mungkin.
3. Langkah 3: Menggabungkan probabilitas prior dan *likelihood* dalam Formula Bayes dan menghitung probabilitas posterior.
4. Langkah 4: Membandingkan probabilitas posterior dari setiap kelas dan menentukan kelas mana yang memiliki probabilitas tertinggi, dengan mempertimbangkan input yang sesuai dengan probabilitas tersebut.

Untuk mempermudah perhitungan probabilitas sebelumnya dan probabilitas posterior, Anda dapat memanfaatkan dua jenis tabel, yaitu tabel frekuensi dan tabel probabilitas. Kedua tabel ini akan memudahkan Anda dalam menghitung probabilitas sebelumnya dan probabilitas sesudah. Tabel frekuensi berisi

informasi tentang seberapa sering label muncul dalam konteks berbagai fitur. Sementara itu, terdapat dua tabel probabilitas, di mana Tabel Probabilitas mencerminkan probabilitas sebelumnya (prior probability) dari label-label yang terlibat, dan Tabel Probabilitas 4.1 menggambarkan probabilitas posterior.

**Tabel 4. 1** Data Cuaca

| No. | Cuaca | Bermain Tenis |
|-----|-------|---------------|
| 1   | Hujan | Tidak         |
| 2   | Hujan | Tidak         |
| 3   | Hujan | Ya            |
| 4   | Cerah | Ya            |
| 5   | Cerah | Ya            |
| 6   | Hujan | Ya            |
| 7   | Cerah | Tidak         |
| 8   | Cerah | Ya            |

Sekarang, kita ingin memprediksi apakah seseorang akan bermain tenis (Ya) atau tidak (Tidak) berdasarkan cuaca yang saat ini cerah. Ini adalah langkah-langkah perhitungan:

### **Langkah 1: Hitung Probabilitas Sebelumnya (*Prior Probability*)**

- $P(\text{Bermain Tenis} = \text{Ya})$ : Jumlah kasus "Ya" dibagi dengan total kasus =  $4/8 = 0.5$
- $P(\text{Bermain Tenis} = \text{Tidak})$ : Jumlah kasus "Tidak" dibagi dengan total kasus =  $4/8 = 0.5$

### **Langkah 2: Hitung Probabilitas Peluang (*Likelihood*)**

- $P(\text{Cuaca} = \text{Cerah} | \text{Bermain Tenis} = \text{Ya})$ : Jumlah kasus dengan Cuaca = Cerah dan Bermain Tenis = Ya dibagi dengan total kasus "Ya" =  $2/4 = 0.5$
- $P(\text{Cuaca} = \text{Cerah} | \text{Bermain Tenis} = \text{Tidak})$ : Jumlah kasus dengan Cuaca = Cerah dan Bermain Tenis = Tidak dibagi dengan total kasus "Tidak" =  $2/4 = 0.5$

## Langkah 3: Hitung Probabilitas Posterior

- $P(\text{Bermain Tenis} = \text{Ya} | \text{Cuaca} = \text{Cerah}) = (P(\text{Bermain Tenis} = \text{Ya}) * P(\text{Cuaca} = \text{Cerah} | \text{Bermain Tenis} = \text{Ya})) = (0.5 * 0.5) = 0.25$
- $P(\text{Bermain Tenis} = \text{Tidak} | \text{Cuaca} = \text{Cerah}) = (P(\text{Bermain Tenis} = \text{Tidak}) * P(\text{Cuaca} = \text{Cerah} | \text{Bermain Tenis} = \text{Tidak})) = (0.5 * 0.5) = 0.25$

## Langkah 4: Prediksi Kelas

- Karena  $P(\text{Bermain Tenis} = \text{Ya} | \text{Cuaca} = \text{Cerah})$  (0.25) tidak lebih tinggi dari  $P(\text{Bermain Tenis} = \text{Tidak} | \text{Cuaca} = \text{Cerah})$  (0.25), kita memprediksi bahwa seseorang tidak akan bermain tenis saat cuaca cerah.

## Contoh Klasifikasi Naïve Bayes dengan Scikit-learn

Untuk menjalankan Script ini, Kita dapat menggunakan lingkungan kerja Google Colab<sup>3</sup> versi yang gratis. Walaupun versi gratis, akan tetapi Google Colab sudah sangat *powerfull* untuk menjalankan berbagai macam program *machine Learning* maupun deep learning.

### #Menentukan Dataset

Dalam contoh ini, kita dapat memanfaatkan dataset simulasi dengan tiga kolom: cuaca, suhu, dan bermain (*weather*, *temperature*, dan *play*). Dua kolom pertama adalah atribut (*weather*, *temperature*), sementara kolom terakhir adalah *play* sebagai label.

```
1 # Menetapkan fitur dan memberi label pada variabel
2 weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy',
3 'Overcast','Sunny','Sunny','Rainy','Sunny','Overcast',
4 'Overcast','Rainy']
5 temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool',
6 'Mild','Cool','Mild','Mild','Mild','Hot','Mild']
7 play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes',
8 'Yes','Yes','Yes','Yes','No']
```

---

<sup>3</sup> <https://colab.research.google.com/>

## #Pengkodean Fitur

Langkah pertama yang perlu kita lakukan adalah mengkonversi label dalam bentuk string menjadi nilai angka. Misalnya, kita dapat mengubah '*Overcast*', '*Rainy*', '*Sunny*' menjadi 0, 1, 2. Proses ini disebut sebagai pengkodean label. Untuk itu, kita memerlukan librari yang disediakan oleh Scikit-learn seperti pada baris ke-2 dari skrip dibawah ini. Dimana Scikit-learn menyediakan pustaka *LabelEncoder* yang dapat digunakan untuk melakukan pengkodean label dengan mengasosiasikan nilai antara 0 dan satu kurang dari jumlah kelas yang berbeda.

```
1 # Impor LabelEncoder
2 from sklearn import preprocessing
3 #membuat labelEncoder
4 le = preprocessing.LabelEncoder()
5 # Mengubah label string menjadi angka.
6 wheather_encoded=le.fit_transform(weather)
7 print (wheather_encoded)
output: [2 2 0 1 1 1 0 2 2 1 2 0 0 1]
```

Secara serupa, kita juga memiliki opsi untuk mengkodekan kolom *temp* dan *play*. Pada baris ke-2 kita membuat variabel *temp\_encoded* untuk menampung hasil transformasi dari kolom *temp* dan pada baris ke-3 membuat variabel *label* untuk menampung hasil transformasi dari kolom *play*.

```
1 # Mengubah label string menjadi angka
2 temp_encoded=le.fit_transform(temp)
3 label=le.fit_transform(play)
4 print ("Temp:",temp_encoded)
5 print ("Play:",label)
output:Temp: [1 1 1 2 0 0 0 2 0 2 2 2 1 2]
Play: [0 0 1 1 1 0 1 0 1 1 1 1 0]
```

Sekarang gabungkan kedua fitur (*weather* dan *temp*) pada baris ke-2 dalam satu variabel (*list tupel*) pada baris ke-3.

```
1 #Gabungkan cuaca dan suhu ke dalam satu daftar tupel
2 features = zip(wheather_encoded,temp_encoded)
3 result_list = list(features)
4 print (result_list)
output: [(2, 1), (2, 1), (0, 1), (1, 2), (1, 0), (1, 0), (0, 0), (2,
2), (2, 0), (1, 2), (2, 2), (0, 2), (0, 1), (1, 2)]
```

Terakhir, sekarang kita dapat melihat hasil prediksi output setelah menjalankan perintah pada baris ke-10. Disini kita menggunakan fungsi GaussianNB() yang ditampung pada variabel model di baris ke-4. Pada baris ke-8, variabel predicted akan menampung hasil dari predik prediksi input berupa “Cuaca” dan “Suhu”, kemudian output berupa keputusan “Bermain”. Misalnya pada Hasil berikut input nya berupa [0,2] yang artinya bahwa Cuaca = Cerah dan Suhu = Sejuk, maka output yang dihasilkan adalah [1] artinya Bermain = Ya.

```
1 #Impor model Gaussian Naive Bayes
2 from sklearn.naive_bayes import GaussianNB
3 #Buat Pengklasifikasi Gaussian
4 model = GaussianNB()
5 # Latih model menggunakan set pelatihan
6 model.fit(result_list,label)
7 #Prediksi Output
8 predicted= model.predict([[0,2]]) # Kolom Pertama [0:Cerah,
   1:Hujan, 2:Mendung], Kolom Kedua [1: Panas, 2:Sejuk, 0:Dingin]
9 # Output 1: Ya, 0: Tidak
10 print ("Predicted Bermain:", predicted)
output: Predicted Bermain: [1]
```

## **Contoh Naive Bayes dengan Multilabel**

Selanjutnya, kita akan mempelajari tentang klasifikasi multi-kelas dalam konteks Naive Bayes, yang dikenal sebagai klasifikasi Naive Bayes multinomial. Dalam pembuatan model, kita akan menggunakan dataset wine, yang merupakan permasalahan klasifikasi multi-kelas yang sangat populer. Dataset ini terdiri dari 13 fitur yang mencakup *alkohol*, *malic\_acid*, *abu*, *alcalinity\_of\_ash*, *magnesium*, *total\_phenols*, *flavanoids*, *nonflavanoid\_phenols*, *proanthocyanins*, *color\_intensity*, *hue*, *od280/od315\_of\_diluted\_wines*, dan *proline*, serta jenis kultivar anggur. Dataset ini memiliki tiga jenis anggur, yaitu *Class\_0*, *Class\_1*, dan *Class\_3*. Di sini, kita akan membangun model untuk mengklasifikasikan jenis anggur. Dataset ini tersedia dalam pustaka scikit-learn, jadi buatlah proyek baru sebagaimana sebelumnya.

## #Loading data

Pertama mari kita muat dataset wine yang diperlukan dari dataset scikit-learn. Dataset wine kita peroleh secara langsung dari *library* yang disediakan oleh scikit-learn. Untuk itu kita bisa secara langsung menggunakan dengan cara memanggil fungsi `wine()` di baris ke-4.

```
1 # Import scikit-learn dataset library
2 from sklearn import datasets
3 # Load dataset
4 wine = datasets.load_wine()
```

## #Exploring Data

Anda dapat mencetak nama target dan fitur, untuk memastikan Anda memiliki dataset yang tepat, seperti potongan skrip pada baris ke-2 dan baris ke-4. Sehingga output akan menampilkan seluruh fitur dan label dari kelas wine.

```
1 # print 13 Nama features
2 print ("Features: ", wine.feature_names)
3 # print the label type of wine(class 0, class 1, class 2)
4 print ("Labels: ", wine.target_names)
output:
    Features: ['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash',
'magnesium', 'total_phenols', 'flavanoids',
'nonflavanoid_phenols', 'proanthocyanins', 'color_intensity',
'hue', 'od280/od315 of diluted wines', 'proline']
    Labels: ['class_0' 'class_1' 'class_2']

5 # print data(feature) shape
6 wine.data.shape
output: (178, 13)
```

## #Memisahkan Data

Pertama, kita memisahkan kolom menjadi variabel dependen dan independen (atau fitur dan label). Kemudian kita membagi variabel-variabel tersebut ke dalam *train* dan *test*.

|                     | <b>Features</b> | <b>Labels</b> |
|---------------------|-----------------|---------------|
| <b>Training Set</b> | X_train         | y_train       |
| <b>Test Set</b>     | X_test          | y_test        |

```
1 # Import train_test_split function
2 from sklearn.model_selection import train_test_split
3 # Split dataset into training set and test set
4 # 70% training and 30% test
5 X_train, X_test, y_train, y_test = train_test_split(
6     wine.data, wine.target, test_size=0.3, random_state=109)
```

Seperti yang kita lihat pada potongan skrip diatas, kita memerlukan *library* `train_test_split` untuk dapat menjalankan fungsi pembagian data. Pada baris ke-5, variabel `x_train`, `x_test`, `y_train`, `y_test` akan digunakan sebagai pembagi data untuk *train set* dan *test set* dengan pembagian untuk data train sebesar 70% dan data test 30% dari dataset.

## #Pembuatan Model

Setelah pemisahan, Anda akan menghasilkan model naïve bayes pada *train set* dan melakukan prediksi menggunakan fitur *test set*.

```
1 #Import Gaussian Naive Bayes model
2 from sklearn.naive_bayes import GaussianNB
3 #Create a Gaussian Classifier
4 gnb = GaussianNB()
5 #Train the model using the training sets
6 gnb.fit(X_train, y_train)
7 #Predict the response for test dataset
8 y_pred = gnb.predict(X_test)
```

Pada baris ke-2, kita akan menggunakan fungsi `GaussianNB` yang ada pada *library* `sklearn.naive_bayes`.

## #Mengevaluasi Model

Setelah pembuatan model, periksa akurasi menggunakan nilai aktual dan prediksi. Hasil akhir dapat dilihat pada baris ke-4 bahwa model mampu memprediksi menggunakan *Test Set* dengan nilai akurasi sebesar 0.9074074074074 atau 90,7%.

```
1 #Import scikit-learn metrics module for accuracy calculation
2 from sklearn import metrics
3 # Model Accuracy, how often is the classifier correct?
4 print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
output: Accuracy: 0.9074074074074
```

#### **4.1.2 Random Forest Decision Tree (RFDT)**

Random Forest adalah teknik pembelajaran mesin yang digunakan untuk memecahkan masalah regresi dan klasifikasi. Algoritma ini merupakan algoritma pembelajaran mesin terawasi yang dibangun dari algoritma pohon keputusan. Ini menggunakan teknik ensemble learning, yang menggabungkan banyak klasifikasi untuk memberikan solusi untuk masalah yang kompleks. Algoritma Random Forest terdiri dari banyak pohon keputusan. 'Hutan' yang dihasilkan oleh algoritma Random Forest dilatih melalui bagging atau bootstrap aggregating. Bagging adalah meta-algoritma ensemble yang meningkatkan akurasi algoritma pembelajaran mesin.

Algoritma Random Forest (RF) menentukan hasil berdasarkan prediksi dari pohon keputusan. RF memprediksi dengan mengambil rata-rata atau mean dari hasil yang dihasilkan dari berbagai pohon. Dengan meningkatnya jumlah pohon, tingkat presisi hasil pun meningkat. Random Forest menghilangkan batasan algoritma pohon keputusan. RF mengurangi overfitting pada dataset dan meningkatkan presisi. RF menghasilkan prediksi tanpa memerlukan banyak konfigurasi dalam paket (seperti scikit-learn).

#### **Fitur-fitur Algoritma Random Forest**

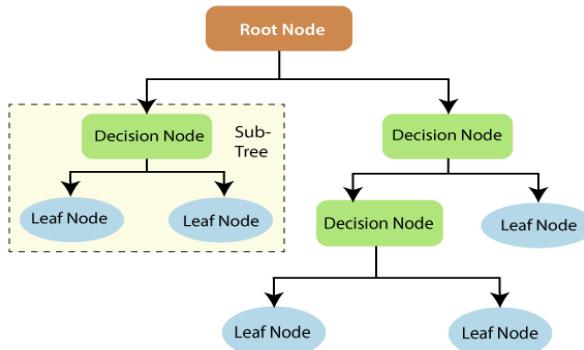
- Lebih akurat daripada algoritma pohon keputusan.
- Memberikan cara efektif untuk menangani data yang hilang.
- Dapat menghasilkan prediksi yang masuk akal tanpa penyetelan hiperparameter.
- Menyelesaikan masalah overfitting pada pohon keputusan.
- Pada setiap pohon Random Forest, subset fitur dipilih secara acak di titik pemisahan node.

## Memahami Decision Tree (Pohon Keputusan)

Pohon keputusan adalah bahan dasar dari algoritma Random Forest. Pohon keputusan adalah teknik pendukung keputusan yang membentuk struktur mirip pohon. Pemahaman tentang pohon keputusan akan membantu kita memahami cara kerja algoritma Random Forest.

Pohon keputusan terdiri dari tiga komponen: node keputusan, node daun, dan node akar. Algoritma pohon keputusan membagi dataset pelatihan menjadi cabang-cabang, yang selanjutnya terbagi menjadi cabang-cabang lain. Urutan ini berlanjut hingga mencapai node daun. Node daun tidak dapat dibagi lebih lanjut.

Node-node dalam pohon keputusan mewakili atribut yang digunakan untuk memprediksi hasil. Node keputusan menghubungkan ke node daun. Diagram berikut menunjukkan tiga jenis node dalam pohon keputusan. Mari kita lihat pada Gambar 4.1 berikut. Terdapat *Root Node* yang berperan sebagai batang utama yang dilanjutkan dengan cabang-cabang lainnya yaitu *Decision Node* yang disebut sebagai *Sub-Tree*. Pada setiap *Decision Node* terdapat *Leaf Node* yang berperan sebagai hasil akhir.



**Gambar 4. 1** Struktur Decision Tree

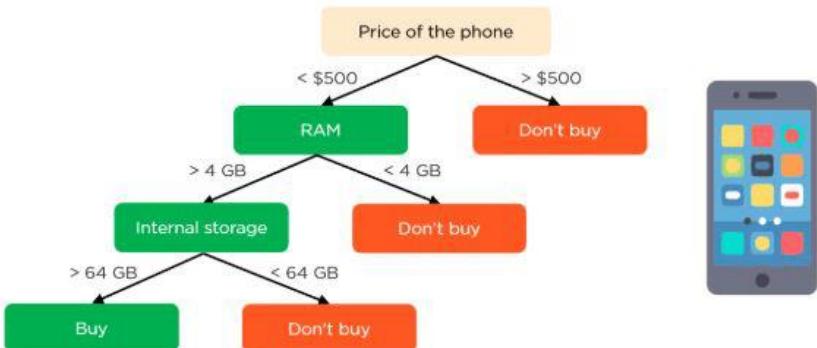
Teori informasi dapat memberikan pemahaman lebih lanjut tentang bagaimana pohon keputusan bekerja. Entropi dan gain

informasi adalah komponen dasar dari pohon keputusan. Gambaran tentang konsep-konsep dasar ini akan meningkatkan pemahaman kita tentang bagaimana pohon keputusan dibangun. Entropi adalah metrik untuk menghitung ketidakpastian. Gain informasi adalah ukuran sejauh mana ketidakpastian dalam variabel target berkurang, dengan diberikan kumpulan variabel independen.

Konsep gain informasi melibatkan penggunaan variabel independen (fitur) untuk mendapatkan informasi tentang variabel target (kelas). Entropi dari variabel target (Y) dan entropi bersyarat dari Y (diberikan X) digunakan untuk mengestimasi gain informasi. Dalam hal ini, entropi bersyarat dikurangkan dari entropi Y.

Gain informasi digunakan dalam pelatihan pohon keputusan. Ini membantu mengurangi ketidakpastian dalam pohon ini. Gain informasi yang tinggi berarti tingkat ketidakpastian yang tinggi (entropi informasi) telah dihilangkan. Entropi dan gain informasi penting dalam membagi cabang, yang merupakan kegiatan penting dalam pembangunan pohon keputusan.

Mari kita lihat contoh sederhana tentang bagaimana pohon keputusan bekerja. Misalkan kita ingin memprediksi apakah seorang pelanggan akan membeli ponsel atau tidak. Fitur-fitur ponsel membentuk dasar keputusannya. Analisis ini dapat disajikan dalam diagram pohon keputusan. Untuk ilustrasi lebih jelasnya dari Gambar 4.1 sebelumnya, mari kita lihat contoh penerapannya dalam kasus pemilihan ponsel pada Gambar 4.2 berikut.



**Gambar 4. 2** Decision Tree pada Pemilihan Ponsel

Dari Gambar 4.2 tersebut, kita dapat melihat bahwa harga sebuah ponsel akan berpengaruh terhadap beberapa aspek seperti RAM, Internal Storage, dan lain-lain. Untuk itu, terdapat dua aspek kunci yang membuat algoritma ini disebut "random":

1. Pengambilan Sampel Acak (*Random Sampling*): Saat membangun Random Forest, langkah awalnya melibatkan pengambilan sampel acak atau bootstrap dari data pelatihan. Dengan kata lain, setiap pohon keputusan tumbuh dari data yang dipilih secara acak dari dataset pelatihan dengan penggantian.
2. Pemilihan Fitur Acak (*Random Feature Selection*): Selama proses pembentukan setiap pohon keputusan, dalam setiap simpul pemisahan, sejumlah sampel acak dari variabel ( $m$  variabel) diambil dari data asli. Dari sampel tersebut, fitur terbaik dipilih untuk digunakan dalam simpul tersebut. Hal ini menghasilkan variasi antara pohon-pohon dalam hutan.

Algoritma ini sebenarnya adalah kombinasi dari beberapa prediksi pohon atau yang sering disebut sebagai decision trees. Setiap pohon dalam Random Forest bergantung pada vektor acak yang diambil secara acak dan merata dari semua pohon dalam

ensemble tersebut. Hasil prediksi akhir diperoleh dengan menggabungkan hasil dari setiap pohon keputusan. Dalam tugas klasifikasi, ini dilakukan melalui pemungutan suara mayoritas (voting), sementara dalam tugas regresi, hasilnya adalah rata-rata prediksi dari semua pohon.

## Penerapan Pohon Keputusan dalam Random Forest

Perbedaan utama antara algoritma pohon keputusan dan algoritma random forest adalah bahwa pembentukan node akar dan pemisahan node dilakukan secara acak dalam yang terakhir. Random forest menggunakan metode bagging untuk menghasilkan prediksi yang dibutuhkan.

Bagging melibatkan penggunaan berbagai sampel data (data pelatihan) daripada hanya satu sampel. Dataset pelatihan terdiri dari observasi dan fitur yang digunakan untuk membuat prediksi. Pohon keputusan menghasilkan output yang berbeda, tergantung pada data pelatihan yang disampaikan ke algoritma random forest. Output ini akan diurutkan, dan yang tertinggi akan dipilih sebagai output akhir.

Contoh pertama kita masih bisa digunakan untuk menjelaskan bagaimana random forest bekerja. Alih-alih memiliki satu pohon keputusan, random forest akan memiliki banyak pohon keputusan. Misalkan kita hanya memiliki empat pohon keputusan. Dalam hal ini, data pelatihan yang terdiri dari observasi dan fitur ponsel akan dibagi menjadi empat node akar.

Node akar bisa mewakili empat fitur yang dapat memengaruhi pilihan pelanggan (harga, penyimpanan internal, kamera, dan RAM). Random forest akan membagi node dengan memilih fitur secara acak. Prediksi akhir akan dipilih berdasarkan hasil dari keempat pohon.

Hasil yang dipilih oleh sebagian besar pohon keputusan akan menjadi pilihan akhir. Jika tiga pohon memprediksi pembelian, dan satu pohon memprediksi tidak membeli, maka prediksi akhir akan

menjadi pembelian. Dalam hal ini, diprediksi bahwa pelanggan akan membeli ponsel. Mari kita pahami menggunakan Persamaan 4.2 berikut.

$$\hat{Y} = \frac{1}{N} \sum_{n=1}^N I(\hat{h}_n(x)) \quad (4.2)$$

Dimana:

$\hat{Y}$  merupakan hasil prediksi akhir.

N merupakan jumlah pohon dalam Random Forest.

I adalah fungsi indikator yang menentukan apakah hasil prediksi pohon ke-n  $\hat{h}_n(x)$  benar atau salah.

Salah satu fitur menarik dari Random Forest adalah kemampuannya untuk memberikan estimasi kesalahan generalisasi sendiri, yang dikenal sebagai *out-of-bag (OOB) error estimate*. Saat membangun pohon, sekitar 2/3 data asli digunakan dalam sampel bootstrap, sementara 1/3 sisanya digunakan untuk menguji kinerja pohon tersebut. OOB error estimation adalah rata-rata dari kesalahan prediksi untuk setiap contoh data pelatihan yang tidak termasuk dalam sampel bootstrap pohon tersebut.

Dalam pembuatan Random Forest, semua kasus pelatihan diuji melalui setiap pohon, dan matriks kedekatan dihitung berdasarkan pasangan kasus yang mencapai simpul terminal yang sama. Penelitian empiris telah mengkonfirmasi bahwa Random Forest memberikan kinerja prediksi yang baik dalam berbagai konteks, termasuk regresi dan klasifikasi. Algoritma ini telah berhasil digunakan dalam berbagai bidang, seperti prediksi keuangan, analisis citra remote sensing, serta aplikasi di bidang genetika dan biomedis. Lebih jauh, ketika dibandingkan dengan metode lain seperti partial least squares regression, support vector machine, dan neural network, Random Forest sering kali unggul dalam hal kinerja (Kowsari et al., 2019).

### **4.1.3 Support Vector Machine (SVM)**

Menguasai algoritma pembelajaran mesin bukanlah mitos sama sekali. Sebagian besar pemula biasanya memulai dengan mempelajari regresi, yang merupakan pendekatan yang relatif mudah dipahami dan diterapkan. Namun, kita harus menyadari bahwa dunia pembelajaran mesin jauh lebih luas daripada hanya regresi logistik atau masalah regresi. Salah satu konsep yang sering terdengar adalah "*Support Vector Machines*" atau SVM.

Bayangkan algoritma pembelajaran mesin sebagai gudang senjata yang berisi berbagai alat seperti kapak, pedang, busur, belati, dan sebagainya. Setiap alat memiliki kegunaannya sendiri, tetapi yang perlu kita pelajari adalah kapan dan bagaimana cara menggunakan alat tersebut dengan tepat. Sebagai contoh, kita bisa menganggap 'Regresi' sebagai pedang yang efisien dalam memotong dan memahami data, namun tidak selalu mampu menangani data yang sangat kompleks. Inilah saatnya '*Support Vector Machines*' atau SVM menjadi relevan. SVM, seperti pisau tajam, dapat bekerja dengan baik pada dataset yang lebih kecil tetapi kompleks, dan dalam beberapa kasus, ia dapat menjadi alat yang lebih kuat dalam membangun model pembelajaran mesin.

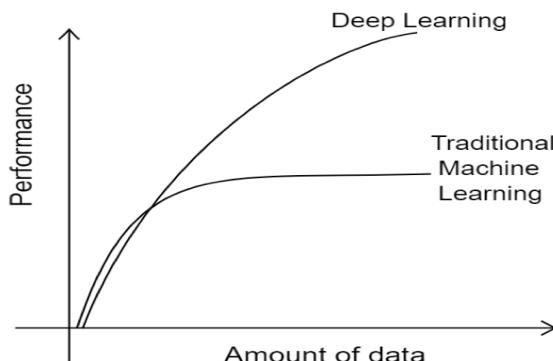
Sebagai contoh kasus, mari kita lihat ilustrasi berikut. Perusahaan Dream Housing Finance bergerak di bidang semua jenis pinjaman rumah. Mereka memiliki kehadiran di seluruh daerah perkotaan, semi-perkotaan, dan pedesaan. Seorang pelanggan pertama-tama mengajukan pinjaman rumah; setelah itu, perusahaan memvalidasi kelayakan pelanggan untuk pinjaman. Perusahaan ini ingin mengotomatisasi proses kelayakan pinjaman (real-time) berdasarkan rincian pelanggan yang diberikan saat mengisi formulir aplikasi online. Rincian ini meliputi Jenis Kelamin, Status Pernikahan, Pendidikan, Jumlah Tanggungan, Pendapatan, Jumlah Pinjaman, Riwayat Kredit, dan lain-lain. Untuk mengotomatisasi proses ini, mereka memberikan masalah dalam mengidentifikasi

segmen pelanggan yang memenuhi syarat untuk jumlah pinjaman sehingga mereka dapat secara khusus menargetkan pelanggan-pelanggan ini.

## 4.2 Deep Learning

Pembelajaran mendalam (*Deep Learning*) menjadi populer saat ini karena kemampuannya dalam meningkatkan komputasi berkinerja tinggi berkat implementasi dari jaringan saraf tiruan. Kemampuan dalam memproses sejumlah data besar pada data yang tidak terstruktur menjadi kekuatan dan fleksibilitas teknik *deep learning*. Algoritma *deep learning* melewati data melalui beberapa lapisan yaitu setiap lapisan mampu mengekstraksi fitur secara progresif dan meneruskannya kedalam lapisan selanjutnya. Lapisan pertama mengestraksi fitur tingkat rendah dan lapisan berikutnya menggabungkan fitur tersebut untuk membentuk representasi secara lengkap (Mathew dkk., 2020).

Kinerja algoritma tradisional *machine Learning* menjadi stabil saat mencapai ambang data pelatihan, sedangkan *deep learning* dalam meningkatkan kinerja dengan jumlah data. Berikut gambaran kinerja dari algoritma tradisional *machine Learning* dan *deep learning* seperti pada Gambar 4.3.



**Gambar 4.3** Gambaran kinerja algoritma *traditional machine Learning* dan *deep learning algorithm* (Goyal dkk., 2018)

#### **4.2.1 Bidirectional Long Short Term Memory Network (BiLSTM)**

Model BiLSTM (*Bidirectional Long Short-Term Memory*) adalah jenis model jaringan saraf tiruan yang digunakan dalam bidang NLP dan tugas-tugas lain yang melibatkan urutan data, seperti pemrosesan teks, pengenalan ucapan, dan sebagainya. BiLSTM adalah ekspansi dari model LSTM yang biasa digunakan dalam tugas-tugas berurutan.

Pembelajaran dalam kedalaman telah mencapai kinerja yang mengesankan dalam berbagai tugas pemrosesan bahasa alami. RNN memperlakukan kalimat sebagai serangkaian kata, mempertimbangkan konteks saat mereka memodelkan struktur kalimat (Kalchbrenner dkk., 2014). Dong (tahun) mengusulkan jaringan saraf rekursif adaptif mandiri yang dapat memahami hubungan sentimen antara kata-kata dengan menggunakan struktur sentimen dan informasi konteks dkk., 2014). Namun, RNN mengalami kendala dalam pemodelan kalimat yang panjang. Untuk mengatasi masalah ini, diusulkan Model LSTM (Sundermeyer dkk., 2012), yang memiliki kemampuan untuk menjaga informasi urutan dan menghasilkan hasil yang kuat dalam berbagai tugas pemodelan urutan.

Berikut adalah beberapa konsep penting dalam model BiLSTM:

1. LSTM. LSTM adalah jenis unit dalam jaringan saraf yang dirancang khusus untuk menangani masalah vanishing gradient dalam pelatihan jaringan saraf berlapis dalam urutan data. LSTM memiliki kemampuan untuk "mengingat" informasi dalam jangka panjang dan "melupakan" informasi yang tidak relevan dalam urutan data.
2. Bidirectional: Model BiLSTM menggabungkan dua lapisan LSTM, satu yang beroperasi maju dari awal hingga akhir

urutan data, dan satu lagi yang beroperasi mundur dari akhir hingga awal urutan data. Ini memungkinkan model untuk memahami konteks sebelum dan sesudah suatu titik dalam urutan data, yang seringkali penting dalam pemrosesan bahasa alami. Dengan demikian, model BiLSTM memiliki kemampuan untuk menangkap konteks dua arah dalam urutan data.

3. Penggunaan dalam NLP: Model BiLSTM sering digunakan dalam tugas-tugas NLP seperti pemodelan bahasa, pemahaman teks, dan tugas-tugas berurutan lainnya. Mereka dapat digunakan untuk tugas seperti pengenalan entitas bernama (NER), analisis sentimen, terjemahan mesin, dan banyak lagi.

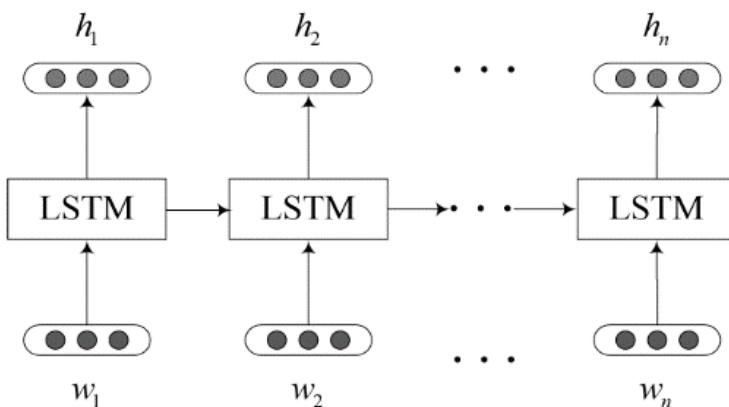
Keuntungan utama dari model BiLSTM adalah kemampuannya untuk menangkap konteks sebelum dan sesudah suatu kata atau token dalam teks, yang seringkali penting untuk pemahaman teks yang lebih baik. Namun, sejak pengenalan model-model yang lebih canggih seperti Transformer, beberapa tugas NLP kini lebih sering menggunakan model Transformer daripada model BiLSTM karena performa yang lebih baik dalam berbagai tugas pemrosesan bahasa.

Mekanisme perhatian mandiri telah diintegrasikan ke dalam model jaringan saraf BiLSTM dengan tujuan meningkatkan kemampuan dalam menangkap informasi yang relevan untuk setiap aspek. Selain itu, telah diajukan model jaringan saraf BiLSTM berbasis perhatian diri yang memiliki penekanan khusus pada istilah-aspek, yang bertujuan untuk menyelesaikan teks pendek yang melibatkan beragam istilah aspek (Xie dkk., 2019). Kami juga mengusulkan sebuah model jaringan saraf BiLSTM berbasis Self-Attention untuk tugas klasifikasi polaritas deteksi ujaran kebencian pada teks pendek. Model yang kami rancang terdiri dari lapisan

encoder kata, lapisan BiLSTM, lapisan perhatian, dan lapisan softmax.

### 1. Lapisan Enkode Kata (Word Encoder Layer)

Setiap kalimat pada lapisan encoder terdiri dari urutan token dan disimbolkan sebagai  $s_i = \{w_{i1}, w_{i2}, \dots, w_{ik}, \dots, w_{in}\}$ , dengan  $w_{ik}$  mewakili kata ke- $k$  dalam kalimat ke- $i$  dan  $n$  menggambarkan panjang kalimat. Setiap kata dalam kalimat diubah menjadi vektor berdimensi  $d$ , yang dikenal sebagai vektor penanaman kata (Pennington dkk., 2014). Setelah semua kata dalam kalimat diwakili oleh vektor dengan dimensi  $d$ , kami membangun matriks *embedding* kata  $S_{n \times d}$ , di mana  $n$  mewakili panjang kalimat dan  $d$  mengindikasikan ukuran *embedding*. Matriks *embedding* kata dianggap sebagai parameter dari model jaringan saraf. Kemudian, matriks *embedding* kata tersebut digunakan sebagai input untuk model jaringan saraf BiLSTM, yang bertujuan untuk mengkodekan kalimat.



**Gambar 4. 4** Arsitektur Umum dari LSTM

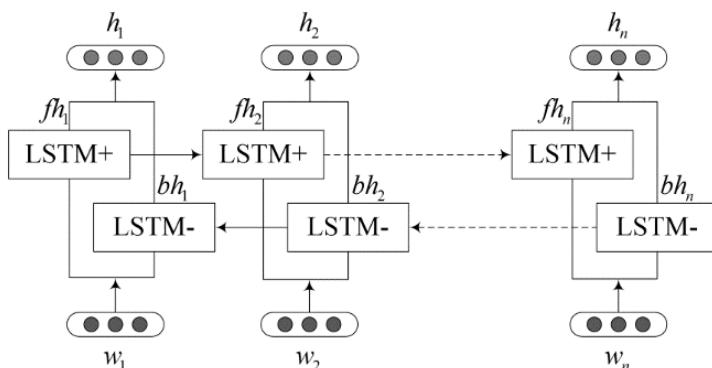
### 2. Lapisan BiLSTM

LSTM merupakan perkembangan dari RNN yang dirancang untuk mengatasi masalah hilang atau meledaknya gradien

yang sering muncul dalam RNN konvensional. Jaringan saraf LSTM terdiri dari tiga gerbang dan sebuah status memori sel. Pada Gambar 4.4 menggambarkan struktur dasar LSTM standar, di mana  $\{w_1, w_2, \dots, w_n\}$  merujuk kepada vektor kata dalam sebuah kalimat, sementara  $\{h_1, h_2, \dots, h_n\}$  menggambarkan vektor tersembunyi.

Dalam model BiLSTM, parameter dari kedua arah yang berlawanan bersifat independen, meskipun mengacu pada kata yang sama dalam sebuah kalimat. Pada Gambar 4.5 menggambarkan struktur dasar model BiLSTM, dengan  $\{w_1, w_2, \dots, w_n\}$  mewakili vektor kata, dan  $n$  adalah panjang kalimat.  $\{f_{h1}, f_{h2}, \dots, f_{hn}\}$  serta  $\{b_{h1}, b_{h2}, \dots, b_{hn}\}$  masing-masing mencerminkan vektor tersembunyi ke arah depan dan vektor tersembunyi ke arah belakang.  $h_n$  adalah singkatan untuk vektor yang digabungkan dari  $f_{hn}$  dan  $b_{hn}$ . Vektor tersembunyi terakhir dari model BiLSTM ditampilkan menggunakan Persamaan 4.3 berikut.

$$h_t = [f_{ht}, b_{ht}] \quad (4.3)$$



**Gambar 4. 5** Arsitektur Umum BiLSTM

Kunci dari LSTM adalah keberadaan sel, dan satu sel LSTM dapat dihitung dengan Persamaan 4.4 sampai 4.9 berikut:

$$X = \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix} \quad (4.4)$$

$$f_t = \sigma(W_f \cdot X + b_f) \quad (4.5)$$

$$i_t = \sigma(W_i \cdot X + b_i) \quad (4.6)$$

$$o_t = \sigma(W_o \cdot X + b_o) \quad (4.7)$$

$$c_t = f_t * c_{t-1} + i_t * \tanh(W_c \cdot X + b_c) \quad (4.8)$$

$$h_t = o_t * \tanh(c_t) \quad (4.9)$$

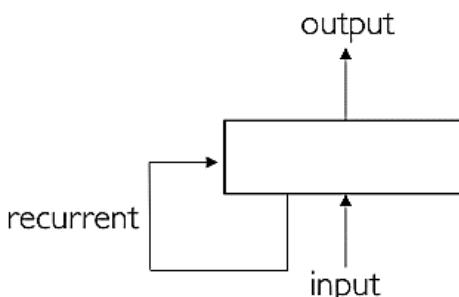
Di mana  $W_f$ ,  $W_i$ ,  $W_o$  adalah matriks bobot, dan  $b_f$ ,  $b_i$ ,  $b_o$  adalah bias dari sel LSTM selama pelatihan. Masing-masing dari mereka dianggap sebagai parameter untuk gerbang input, gerbang lupa, dan gerbang output. Fungsi sigmoid ditunjukkan oleh  $\sigma$  dan  $*$  mewakili perkalian berdasarkan elemen. Sedangkan  $x_t$  adalah singkatan dari vektor penanaman kata sebagai unit input ke dalam LSTM, sementara  $h_t$  adalah vektor tersembunyi. Dengan demikian,  $h_n$  dapat digunakan untuk mewakili sebuah kalimat.

### 3. Lapisan Perhatian (Attention Layer)

Polaritas sentimen dalam sebuah kalimat tidak hanya dipengaruhi oleh konteks saja, tetapi juga sangat terkait dengan istilah opini dan aspek-aspek yang ada. Meskipun demikian, dalam sebuah kalimat, tidak semua kata konteks memiliki kontribusi semantik yang sama. Untuk mengatasi masalah ini, beberapa peneliti menggunakan mekanisme perhatian diri untuk mengekstraksi kata-kata yang lebih signifikan dengan memberikan bobot yang lebih tinggi, sehingga meningkatkan relevansinya (Xie et al., 2019).

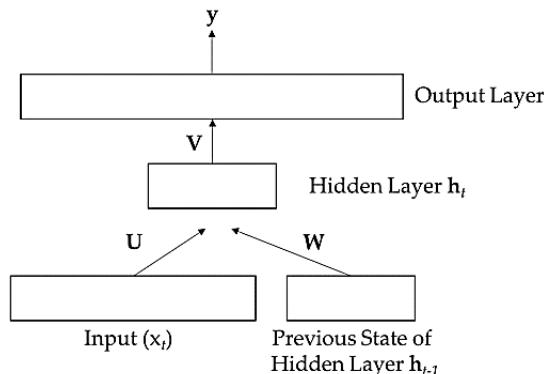
#### 4.2.2 Recurrent Neural Network (RNN)

Ide dasar dari jaringan saraf rekuren (Recurrent Neural Network/RNN) adalah menciptakan pola jaringan yang mampu mengatasi data berurutan atau deret waktu, seperti data perkiraan cuaca. Dalam hal ini, kondisi cuaca saat ini sangat terkait dengan kondisi cuaca sebelumnya. Sebagai contoh, jika cuaca sebelumnya adalah mendung, kemungkinan besar hari ini akan hujan. Walaupun ada perbedaan antara data berurutan dan deret waktu, RNN berfokus pada sifat data yang mengindikasikan bahwa informasi pada waktu sebelumnya ( $t-1$ ) memengaruhi informasi pada waktu selanjutnya ( $t$ ). Intinya, RNN mampu mengingat sejarah data. Secara lebih umum, jika kita diberikan serangkaian input  $x = (x_1; \dots; x_T)$ , data  $x_t$  (seperti vektor, gambar, teks, atau suara) dipengaruhi oleh data sebelumnya (sejarah) dan dapat dituliskan sebagai  $P(x_t | f(x_1; \dots; x_{t-1}))$ . Pada dasarnya, RNN bertujuan untuk mengingat, di mana kita ingin mengingat seluruh sekuens (berbeda dengan asumsi Markov yang hanya mengingat sekuens secara terbatas), sehingga RNN mampu mengenali ketergantungan panjang, seperti  $x_t$  yang dapat bergantung pada  $x_1$ . RNN paling dasar menggambarkan dirinya sendiri seperti pointer yang merujuk pada dirinya sendiri. Berikut pada Gambar 4.6 menunjukkan bentuk konseptual dari RNN.



**Gambar 4. 6** Bentuk konseptual paling sederhana Recurrent Neural Network

Ilustrasi ini mungkin sulit dipahami karena terlihat sangat konseptual. Secara lebih matematis diilustrasikan pada Gambar 4.7 sebagai berikut. Perhitungan keadaan tersembunyi (hidden state) pada waktu ke- $t$  bergantung pada input pada waktu ke- $t$  ( $x_t$ ) dan keadaan tersembunyi pada waktu sebelumnya ( $h_{t-1}$ ).



**Gambar 4. 7** Konsep Recurrent Neural Network

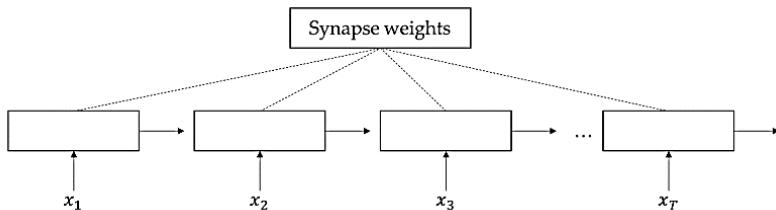
Prinsipnya adalah mengingat (memorialisasi) kejadian sebelumnya. RNN dapat direpresentasikan dengan Persamaan 4.10 berikut.

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}) \quad (4.10)$$

Dimana  $f$  adalah fungsi aktivasi (non-linear dan dapat diubah). Untuk penyederhanaan, kita tidak akan membahas bias ( $b$ ) dalam fungsi-fungsi berikutnya. Prinsip ini juga dapat digunakan pada variasi jaringan saraf seperti long short-term memory network (LSTM).

RNN adalah arsitektur jaringan saraf yang dirancang untuk menangani masalah pada asumsi Markov. Ide utamanya adalah mampu mengingat informasi sebanyak mungkin, berbeda dengan asumsi Markov yang terbatas pada informasi yang lebih baru, sehingga tidak ada informasi yang hilang.

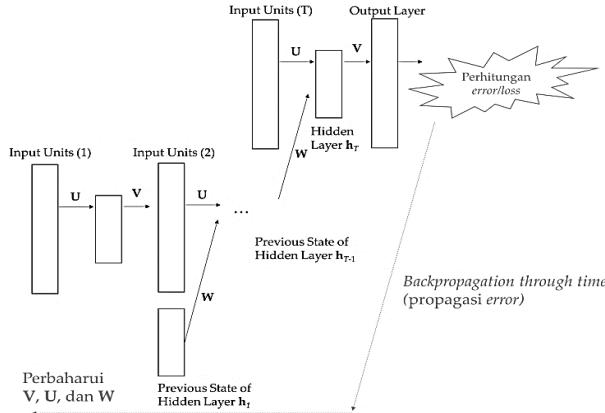
Pelatihan pada jaringan saraf rekuren dapat menggunakan metode backpropagation. Ada metode lain yang disebut "backpropagation through time" yang cocok untuk pelatihan sekuensial atau deret waktu.



**Gambar 4. 8** Konsep feed forward pada Recurrent Neural Network (RNN)

Sebagai contoh seperti pada Gambar 4.8, kita dapat mengambil sekuens input  $x$  dengan panjang  $T$ , di mana  $x_t$  adalah input ke- $i$  (data point) yang bisa berupa vektor, gambar, teks, dan lain sebagainya. Data ini disampaikan ke RNN dalam feed forward, yang kemudian diperbaharui parameter-parameter (bobot sinapsis) melalui backpropagation error.

RNN memiliki prinsip berbagi parameter, di mana neuron yang sama digunakan berulang kali saat proses feed forward. Setelah selesai proses feed forward, parameter diperbaharui berdasarkan propagasi kesalahan (*backpropagation*). Dalam "*backpropagation through time*," parameter diperbaharui setelah mencapai hidden state paling awal (Gambar 4.9).



**Gambar 4. 9 Konsep backpropagation through time**

Walaupun RNN secara konseptual mampu mengingat seluruh informasi sebelumnya, hal ini sulit dilakukan dalam praktik untuk sekuens yang panjang karena masalah "vanishing" atau "exploding" gradient. Untuk mengatasi masalah ini, RNN melakukan representasi ulang sekuens input ke bentuk output vektor  $y$ , yang nantinya digunakan untuk tugas lanjutan, seperti klasifikasi. Bentuk konseptual ini dapat dirumuskan dalam Persamaan 4.11. Biasanya, vektor  $y$  diteruskan ke jaringan saraf tiruan multi-lapisan (MLP) dan fungsi softmax untuk tugas akhir dalam bentuk probabilitas, sebagaimana ditunjukkan dalam Persamaan 4.12.

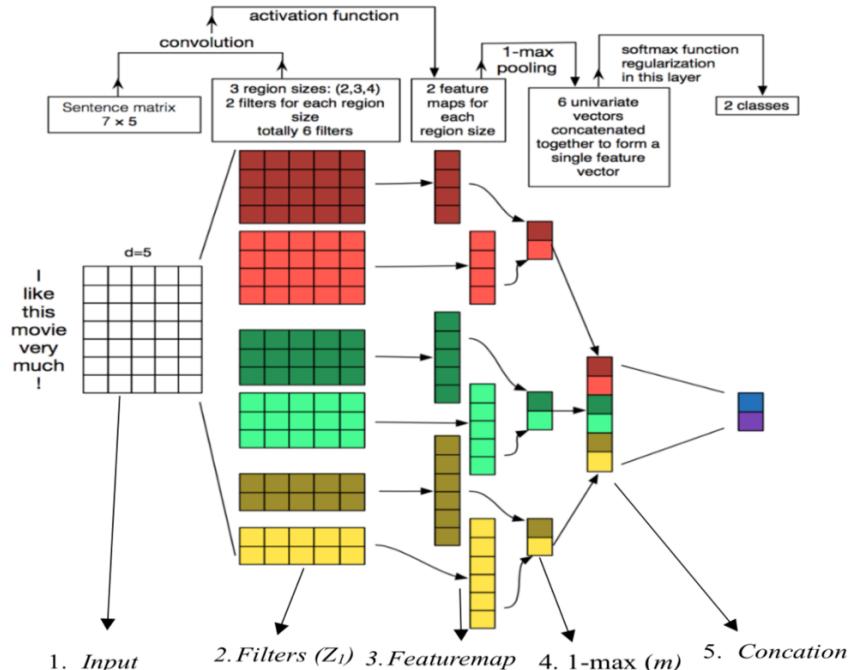
$$y = \text{RNN}(x_1; \dots; x_N) \quad (4.11)$$

$$\text{final output} = \text{softmax}(\text{MLP}(y)) \quad (4.12)$$

#### 4.2.3 Convolutional Neural Network (CNN)

*Convolutional neural network* (CNN) merupakan kelas jaringan saraf tiruan yang termasuk pada pembelajaran dalam (*deep learning*) yang umumnya diimplementasikan pada data citra. Akan tetapi dalam beberapa tahun terakhir ini, model CNN menjadi lebih

populer serta mendapat perhatian yang luas karena tidak hanya digunakan untuk menyelesaikan masalah *image processing* namun juga dalam bidang pengolahan bahasa alami atau *natural language processing* (NLP). CNN telah berhasil dalam berbagai penerapan klasifikasi teks (Brownlee, 2017), (Zhang dan LeCun, 2015).



**Gambar 4. 10** Arsitektur CNN untuk klasifikasi teks (Zhang dan Wallace, 2017)

Pada Gambar 4.10 diatas, bagaimana model CNN mampu melakukan klasifikasi pada teks (Zhang dan Wallace, 2017). Model CNN sederhana dengan sedikit pengaturan *hyperparameter* dan vektor statis mampu mencapai hasil yang sangat baik pada beberapa tolok ukur meningkatkan *state-of-the-art* pada 4 dari 7 tugas.

### 1. *Input* ( $x$ )

*Input* ( $x$ ) yaitu kalimat yang telah di-*encoding* pada proses sebelumnya menjadi matriks  $7 \times 5$ , dan tanda seru diperlakukan sebagai kata.

## 2. *Filters* (f)

Salah satu sifat yang diinginkan dari CNN adalah mempertahankan orientasi spasial 2D dalam visi komputer. Teks, seperti gambar, juga memiliki orientasi. Walaupun 2 dimensi, teks memiliki struktur satu dimensi di mana urutan kata penting. Ukuran wilayah mengacu pada jumlah baris yang mewakili kata dari matriks kalimat yang akan disaring. (Zhang dan Wallace, 2017) memilih untuk menggunakan 2 filter sampai 6 filter komplementer untuk mempertimbangkan 2, 3, dan 4 kata.

## 3. *Featuremap* (c)

Setelah dilakukan perkalian, maka didapat *output* (o) sementara, untuk mendapatkan *featuremap* (c), peneliti menambahkan istilah bias (skalar, misalnya: bentuk  $1 \times 1$ ) dan menerapkan fungsi aktivasi (A1), semisal fungsi aktivasi ReLU.

## 4. *1-max* (m)

Setelah berhasil melakukan penghitungan fitur, maka dilakukan *1-max* (mengambil nilai terbesar untuk setiap fitur), perhatikan bahwa dimensi c bergantung pada s dan h, dengan kata lain, panjang kolom c bervariasi di antara kalimat dengan panjang yang berbeda dan filter dari ukuran wilayah yang berbeda. Untuk mengatasi masalah ini, peneliti menggunakan fungsi penyatuhan *1-max* dan mengekstrak angka terbesar dari setiap vektor c.

## 5. Pengabungan *1-max* (*concatenate 1-max*)

Setelah "*1-max pooling* (m)", didapat vektor dengan panjang yang tetap yakni 6 elemen. Vektor dengan panjang tetap ini kemudian dapat dimasukkan ke dalam fungsi *softmax* agar terhubung penuh. Kesalahan dari klasifikasi kemudian diperbanyak kembali ke parameter berikut sebagai bagian dari pembelajaran:

- a. Matriks yang diproduksi o.
- b. Istilah bias yang ditambahkan ke o untuk menghasilkan c.
- c. Vektor kata (opsional, menggunakan kinerja data validasi).

#### 4.2.4 Bidirectional Gated Recurrent Unit (BiGRU)

Jaringan saraf BiGRU (*bidirectional gated recurrent unit*) adalah jaringan saraf GRU yang ditingkatkan dengan struktur dua lapis. Struktur dua lapis ini menyediakan lapisan output dengan informasi kontekstual yang lengkap dari informasi input setiap saat. Ide dasar dari jaringan saraf BiGRU adalah bahwa urutan masukkan dilewatkan melalui jaringan saraf maju dan jaringan saraf mundur, dan kemudian, luaran dari keduanya terhubung dalam lapisan output yang sama (Miao dkk., 2019). Pada jaringan saraf BiGRU, setiap lapisan maju menghitung output lapisan tersembunyi setiap saat dari depan ke belakang, dan lapisan belakang menghitung output dari lapisan tersembunyi pada setiap waktu dari belakang ke depan.

Model GRU dilengkapi dengan dua gerbang yaitu gerbang pembaruan dan gerbang reset. Berbeda dengan LSTM, GRU menggunakan gerbang pembaruan alih-alih gerbang untuk mewakili pengaruh keadaan tersembunyi sebelumnya pada keadaan tersembunyi saat ini. Tingkat kontrol tumbuh seiring dengan meningkatnya skor gerbang pembaruan. Gerbang pengaturan ulang mewakili tingkat pengabaian output neuron lapisan tersembunyi pada waktu sebelumnya. Lebih sedikit informasi yang hilang karena kecepatan gerbang penyetelan ulang meningkat (Sindhu dkk., 2021). Spesifik proses perhitungan pada model GRU seperti persamaan berikut ini.

$$Z_t = \sigma(W_i * [h_{t-1}, x_t]) \quad (4.13)$$

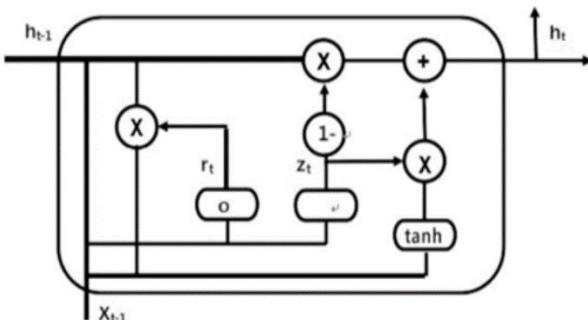
$$r_t = \sigma(W_r * [h_{t-1}, x_t]) \quad (4.14)$$

$$Z_t = \sigma(W_i * [h_{t-1}, x_t]) \quad (4.15)$$

$$h_t = \tanh(W_c * [r_t * h_{t-1}, x_t]) \quad (4.16)$$

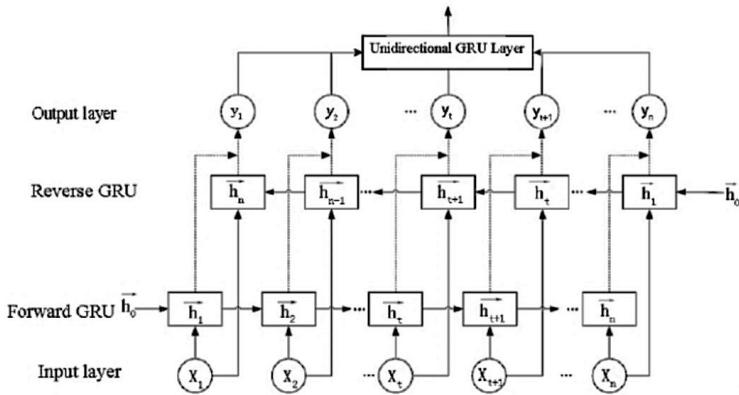
$$h_t = (1 - z_t) * c_{t-1} + z_t * \tilde{h}_t \quad (4.17)$$

Dimana,  $z_t$  pada persamaan (4.13) mewakili gerbang pembaruan yang mewakili fungsi Sigmoid,  $r_t$  pada persamaan (4.14) mewakili gerbang reset,  $W_r$  dan  $W_i$  pada persamaan diatas mewakili semua matriks parameter pelatihan,  $\tanh$  pada persamaan (4.15) mewakili fungsi tangen hiperbolik,  $h_{t-1}$  mewakili luaran neuron lapisan tersembunyi di momen sebelumnya,  $h_{\sim t}$  mewakili status aktivasi kandidat,  $x_t$  mewakili masukkan saat ini, dan  $W_c$  dan  $U$  mewakili matriks parameter pelatihan. Berikut secara umum struktur dari unit GRU seperti pada Gambar 4.11.



**Gambar 4. 11** Struktur unit GRU

Jaringan BiGRU memiliki kapasitas untuk mempelajari hubungan antara faktor-faktor yang mempengaruhi beban masa lalu dan masa depan dan beban saat ini, yang lebih kondusif untuk mengekstraksi fitur-fitur dalam dari data beban (Qiu dkk., 2020). Seperti dapat dilihat dari Gambar 4.12, jaringan saraf BiGRU terdiri dari dua lapisan GRU. Terdapat dua nilai output pada lapisan tersembunyi, satu untuk output *forward* dan yang lainnya untuk output *reverse*. Proses perhitungannya mirip dengan jaringan GRU satu arah. Perhitungan luaran GRU *forward* dan *reverse* seperti terlihat pada Persamaan (4.18) dan (4.19):



**Gambar 4. 12** Struktur diagram BiGRU (Miao dkk., 2019)

$$\vec{Y}_{ft} = \sigma(W_{ft}x_t + U_{ft}\vec{Y}_{ft-1}) \quad (4.18)$$

$$\vec{Y}_{bt} = \sigma(W_{bt}x_t + U_{bt}\vec{Y}_{bt-1}) \quad (4.19)$$

Pada persamaan (4.18) dan (4.19),  $Y_{ft-1}$  dan  $Y_{bt-1}$  masing-masing mewakili output dari lapisan tersembunyi dari GRU *forward* dan *reverse* pada langkah  $t-1$  yang mana mewakili fungsi aktivasi Sigmoid.  $X_t$  mewakili masukkan dari langkah  $t$ .  $W_{ft}$  dan  $W_{bt}$  mewakili masukkan  $U$  dan  $U_{bt}$  mewakili matriks bobot lapisan tersembunyi dari matriks bobot maju GRU positif dan negatif pada langkah waktu  $t$  dan membalikkan GRU pada langkah waktu  $t-1$ .

# BAB V

## METRIK EVALUASI

### 5.1 Confusion Matrix

*Confusion matrix* adalah tabel yang secara khusus menggambarkan kinerja suatu model atau algoritma. Dalam matriks ini, setiap baris merepresentasikan kelas aktual data, sementara setiap kolom merepresentasikan kelas prediksi dari data (Saputro & Sari, 2020). Dalam konteks, confusion matrix digunakan untuk mengevaluasi klasifikasi sentimen, seperti yang ditampilkan dalam Tabel 5.1.

**Tabel 5. 1** Confusion Matrix

|                     |                 | Kelas Prediksi                |                               |
|---------------------|-----------------|-------------------------------|-------------------------------|
|                     |                 | <i>Positive</i>               | <i>Negative</i>               |
| <b>Kelas Aktual</b> | <i>Positive</i> | <i>True Positive</i><br>(TP)  | <i>False Negative</i><br>(FN) |
|                     | <i>Negative</i> | <i>False Positive</i><br>(FP) | <i>True Negative</i><br>(TN)  |

Keterangan:

*True Positive (TP)* : data kelas positif yang diprediksi dengan benar sebagai kelas positif.

*False Positive (FP)*: data kelas negatif yang salah diprediksi sebagai kelas positif.

*True Negative (TN)*: data kelas negatif yang diprediksi dengan benar sebagai kelas negatif.

*False Negative (FN)*: data kelas positif yang salah diprediksi sebagai kelas negatif.

Nilai-nilai yang terdapat dalam *confusion matrix* digunakan untuk menghitung berbagai metrik evaluasi, seperti *precision*, *recall*, *F1-Score*, dan akurasi. *Precision* adalah rasio antara jumlah data positif yang diklasifikasikan dengan benar dan total jumlah data

yang diklasifikasikan sebagai positif. *Recall* adalah rasio antara jumlah data positif yang diklasifikasikan dengan benar dan total jumlah data positif (Amrutha dan Bindu, 2019). *F1-Score*, atau kadang disebut *F-measure*, adalah nilai rata-rata harmonik dari *recall* dan *precision* (Sethy dan Ramabhadran, 2008). Sementara itu, akurasi adalah rasio antara jumlah prediksi yang benar dengan total data yang diprediksi.

Rumus perhitungan untuk mendapatkan nilai *precision*, *recall*, *F1-Score*, dan akurasi dapat ditemukan dalam persamaan 5.1 hingga 5.4 seperti yang dijelaskan dalam sumber-sumber yang disebutkan.

$$\text{Accuracy} = \frac{(TP+TN)}{(TP+FP+FN+TN)} \quad (5.1)$$

$$\text{Precision} = \frac{TP}{TP+FP} \quad (5.2)$$

$$\text{Recall} = \frac{TP}{TP+FN} \quad (5.3)$$

$$\text{F1 Score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \quad (5.4)$$

Mari kita lihat ilustrasi berikut untuk dapat memahami proses dalam confusion matrix.

- True Positive (TP):
  - ✓ Penjelasan: Anda menghasilkan prediksi positif dan itu tepat.
  - ✓ Misalnya, Anda memprediksi bahwa seseorang memiliki penyakit dan prediksi tersebut terbukti benar, orang tersebut memang menderita penyakit.
- True Negative (TN):
  - ✓ Penjelasan: Anda menghasilkan prediksi negatif dan itu tepat.
  - ✓ Contohnya, Anda memprediksi bahwa suatu produk tidak cacat, dan kenyataannya produk tersebut memang tidak mengalami cacat.

- False Positive (FP): (Kesalahan Tipe 1)
  - ✓ Penjelasan: Anda menghasilkan prediksi positif, namun ternyata itu tidak benar.
  - ✓ Sebagai contoh, Anda memprediksi bahwa seseorang memiliki ketergantungan terhadap zat tertentu, padahal kenyataannya orang tersebut tidak memiliki ketergantungan.
- False Negative (FN): (Kesalahan Tipe 2, kesalahan tipe 2 ini sangat berbahaya)
  - ✓ Penjelasan: Anda menghasilkan prediksi negatif, namun kenyataannya sebaliknya.
  - ✓ Misalnya, Anda memprediksi bahwa suatu obat aman digunakan, tetapi kenyataannya obat tersebut memiliki efek samping yang berbahaya.

Dari ilustrasi sebelumnya, dapat dijelaskan bahwa:

- ✓ Nilai Prediksi merupakan output dari program, yang dapat berupa Positif atau Negatif.
- ✓ Nilai Aktual merupakan nilai yang sebenarnya, dengan kemungkinan nilai True atau False.

Sekarang, mari kita lihat contoh implementasinya dalam sebuah kasus. Misalnya, sebuah perusahaan mengembangkan model yang dilatih untuk memprediksi apakah seorang karyawan di perusahaan tersebut akan berhasil dalam pelatihan keterampilan tertentu atau tidak. Dengan asumsi perusahaan memiliki total 175 karyawan, hasil dari model klasifikasi menunjukkan bahwa karyawan yang diprediksi berhasil dalam pelatihan sebanyak 145, sementara karyawan yang diprediksi tidak berhasil sebanyak 30. Namun, pada kenyataannya, terdapat 150 karyawan yang berhasil dan 25 karyawan yang tidak berhasil dalam pelatihan (Tabel 5.2).

**Tabel 5. 2** Contoh Confusion Matrix

| n= 175                | Aktual: Positif (1) | Aktual: Negatif (0) |
|-----------------------|---------------------|---------------------|
| Predksi: Positif (1)  | TP: 125             | FP: 20              |
| Prediksi: Negatif (0) | FN: 25              | TN: 5               |
| Total                 | <b>150</b>          | <b>25</b>           |

Dalam Tabel 5.2 Contoh Confusion Matrix dengan total karyawan (n) sebanyak 175, dapat dijelaskan sebagai berikut:

- True Positive (TP): kita memprediksi karyawan berhasil dalam pelatihan dan memang benar karyawan tersebut berhasil.
- True Negative (TN): kita memprediksi karyawan tidak berhasil dalam pelatihan dan memang benar karyawan tersebut tidak berhasil.
- False Positive (FP): kita memprediksi karyawan berhasil dalam pelatihan, tetapi ternyata prediksi salah, karyawan tersebut sebenarnya tidak berhasil.
- False Negative (FN): kita memprediksi karyawan tidak berhasil dalam pelatihan, tetapi ternyata prediksi salah, karyawan tersebut sebenarnya berhasil. Kesalahan tipe 2 ini dapat mengakibatkan karyawan yang dianggap tidak berhasil padahal sebenarnya berhasil, dengan konsekuensi kurangnya pengakuan terhadap prestasi mereka.

Dengan demikian, nilai accuracy, precision, recall, dan F-1 score dapat dihitung sebagai berikut:

- Accuracy menggambarkan seberapa akurat model dalam mengklasifikasikan dengan benar.

$$\text{Accuracy} = \frac{TP+TN}{TP+FP+FN+TN} = \frac{125+5}{125+20+25+5} = 0.742 (74.2\%)$$

- Precision menggambarkan akurasi antara data yang diminta dengan hasil prediksi yang diberikan oleh model.

$$\text{Precision} = \frac{TP}{TP+FP} = \frac{125}{125+20} = 0.86 (86\%)$$

- Recall atau sensitivity menggambarkan keberhasilan model dalam menemukan kembali sebuah informasi.

$$Recall = \frac{TP}{TP+FN} = \frac{125}{125+25} = 0.83 (83\%)$$

- F-1 Score menggambarkan perbandingan rata-rata precision dan recall yang dibobotkan.

$$F1\ Score = \frac{2 \times Recall \times Precision}{Recall + Precision} = \frac{2 \times 0.83 \times 0.86}{0.83 + 0.86} \\ = 0.84 \times 100\% = \mathbf{84\%}$$

## 5.2 Micro-Averaged F1-measure

*Micro-Averaged F1-measure* dapat digunakan untuk mengevaluasi kinerja model pada masalah kelas multilabel (Salminen dkk., 2018). Dalam konteks kelas multilabel, setiap sampel dapat terhubung dengan satu atau lebih kelas, dan model harus melakukan prediksi untuk setiap kelas yang mungkin. *Micro-Averaged F1-measure* mengukur *precision* dan *recall* secara keseluruhan terhadap semua kelas yang ada, tanpa memperhatikan kelas-kelas secara individual. Metrik ini berguna untuk mendapatkan pemahaman tentang sejauh mana model mampu mengklasifikasikan semua kelas dengan benar secara keseluruhan.

*Micro-Averaged F1-measure* adalah sebuah turunan dari metrik evaluasi *F1-Score*. Metrik ini menghitung total *True Positive (TP)*, *False Positive (FP)*, dan *False Negative (FN)* pada tabel *Confusion Matrix*, tanpa mempertimbangkan setiap kelas secara individual, melainkan secara keseluruhan (Ayo dkk., 2020b). Sebaliknya, perhitungan nilai *Precision* dan *Recall* biasanya dilakukan untuk tiap kelas secara terpisah, sementara *Micro-Averaged F1-measure* melakukan perhitungan ini secara menyeluruh terhadap semua kelas yang ada. Persamaan dari *Precision* dan *Recall* untuk *Micro-Averaged F1-measure* dapat dilihat pada Persamaan 5.5 dan 5.6.

$$Precision_{(Micro Avg. F1 measure)} = \frac{\sum_{i=1}^N TP_i}{\sum_{i=1}^N TP_i + FP_i} \quad (5.5)$$

$$Recall_{(Micro Avg. F1 measure)} = \frac{\sum_{i=1}^N TP_i}{\sum_{i=1}^N TP_i + FN_i} \quad (5.6)$$

Perbedaan yang utama antara *Micro-Averaged F1-measure* dengan variasi *F1-Score* lainnya terletak pada pendekatan dalam menghitung *True Positive (TP)*, *False Positive (FP)*, dan *False Negative (FN)*. Walaupun demikian, rumus yang digunakan untuk menghitung *Micro-Averaged F1-measure* pada dasarnya tetap mirip dengan rumus *F1-Score* yang lebih umum. Persamaan yang menggambarkan perhitungan *Micro-Averaged F1-measure* dapat ditemukan pada Persamaan 5.7 (Liu dkk., 2010).

$$Micro Averaged F1 Measure = \frac{2 \times Precision \times Recall}{(Precision + Recall)} \quad (5.7)$$

### 5.3 Area Under the ROC Curve (AUC – ROC)

AUC-ROC adalah metrik yang mengukur sejauh mana model mampu memisahkan kelas positif dan negatif. Proses ini diukur dengan menghitung area di bawah kurva ROC, yang merupakan grafik yang menggambarkan hubungan antara tingkat *True Positive Rate (TPR)* dan tingkat *False Positive Rate (FPR)* pada berbagai ambang batas prediksi (Ayo dkk, 2021). Metrik AUC-ROC digunakan untuk menilai kinerja model klasifikasi dengan membandingkan *sensitivity (True Positive Rate)* dan *specificity (False Positive Rate)*. Berbeda dengan *confusion matrix*, dimana memberikan informasi rinci tentang jenis kesalahan yang dibuat oleh model, seperti kesalahan dalam memprediksi positif atau negatif. Akan tetapi, kedua metrik ini sering digunakan bersama-sama untuk memberikan pemahaman yang lebih lengkap tentang kinerja model klasifikasi. Untuk melakukan perhitungan TPR dan FPR dapat menggunakan persamaan 5.8 dan 5.9.

Untuk menghitung AUC-ROC, langkah-langkah berikut dapat diikuti:

1. Kumpulkan hasil prediksi model dan label sebenarnya dari data yang diuji.
2. Hitung *True Positive Rate* (TPR) dan *False Positive Rate* (FPR) dengan menggunakan rumus berikut:

$$TPR = \frac{TP}{(TP+FN)} \quad (5.8)$$

$$FPR = \frac{FP}{(FP+TN)} \quad (5.9)$$

dimana, TP adalah jumlah *True Positives*, FN adalah jumlah *False Negatives*, FP adalah jumlah *False Positives*, dan TN adalah jumlah *True Negatives*.

3. Buat kurva ROC dengan TPR sebagai sumbu Y dan FPR sebagai sumbu X, mencakup berbagai ambang batas prediksi.
4. Hitung luas area di bawah kurva ROC, yang disebut AUC (*Area Under the Curve*) ROC.

Nilai AUC-ROC berada dalam rentang antara 0 hingga 1. Semakin besar nilainya, semakin baik kemampuan model dalam membedakan antara kelas positif dan negatif. Sebaliknya, nilai 0,5 mengindikasikan kinerja model yang sama dengan hasil lemparan koin (Kumar dan Sachdeva, 2020).

Berdasarkan hasil perhitungan confusion matrix pada contoh kasus sebelumnya, mari kita lihat perhitungan untuk menghitung nilai AUC-ROC.

$$TPR = \frac{125}{125+25} = \frac{125}{150} = 0.8333$$

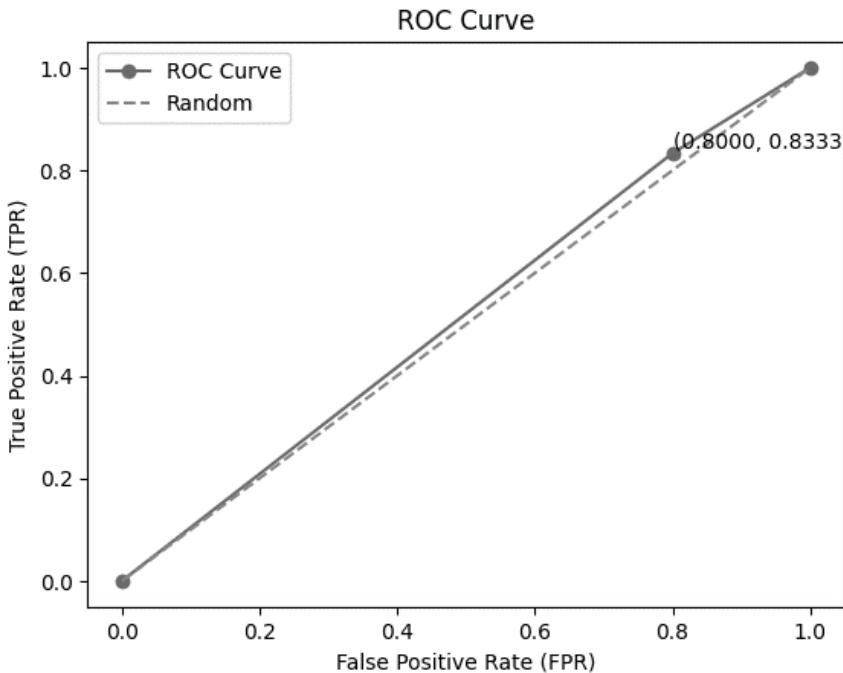
$$FPR = \frac{20}{20+5} = \frac{20}{25} = 0.8$$

Setelah itu, kita dapat membuat kurva ROC dan menghitung AUC-ROC. Namun, karena kita hanya memiliki satu titik pada kurva ROC dalam contoh Anda, kita dapat menyimpulkan bahwa

AUC-ROC dalam kasus ini adalah luas segitiga yang dibentuk oleh titik  $(0,0)$ ,  $(FPR, TPR)$ , dan  $(1,1)$ . Mari hitung luasnya:

$$AUC - ROC = \frac{1}{2} \times FPR \times TPR = \frac{1}{2} \times 0.8 \times 0.8333 = 0.3333$$

Hasil ini menunjukkan bahwa AUC-ROC dari model pada satu titik tertentu adalah sekitar 0.3333. Penting untuk dicatat bahwa interpretasi AUC-ROC yang lebih signifikan dan bermakna dapat diperoleh dengan mengukur kurva ROC dari berbagai threshold, yang dapat menghasilkan kurva ROC yang lebih lengkap dan akurat. Sehingga ketika buatkan dalam bentuk kurva, maka akan tampil seperti pada Gambar 5.1 berikut.



**Gambar 5. 1** Hasil Kurva AUC-ROC

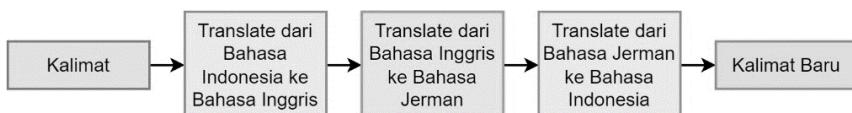
## **BAB VI**

### **STUDI KASUS: EKSPANSI SEMANTIK**

Pada Bab ini, kita akan membahas mengenai pendekatan ekspansi semantik untuk teks pendek. Penambahan variasi kata ini akan memperkaya informasi dari kalimat. Ekspansi semantik pada teks pendek memiliki peran yang sangat penting dalam pemahaman dan analisis teks. Dalam teks pendek, informasi sering kali tersirat atau tersembunyi di balik kata-kata, dan ekspansi semantik membantu dalam mengungkapkan makna yang lebih dalam dan kompleks. Hal ini memungkinkan untuk memahami konteks, mengidentifikasi hubungan yang relevan antar kata atau konsep, mengatasi ambiguitas, dan mengenali sentimen serta nuansa emosi yang terkandung dalam teks. Selain itu, ekspansi semantik juga mendukung pengenalan entitas, pengenalan perbandingan dan kontrast, serta pemahaman pesan yang berisi ironi atau sarkasme. Dengan demikian, ekspansi semantik pada teks pendek menjadi kunci dalam berbagai aplikasi pemrosesan bahasa alami dan analisis teks, menghasilkan interpretasi yang lebih akurat dan relevan. Pada bagian ini, kita akan melihat lebih jauh tentang proses yang dapat digunakan dalam memperkaya fitur semantik melalui proses back-translation, disambigu kata, dan ekspansi teks. Selain itu, pembahasan mengenai pengukuran kemiripan semantik akan kita bahas secara sederhana dalam bagian ini. Umumnya, dalam melakukan memperkaya semantik harus ada target kata. Target kata dari suatu kalimat dapat dideteksi sebagai kata sifa (*adj*), kata kerja (*verb*), dan kata benda (*noun*) yang memiliki makna serupa atau sinonim.

## 6.1 Back-Translation

Back-translation adalah proses menerjemahkan teks dari satu bahasa ke bahasa lain, dan kemudian menerjemahkannya kembali ke bahasa asalnya. Tujuan dari back-translation adalah untuk menguji keakuratan dan ketepatan terjemahan, serta memastikan bahwa makna teks tetap sama setelah melalui proses terjemahan dua kali. Proses ini sering digunakan dalam evaluasi kualitas terjemahan, terutama ketika bekerja dengan model penerjemahan mesin atau dalam penelitian linguistik. Mari kita lihat alur proses back-translation pada Gambar 6.1. Hasil terjemahan ini memungkinkan untuk mendapatkan kalimat yang berbeda namun masih mempertahankan makna sama.



**Gambar 6. 1** Proses Back-Translation

Metode back-translation umumnya juga digunakan sebagai salah satu teknik augmentasi data, di mana untuk menambah jumlah data. Proses ini melibatkan penerjemahan teks dari bahasa sumber ke bahasa target, dan kemudian menerjemahkannya kembali ke bahasa sumber (lihat pada Tabel 6.1). Hal ini bertujuan untuk menghasilkan data tambahan yang berbeda namun tetap mempertahankan makna yang sama, meningkatkan variasi dalam kumpulan data. Metode back-translation telah terbukti efektif dalam meningkatkan kualitas kalimat dan jumlah data yang digunakan dalam pembelajaran mesin, khususnya untuk dataset yang sedikit.

**Tabel 6. 1** Contoh penerjemahan balik kalimat dari bahasa Indonesia ke Inggris ke Indonesia

| Kalimat asli   | Encoder from Indonesia to English                                    | Hasil Decoder from English to Indonesia                      |
|--|--|--|
| Wakil rakyat kok goblok  | People's representatives are stupid                                  | Wakil Rakyat itu bodoh                                       |
| Bagusnya bubarin aja anggota yang gak mutu.                            | It's better to just disband members who don't meet the requirements. | Lebih baik bubarkan saja anggota yang tidak memenuhi syarat. |
| Kalian dari awal udah keliatan goblok dan kelihatan makin goblok lagi. | You've been stupid from the start and you look even more stupid.     | Kamu sudah bodoh sejak awal dan kamu terlihat lebih bodoh.   |

Proses back-translation memainkan peran penting dalam meningkatkan kualitas dan keragaman kalimat. Dengan memperkenalkan variasi kalimat baru, model deteksi dapat lebih baik dalam mengenali pola ujaran kebencian yang beragam. Hal ini juga membantu mengatasi keterbatasan jumlah data yang mungkin tersedia, yang dapat menjadi masalah dalam pengembangan model yang handal. Data yang telah ditingkatkan dengan teknik back-translation kemudian diintegrasikan ke dalam dataset pelatihan, sehingga meningkatkan ketersediaan dan keragaman data latihan.

Namun, penting untuk mempertimbangkan bahwa proses back-translation dapat memunculkan masalah tersendiri, terutama dalam menjaga konsistensi dan akurasi makna kalimat. Hasil terjemahan kembali ke bahasa Indonesia mungkin mengalami distorsi atau perubahan dalam makna aslinya. Oleh karena itu, metode ini perlu diterapkan dengan hati-hati dan hasil terjemahan perlu dievaluasi untuk memastikan bahwa variasi yang dihasilkan masih konsisten dengan konteks semula. Penerapan back-translation dapat dilakukan menggunakan *script* berikut.

## # Pendefinisian Back-translation

```
def back_translate(text, target_language='en',
source_language='auto'):
    url =
f"https://translate.googleapis.com/translate_a/single?client
=gtx&sl={source_language}&tl={target_language}&dt=t&q={text}
"
    response = requests.get(url)
    translation = response.json()[0]
    bahasa = ""
    try:
        for i in translation:
            bahasa += i[0]
    except:
        pass
    return bahasa
```

Fungsi `back_translate` dalam *script* tersebut merupakan implementasi dari back-translation, sebuah teknik augmentasi data yang umum digunakan dalam pemrosesan bahasa alami. Fungsi ini menggunakan layanan terjemahan Google Translate untuk menerjemahkan teks dari bahasa target ke bahasa sumber, dan kemudian menerjemahkannya kembali ke bahasa target aslinya. Proses ini membantu menciptakan variasi dalam data pelatihan dengan memperkenalkan variasi sintaksis dan semantik. Fungsi tersebut mengonfigurasi URL dengan parameter teks, bahasa sumber, dan bahasa target, dan kemudian mengambil respons JSON dari layanan terjemahan. Hasil terjemahan diambil dari respons tersebut dan dikembalikan sebagai teks hasil back-translation. Pemrosesan teks menggunakan *loop* dan *exception* untuk menangani respons JSON yang mungkin bervariasi.

Teknik augmentasi dengan back-translation menjadi salah satu alat penting dalam memperkaya data latih untuk kasus tertentu. Dalam upaya untuk menciptakan model yang tanggap dan akurat, mengatasi masalah kurangnya variasi data awal adalah langkah yang penting. Dengan mengintegrasikan variasi baru dari data yang ada, model dapat memiliki pandangan yang lebih luas dan lebih baik dalam mengenali berbagai bentuk ujaran negatif.

## # Proses Back-translation

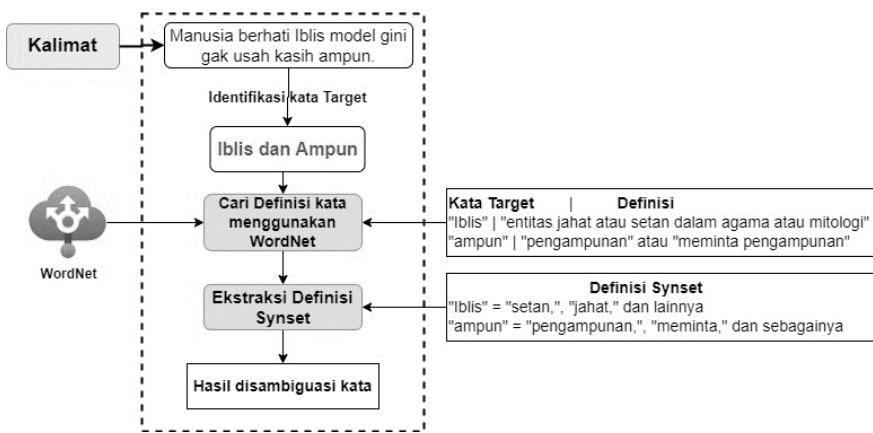
```
for text in X_train[start:stop]:  
    #translated_text_fr = back_translate(text,  
target_language='fr')  
    #translated_text_en = back_translate(translated_text_fr,  
target_language='en')  
    translated_text_en = back_translate(text,  
target_language='en')  
    back_translated_text = back_translate(translated_text_en,  
target_language='id')  
    newX.append(back_translated_text)  
    nomor = nomor+1  
    print(f"{nomor}")  
    print("Original text:", text)  
    # print("Translated text:", back_translated_text)  
    print("Back translated text:", back_translated_text)  
    print("=="*20)
```

Potongan *script* tersebut merupakan implementasi proses augmentasi data menggunakan teknik *back-translation* pada dataset pelatihan (*X\_train*). Proses ini melibatkan iterasi melalui setiap teks pada rentang indeks dari *start* hingga *stop* dalam *X\_train*. Pada setiap iterasi, teks diterjemahkan dari bahasa sumber (dalam hal ini bahasa Inggris) ke bahasa target (dalam hal ini bahasa Indonesia) dan kemudian diterjemahkan kembali ke bahasa sumber. Hasil terjemahan kembali ini kemudian ditambahkan ke dalam list ‘*newX*’ , yang nantinya akan menjadi data yang telah di-augmentasi. Setiap iterasi juga mencetak informasi termasuk nomor iterasi, teks asli, dan teks.

## 6.2 Disambiguasi Kata

Disambiguasi arti kata adalah salah satu masalah tertua dalam pemrosesan bahasa alami. Disambiguasi arti kata adalah tugas untuk secara otomatis mengidentifikasi arti yang tepat dari sebuah kata yang ambigu menurut konteks tertentu di mana kata tersebut muncul (Laatar dkk., 2018). Sebagian besar karya yang berkaitan dengan disambiguasi arti kata diterapkan pada bahasa Inggris. Namun banyak peneliti baru-baru ini diperkenalkan untuk memecahkan disambiguasi makna kata. Misalnya, Bouhriz dan

Benabbou (2016) berfokus pada Wordnet Arab untuk mengekstrak pengertian kata yang ambigu. Tujuannya adalah untuk memanfaatkan tidak hanya konteks lokal tetapi juga konteks global untuk menentukan arti yang benar dari sebuah kata yang ambigu. Untuk alur proses dari diambiguasi kata dapat dilihat pada ilustrasi Gambar 6.2.



**Gambar 6. 2** Proses Disambiguasi Kata

Menentukan target kata dalam proses ini umumnya dilakukan dengan mengidentifikasi kata sifat dan kata benda yang umumnya lebih cenderung ambigu daripada kata kerja. Beberapa penelitian telah melaporkan hasil dalam domain disambiguasi makna kata (*Word Sense Disambiguation* atau WSD) dengan menggunakan metode yang dikenal sebagai algoritma Lesk atau variasi lain yang diperluas dengan leksikon WordNet, seperti yang dijelaskan dalam studi-studi oleh (Ekedahl dan Golub, 2004) dan (Naskar dan Bandyopadhyay, 2007). Secara umum, WSD merupakan proses otomatis yang mengidentifikasi makna tertentu dari kata yang menjadi fokus dalam konteks yang relevan. Ini telah mendapatkan perhatian yang signifikan dalam dekade terakhir, menghasilkan banyak kemajuan dalam peningkatan kualitas hasil

Setelah diperoleh daftar istilah kata pada proses deteksi target kata, maka kita dapat mengeliminasi seluruh kumpulan daftar kata tersebut berdasarkan kata-kata ambigu. Kata ambigu merupakan kata yang bermakna ganda, sehingga hanya kata yang sesuai dengan kalimat sebelumnya yang akan digunakan. Proses disambigu kata sebagai berikut. Mari kita lihat potongan dari *script* berikut.

## # Disambigu Kata Menggunakan Algoritma Lesk

```
def lesk(kata, kalimat):
    synsets = wn.synsets(kata, lang='ind')
    max_overlap = 0
    context = set(kalimat.split())
    for synset in synsets:
        url =
f"https://translate.googleapis.com/translate_a/single?client
=gtx&sl=en&tl=id&dt=t&q={synset.definition()}"
        translation = requests.get(url)
        kalimat = translation.json()[0][0][0]
        # casefolding
        kalimat = casefolding(kalimat)
        signature = set(kalimat.split())
        for example in synset.examples():
            # print(example)
            urlContoh =
f"https://translate.googleapis.com/translate_a/single?client
=gtx&sl=en&tl=id&dt=t&q={example}"
            translationExamples = requests.get(urlContoh)
            contohKata = translationExamples.json()[0][0][0]
            # casefolding
            contohKata = casefolding(contohKata)
            signature.update(set(contohKata.split()))
            overlap = len(signature.intersection(context))
            if overlap > max_overlap:
                max_overlap = overlap
                best_sense = synset
    # Cek max overlap
    sinonim = []
    if max_overlap==0:
        for lemma in synsets[0].lemma_names(lang='ind'):
            sinonim.append(lemma)
    else:
        for lemma in best_sense.lemma_names(lang='ind'):
            sinonim.append(lemma)
    return sinonim
```

Fungsi `lesk` dalam *script* tersebut menerapkan metode Lesk untuk menyelesaikan ambiguitas kata dalam suatu kalimat. Metode Lesk berusaha untuk menentukan makna kata dengan melihat konteks di sekitarnya. Berikut adalah penjelasan singkat dari potongan script tentang bagaimana fungsi ini bekerja:

- Input: Fungsi menerima dua parameter: kata (kata yang ambigu) dan `kalimat` (kalimat di mana kata tersebut digunakan).
- Sinonim: Pertama, fungsi mendapatkan semua `synset` (kelompok makna) dari kata yang ambigu menggunakan `WordNet (wn.synsets(kata, lang='ind'))`.
- Pemrosesan Sinonim: Untuk setiap `synset`, fungsi mendapatkan definisi kata dan contoh penggunaannya. Definisi dan contoh diterjemahkan ke dalam bahasa Indonesia menggunakan layanan terjemahan Google Translate. *Casefolding* (pengubahan huruf ke huruf kecil) diterapkan pada hasil terjemahan.
- Penandaan (*Signature*): Untuk setiap `synset`, fungsi membuat "*signature*" berupa himpunan kata-kata dari definisi dan contoh yang sudah diterjemahkan.
- Perhitungan *Overlap*: Fungsi menghitung jumlah kata yang tumpang tindih (*overlap*) antara *signature synset* dan kata-kata dalam kalimat konteks.
- Pemilihan Makna Terbaik: Fungsi memilih makna kata yang memiliki *overlap* maksimum dengan kalimat konteks. Jika tidak ada tumpang tindih (*overlap = 0*), maka fungsi memilih sinonim dari makna pertama (`synsets[0]`). Jika terdapat tumpang tindih, maka fungsi memilih sinonim dari makna terbaik yang memiliki *overlap* maksimum.

- *Output*: Fungsi mengembalikan daftar sinonim yang terkait dengan makna kata yang dianggap paling sesuai dengan konteks.

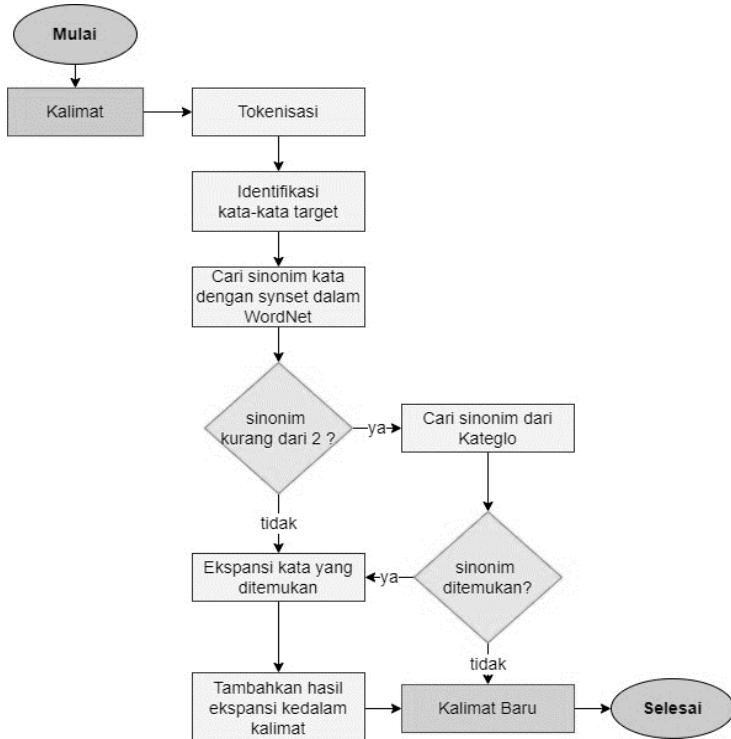
Fungsi ini membantu memecahkan ambiguitas makna kata dalam kalimat dengan memanfaatkan informasi dari definisi dan contoh penggunaan kata dalam WordNet, serta dengan memeriksa sejauh mana kata tersebut cocok dengan konteks kalimat yang diberikan.

### 6.3 Ekspansi Teks

Ekspansi teks adalah strategi yang digunakan dalam analisis teks untuk meningkatkan pemahaman atau cakupan informasi dalam sebuah dokumen atau korpus teks. Tujuannya adalah untuk mendapatkan gambaran yang lebih komprehensif tentang topik atau isu tertentu yang sedang diteliti. Dalam konteks ini, ekspansi teks dapat mencakup beberapa metode, seperti penggunaan sinonim, ekstraksi entitas, atau mencari hubungan antar kata kunci dalam teks (Sharma dkk., 2021).

Salah satu pendekatan yang sering digunakan dalam ekspansi teks adalah dengan mencari sinonim atau kata-kata yang serupa dalam konteks tertentu melalui basis pengetahuan eksternal seperti WordNet maupun Kateglo. Ekspansi teks memainkan peran penting dalam meningkatkan pemahaman kontekstual semantik melalui pendekatan berbasis ekspansi kueri (Muzakir dkk., 2023). Misalnya saja kata “*Kaum Cebol*”, yang dapat menimbulkan ambiguitas bagi penggunanya. Pencarian informasi ini di internet atau referensi lain mengungkapkan bahwa kata tersebut memiliki banyak arti berdasarkan sinonim. Dengan demikian, ekspansi kata menjadi mungkin melalui deteksi kata, sehingga memungkinkannya mewakili konsep yang lebih luas. Proses ini dilakukan dengan cara memodifikasi atau memperkaya kata-kata dalam teks atau kueri

dengan kata-kata lain yang memiliki hubungan semantik atau makna serupa. Tujuan utama dari ekspansi semantik yaitu untuk meningkatkan akurasi dan relevansi hasil dalam pencarian informasi atau analisis teks.



**Gambar 6.3** Proses Ekspansi Teks

Pada Gambar 6.3 menunjukkan alur proses ekspansi yang memanfaatkan basis pengetahuan WordNet dan Kateglo untuk memperkaya informasi kata. Ekspansi semantik merujuk kepada sejumlah teknik yang digunakan untuk mengubah kata-kata dengan tujuan memenuhi kebutuhan informasi tertentu. Terdapat tiga kategori utama dalam teknik ekspansi kata, yaitu *Manual Semantic Expansion (MSE)*, *Automatic Semantic Expansion (ASE)*, dan *Interactive Semantic Expansion (ISE)* (Farhan dkk., 2020). *MSE* merupakan metode yang memungkinkan pengguna untuk secara

manual memodifikasi kueri tanpa bantuan sistem. Di sisi lain, *ASE* adalah pendekatan yang secara otomatis memodifikasi kueri tanpa campur tangan pengguna, seperti sistem yang memasukkan istilah thesaurus ke dalam kueri. Salah satu contoh implementasi *ASE* untuk bahasa Indonesia adalah *website* kateglo.com, yang digunakan dalam penelitian ini untuk melakukan proses ekspansi semantik. Selain itu juga, dalam penelitian ini kami menggunakan WordNet sebagai basis data leksikal dan tesaurus.

WordNet adalah sebuah basis data leksikal dan tesaurus yang populer untuk mengorganisir kata-kata dalam bahasa Inggris berdasarkan hubungan semantik. Ini adalah sumber daya leksikal yang berisi kumpulan kata-kata dan hubungan antara kata-kata tersebut, serta definisi dan contoh penggunaan. WordNet digunakan secara luas dalam pemrosesan bahasa alami, analisis teks, pemahaman bahasa manusia, dan aplikasi lain yang melibatkan pemahaman dan analisis bahasa. Beberapa fitur penting dari WordNet meliputi sinonim, hubungan semantik, definisi dan contoh, serta pengelompokan kata-kata.

Proses ekspansi semantik berguna dalam meningkatkan efektivitas analisis teks, pengambilan informasi, dan pencarian informasi dengan memungkinkan pemahaman yang lebih dalam terhadap konten teks yang sedang dianalisis atau dicari. Dengan memperluas cakupan semantik, proses ini dapat membantu mengidentifikasi informasi yang mungkin terlewatkan atau memperbaiki relevansi hasil pencarian.

Terdapat beberapa tahapan dalam proses ekspansi semantik yang bertujuan untuk meningkatkan akurasi dan relevansi hasil analisis atau pencarian dengan memasukkan sinonim, kata-kata serupa, atau konsep yang terkait dalam teks atau *query*. Berikut adalah beberapa tahapan umum dalam ekspansi teks:

1. Identifikasi Kata Kunci: Tahap pertama dalam ekspansi semantik adalah mengidentifikasi kata-kata kunci atau *query*

yang akan diperluas secara semantik. Ini bisa berupa kata-kata dalam *query* pencarian atau kata-kata kunci dalam dokumen yang sedang dianalisis.

2. Pemahaman Konteks: Memahami konteks dari kata-kata kunci atau *query* sangat penting. Ini melibatkan penentuan bagaimana kata-kata tersebut digunakan dalam kalimat atau dokumen dan apa yang mereka maksudkan.
3. Pencarian Sinonim: Salah satu metode yang umum digunakan dalam ekspansi semantik adalah mencari sinonim atau kata-kata serupa untuk kata-kata kunci atau *query*. Ini dapat dilakukan menggunakan kamus sinonim atau teknik pemrosesan bahasa alami yang menggunakan *embedding* kata untuk menemukan kata-kata yang memiliki makna serupa.
4. Ekspansi Entitas: Dalam beberapa kasus, ekspansi entitas seperti nama orang, tempat, atau konsep penting lainnya dapat digunakan untuk memperkaya konteks teks. Ini membantu dalam mengidentifikasi konsep-konsep terkait yang mungkin relevan dengan *query* atau dokumen.
5. Penggunaan Algoritma Semantik: Penggunaan algoritma seperti word2vec atau fastText dapat membantu dalam menemukan kata-kata yang berkaitan secara semantik dengan kata-kata kunci. Algoritma semantik ini dapat memahami hubungan antara kata-kata dalam teks dan menyarankan kata-kata yang relevan.
6. Pemilihan Kata-kata Baru: Setelah kata-kata tambahan atau kata-kata baru ditemukan, langkah selanjutnya adalah memilih kata-kata mana yang akan dimasukkan ke dalam teks atau *query* awal. Pemilihan ini didasarkan pada relevansi dan signifikansi kata-kata tambahan tersebut terhadap konteks.

7. Perubahan Teks atau *Query*: Tahap terakhir adalah mengganti kata-kata kunci atau memodifikasi *query* awal dengan kata-kata tambahan atau hasil dari ekspansi semantik. Ini akan menghasilkan *query* yang lebih luas dan cakupan informasi yang lebih besar.

## Contoh implementasi ekspansi semantik

Pada tahap awal, kita akan mendefinisikan target kata yang akan dikita perluas menggunakan WordNet yang merupakan kamus bahasa yang support multi-bahasa. Setiap target kata dari suatu kalimat akan dideteksi sebagai kata sifa (adj), kata kerja (verb), dan kata benda (noun) dalam kumpulan data kemudian diperluas melalui penambahan kata-kata yang memiliki makna serupa atau sinonim. Jika daftar target kata tidak diketemukan pada WordNet, maka akan dicek pada Kateglo. Untuk menjalankan proses ekspansi teks, kita memerlukan beberapa *package library* diantaranya: `wordnet`, `omn-1.4`, `stopwords`, dan `punk` yang terdapat pada *library nltk* dengan cara menginstall kedalam project kita menggunakan perintah `nltk.download('wordnet')`.

### # Import library

```
1 import pandas as pd
2 import json
3 import requests
4 from spacy.lang.id import Indonesian
5 from Sastrawi.StemmerFactory import StemmerFactory
6 from nltk.corpus import wordnet as wn
```

Dari potongan skrip diatas, *library* yang terpenting untuk kita gunakan sebagai bagian ekspansi teks yaitu pada baris ke-3, baris ke-4, dan baris ke-6.

### # Pendefinisian Kata di WordNet

```
1 def definisiKata(kata):
2     synsets = wn.synsets(kata, lang='ind')
3     if len(synsets) == 0:
4         try:
```

```

5     url =
f"http://kateglo.lostfocus.org/api.php?format=json&phrase={k
ata}"
6     response = requests.get(url)
7     data = response.text
8     parsed = json.loads(data)
9     definisi =
parsed["kateglo"]["definition"][0]['def_text']
10    return definisi
11 except:
12     return kata
13 else:
14     url =
f"https://translate.googleapis.com/translate_a/single?client
=gtx&sl=en&tl=id&dt=t&q={synsets[0].definition() }"
15     translation = requests.get(url)
16     kalimat = translation.json()[0][0][0]
# casefolding
17     kalimat = casefolding(kalimat)
18     return kalimat

```

Pada *script* diatas, kita akan menentukan jenis *synsets* WordNet yang akan digunakan dalam bahasa Indonesia di baris ke-2. Selanjutnya, kita juga akan menggunakan layanan dari Kateglo untuk mengakses seluruh data kata menggunakan perintah pada baris ke-5. Pada baris ke-5 ini, variabel kata akan di sesuaikan berdasarkan input dari kata yang akan dicari sinonimnya. Proses pengambilan kata di Kateglo didasarkan pada sinonim kata yang akan dicari berdasarkan sebagai kata sifat (*adj*), kata kerja (*verb*), dan kata benda (*noun*).

## # Pendefinisian Pengambilan Kata dari Kateglo

```

1 def kateglo_role(kata):
2     url =
f"http://kateglo.lostfocus.org/api.php?format=json&phrase={k
ata}"
3     response = requests.get(url)
4     data = response.text
5     parsed = json.loads(data)
6     role = parsed["kateglo"]["lex_class"]
7     return role
8 def sinonimKateglo(kata):
9     url =
f"http://kateglo.lostfocus.org/api.php?format=json&phrase={k
ata}"

```

```

10     response = requests.get(url)
11     data = response.text
12     parsed = json.loads(data)
13     post = parsed["kateglo"]["relation"]["s"]
14     daftarKata = []
15     for i in range(len(post)-1):
16         daftarKata.append(post[f"{i}"]["related_phrase"])
17     return daftarKata

```

Informasi kata berdasarkan *input* yang akan dicari kemudian akan di tampung dalam variabel `response` pada baris ke-10. Selanjutnya, daftar kata yang akan di cek dan diambil berdasarkan aturan dari Kateglo dengan menentukan *type relation* yaitu *s* (*sinonim*). Kemudian hasil tersebut akan di gabungkan kedalam daftar kandidat kata untuk semua *type related\_phrase*. Setelah itu, proses ekspansi akan berlangsung dengan mengambil seluruh daftar istilah kata target yang diperoleh berdasarkan fungsi yang sudah dibangun sebelumnya. Untuk lebih lengkap, silahkan pahami script ekspansi semantik berikut ini.

## # Ekspansi Semantik

```

def semantic_expantion(text):
    teksAsli = copy.copy(text)
    text = text.split()
    teksAnt=[]
    # Antonim
    for x in text:
        teksAnt.append(x)
    for y in range(0, len(teksAnt)):
        if(teksAnt[y]=="tidak"):
            try:
                # Get Kateglo
                url =
                "http://kateglo.lostfocus.org/api.php?format=json&phrase=" + teksAnt[y+1]
                response = requests.get(url)
                data = response.text
                parsed = json.loads(data)
                # Cek Kalimat
                if(parsed['kateglo']['lex_class']=="adj"):
                    post = parsed["kateglo"]["relation"]["a"]
                    lenPost=len(post)
                    for x in range(0,lenPost):
                        st=str(x)
                        value=post[st]
                        if(value['rel_type']=='a' and
                           value['lex_class']=='adj'):

```

```

        ant=value['related_phrase']
        teksAnt[y]=ant
        teksAnt[y+1]=""
        break
    except:
        print("ERROR")
# Sinonim
teksSin = []
textJoin = " ".join(teksAnt)
text = textJoin.split()
for x in text:
    teksSin.append(x)
# Jika Kata Kurang Panjang
if len(teksSin) <= 3:
    teksBaru = []
    for y in range(0, len(teksSin)):
        teksBaru.append(teksSin[y])
        definisi = definisiKata(teksSin[y])
        definisiSplit = definisi.split()
        for x in definisiSplit:
            teksBaru.append(x)
    teksSin = []
    teksSin = copy.copy(teksBaru)
detailDaftarSinonim = {}
listCekKata = []
for y in range(0, len(teksSin)):
    try:
        jenisKata = wn.synsets(teksSin[y], lang="ind")
        kondisiKasar = kataKasar(teksSin[y])
        if jenisKata[0].pos() in roleKata or kondisiKasar:
            try:
                listCekKata.append(y)
                role = kateglo_role(teksSin[y])
                daftarAllSinonimNew = lesk(teksSin[y], textJoin)
                daftarSinonim = []
                daftarAllSinonim = [daftarSinonim for
daftarSinonim in daftarAllSinonimNew if daftarSinonim !=
teksSin[y]]
                # Cek Jenis Sinonim
                daftarKataRelevan = {}
                daftarNilai = {}
                if len(daftarAllSinonim) <= 1:
                    daftarAllSinonim.extend(sinonimKateglo(teksSin[y
]))
                for x in range(0, len(daftarAllSinonim)):
                    kata = daftarAllSinonim[x].replace('_', ' ')
                    kata = casefolding(kata)
                    nilai = cekSimilarity(teksSin[y], kata)
                    daftarNilai[x] = nilai
                    daftarKataRelevan[x] = kata
                # Sorted Nilai

```

```

        sortedNilai = sorted(daftarNilai.items(),
key=lambda x: x[1], reverse=True)
        if len(sortedNilai)<2:
            highestKey = [item[0] for item in
sortedNilai[:1]]
        else:
            highestKey = [item[0] for item in
sortedNilai[:2]]
        for i in highestKey:
            daftarSinonim.append(daftarKataRelevan[i])
        if len(daftarSinonim) < 2:
            detailDaftarSinonim[y] = daftarSinonim[0]
        else:
            detailDaftarSinonim[y] = daftarSinonim[0] + " "
+ daftarSinonim[1]
        except:
            pass
    except:
        pass
    addNew = copy.copy(teksSin)
    for key in reversed(detailDaftarSinonim):
        try:
            addNew.insert(key+1, detailDaftarSinonim[key])
        except:
            addNew.append(detailDaftarSinonim[value])
    text = " ".join(addNew)
    return text

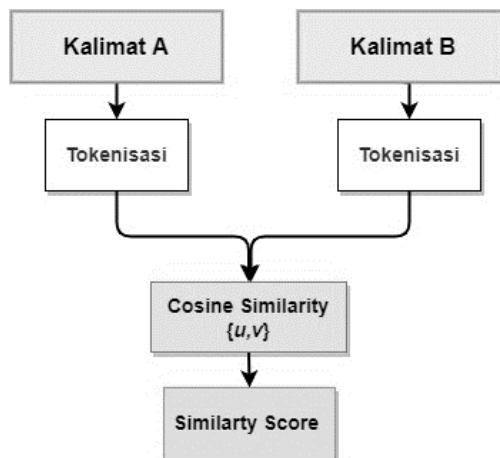
```

Dari *script* diatas, ada tiga aturan yang diberikan untuk ekspansi semantik, yaitu menentukan Antonim, Sinonim, dan pengecekan jenis kata dan kata kasar. Pada aturan Antonim kata, pertama-tama, script ini mencoba mendeteksi kata-kata yang diawali dengan "tidak". Jika mendeteksi "tidak", *script* akan mencoba mendapatkan antonim dari kata berikutnya menggunakan layanan dari Kateglo API. Jika kata tersebut adalah kata sifat (*adj*), *script* akan menggantikan kata "tidak" dengan antonimnya. Kemudian untuk Sinonim kata, Script ini kemudian mencoba mencari sinonim dari kata-kata dalam teks menggunakan berbagai sumber, seperti WordNet (*wn*) dan Kateglo API. Jika panjang teks kurang dari atau sama dengan 3 kata, script akan mencari sinonim dari setiap kata dan menambahkannya ke teks. Jika panjang teks lebih dari 3 kata, *script* akan mencoba menentukan sinonim terbaik dengan memeriksa jenis kata, peran kata (*role*), dan nilai similaritas. Terakhir yaitu

menentukan jenis kata dan kata kasar, Script melakukan pengecekan jenis kata menggunakan WordNet. Jika jenis kata dan kondisi kasar memenuhi syarat, *script* akan melanjutkan untuk mencari sinonim.

#### 6.4 Kemiripan Semantik

Pencarian sinonim dalam fase pencocokan kata melibatkan perhitungan kemiripan semantik kata atau *semantic similarity* dengan memanfaatkan daftar sinonim-sinonim dari WordNet. Proses melibatkan pengambilan semua sinonim yang mungkin dari setiap kata sinonim yang digunakan oleh penerjemah dalam dokumen pengujian, daripada menggunakan kata-kata asli dari dokumen referensi. Hal ini menunjukkan bahwa keduanya harus memiliki jenis kata yang sama dan termasuk dalam kumpulan yang sama dalam WordNet untuk mencapai skor tertinggi (Alaa dkk., 2016). Ilustrasi dari proses ini dapat dilihat pada Gambar 6.4.



**Gambar 6. 4** Proses Semantic Similarity

Kesamaan semantik memiliki tujuan untuk menilai perbandingan antara kata-kata berdasarkan konsep semantik dengan menggunakan metrik kesamaan semantik (Kulmanov dkk., 2021). Salah satu teknik yang sering diterapkan untuk mengukur kesamaan semantik adalah *cosine similarity* (Mahmoud dan Zrigui, 2017),

yang bekerja dengan mengidentifikasi sejauh mana kesamaan antara kalimat 1 (S1) dan kalimat 2 (S2) dengan menghitung jumlah elemen yang serupa di antara keduanya. Vektor kata yang digunakan untuk mengukur kesamaan kosinus dapat dilihat dalam Persamaan (6.1).

$$\text{Cos}(S1, S2) = \frac{S1 \cdot S2}{\|S1\| \cdot \|S2\|} = \frac{\sum_{i=1}^k S1_i S2_i}{\sqrt{\sum_{i=1}^k S1_{i2} \sum_{i=1}^k S2_{i2}}} \quad (6.1)$$

Dalam konteks representasi vektor kalimat dari teks kandidat target S dan teks sumber S1 dalam dimensi k, dilakukan perbandingan antara setiap kalimat dari dua dokumen target dengan semua kalimat dari dokumen sumber. Tujuannya adalah mengidentifikasi hubungan semantik antara kandidat target dan kalimat sumber dengan menggunakan metode *Cosine Similarity*, yang menghitung jumlah kata yang mirip antara kalimat sumber S1 dan kalimat kandidat target S2.

Sebagai ilustrasi, proses pengukuran kemiripan semantik dari kalimat A “*dia memang pintar*” dan kalimat B “*kamu selalu bijak*”. Untuk itu, dapat dilakukan melalui beberapa tahapan berikut.

- a. Persiapan Teks. Tahap persiapan ini menggunakan kalimat A dan B diatas.
- b. Tokenisasi. Tahap ini menggunakan tokenizer dari model IndoBERT untuk mengonversi teks A dan teks B menjadi token. Ini akan menghasilkan daftar token yang siap digunakan dalam model BERT. Token khusus “[CLS]” digunakan untuk menandai awal teks, dan “[SEP]” digunakan untuk menandai akhir teks.
  - 1) Tokenisasi teks A: “[CLS”, “*dia*”, ”*memang*”, ”*pintar*”, ”[SEP]”.
  - 2) Tokenisasi teks B: “[CLS”, “*kamu*”, ”*selalu*”, ”*bijak*”, ”[SEP]”.
- c. Konversi ke model *embedding*. Tahap ini menggunakan model IndoBERT yang terlatih sebelumnya untuk mengonversi

token-token teks A dan teks B menjadi vektor *embedding*. Setiap token akan memiliki representasi vektor dalam ruang semantik. Misalnya, telah dikonversi token-token ini menjadi vektor *embedding* sebagai berikut:

- 1) Vektor *embedding* untuk teks A,  $S_1$ , adalah vektor numerik tiga dimensi, misalnya [0.1, 0.2, 0.3].
- 2) Vektor *embedding* untuk teks B,  $S_2$ , adalah vektor numerik tiga dimensi, misalnya [0.2, 0.1, 0.4].
- d. Perhitungan *Cosine Similarity*. Tahap ini menghitung skor kesamaan kosinus antara vektor *embedding*  $S_A$  dan  $S_B$  menggunakan rumus *cosine similarity*.

- 1) Hitung *dot product*:

$$\begin{aligned} S_1 \cdot S_2 &= (0.1 * 0.2) + (0.2 * 0.1) + (0.3 * 0) \\ &= 0.02 + 0.02 + 0.12 = 0.16 \end{aligned}$$

- 2) Hitung norma Euclidean dari  $S_1$  dan  $S_2$ :

$$\|S_1\| = \sqrt{0.1^2 + 0.2^2 + 0.3^2} = \sqrt{0.01 + 0.04 + 0.09} \approx 0.374$$

$$\|S_2\| = \sqrt{0.2^2 + 0.1^2 + 0.4^2} = \sqrt{0.04 + 0.01 + 0.16} \approx 0.459$$

- 3) Selanjutnya, hitung skor *cosine similarity*:

$$\text{Cosine Similarity } (S_1, S_2) = \frac{0.16}{(0.374 * 0.459)} \approx 0.724$$

- e. Hasil skor kemiripan semantik. Pada tahap ini, skor *cosine similarity* antara kalimat "dia memang pintar" dan "kamu selalu bijak" adalah sekitar 0.724. Skor ini menunjukkan bahwa secara semantik, kedua teks ini memiliki tingkat kemiripan yang cukup tinggi, karena skor kesamaan semantik mendekati 1.

Pada implementasi dalam bahasa pemrograman python, mari kita lihat contoh dalam script berikut ini.

## # Load Model IndoBERT dan Tokenizer

```
model_name = 'indobenchmark/indobert-base-p2'  
tokenizer = AutoTokenizer.from_pretrained(model_name)  
model = AutoModel.from_pretrained(model_name)
```

## # Pembobotan Kemiripan Kata

```
def cekSimilarity(kata1, kata2):
    if kata1 == kata2:
        return 0
    # Tokenize the words and add special tokens
    tokens = ['[CLS]'] + tokenizer.tokenize(kata1) +
    ['[SEP]'] + tokenizer.tokenize(kata2) + ['[SEP]']
    input_ids = tokenizer.convert_tokens_to_ids(tokens)
    input_ids =
tokenizer.build_inputs_with_special_tokens(input_ids)
    # Convert the input to a PyTorch tensor
    inputs = torch.tensor([input_ids])
    # Pass the input through the model to get the embeddings
    with torch.no_grad():
        outputs = model(inputs)
        embeddings = outputs[0][0]
    # Calculate the cosine similarity between the two word
embeddings
    return 1 - cosine(embeddings[1], embeddings[2])
```

Fungsi `cekSimilarity` dalam script tersebut melakukan perhitungan kemiripan semantik antara dua kata menggunakan metode *cosine similarity*. Berikut adalah penjelasan singkat tentang proses tersebut:

- Pengecekan Kemiripan Kata: Fungsi ini memeriksa apakah kedua kata (kata 1 dan kata 2) sama. Jika ya, fungsi mengembalikan nilai 0, karena kata yang sama memiliki cosine similarity 1.
- Tokenisasi dan Penyisipan Token Khusus: Kata-kata tersebut di-tokenisasi dengan menambahkan token khusus seperti '[CLS]' (token awal) dan '[SEP]' (token pemisah) untuk mempersiapkan input model. Hasil token disusun menjadi daftar tokens.
- Konversi Token menjadi ID dan Pembentukan Input Tensor: Token-token dikonversi menjadi ID menggunakan tokenizer. Input tensor dibangun dengan menambahkan token-token khusus. Input tensor diubah menjadi tensor *PyTorch*.

- Penggunaan Model untuk Mendapatkan Embedding: Tensor input diberikan ke model untuk mendapatkan embedding kata-kata. Embedding diambil dari output model.
- Perhitungan Cosine Similarity: Cosine similarity dihitung antara embedding kedua kata. Fungsi mengembalikan nilai 1 dikurangi cosine similarity. Hal ini dilakukan karena semakin besar nilai cosine similarity, semakin besar kemiripan semantik antara kata-kata tersebut.

Jadi, fungsi ini menggunakan model (yang mungkin merupakan model bertipe transformer) untuk mendapatkan embedding kata-kata dan kemudian menghitung cosine similarity antara embedding tersebut untuk menilai kemiripan semantik antara dua kata. Semakin mendekati 1 nilai cosine similarity-nya, semakin mirip secara semantik kedua kata tersebut.

Penggunaan model bahasa bertingkat seperti IndoBERT memberikan dimensi baru dalam mengolah teks dalam sistem deteksi ujaran kebencian. Model ini telah mengalami pelatihan ekstensif dalam berbagai tugas pemrosesan bahasa alami, memainkan peran sentral dalam memahami dan merepresentasikan makna kata-kata dalam kalimat dengan lebih kaya dan kontekstual.

Dalam tahap ini, setiap kata dalam kalimat diproses oleh model IndoBERT untuk menghasilkan vektor representasi yang berisi informasi tentang makna, nuansa, dan konteks kata tersebut dalam kalimat. Representasi ini mencakup karakteristik semantik yang kompleks, seperti relasi antarkata dan interpretasi semantik yang lebih mendalam. Dengan memanfaatkan representasi ini, model dapat lebih baik dalam memahami arti yang sebenarnya dari kata-kata dan bagaimana kata-kata tersebut berinteraksi dalam konteks kalimat.

Vektor representasi kata yang dihasilkan oleh IndoBERT menjadi dasar bagi analisis selanjutnya. Representasi ini diteruskan

ke langkah-langkah berikutnya, seperti pengukuran kemiripan semantik dan pembentukan model embedding. Kemampuan model IndoBERT dalam menangkap nuansa dan konteks kata-kata secara lebih baik memungkinkan sistem deteksi untuk lebih akurat dalam mengenali pola-pola ujaran negatif yang tersembunyi dalam bahasa.

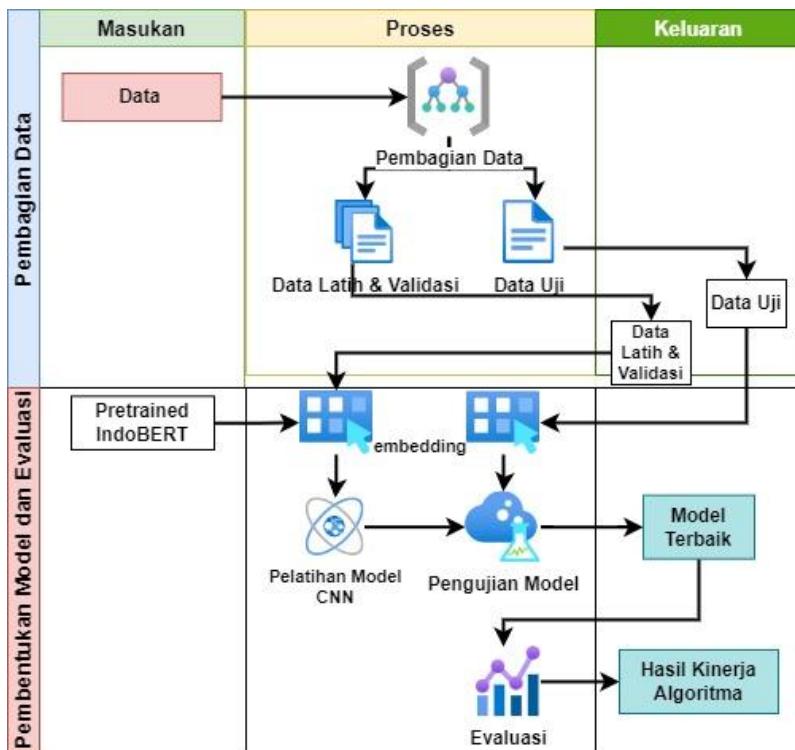
Dalam konteks ujaran kebencian misalnya, makna dan nuansa kata-kata dapat mempengaruhi tingkat keberbahayaan dan intensitas ujaran tersebut. Dengan menerapkan model *pretrained* IndoBERT, sistem pengklasifikasi menjadi lebih mampu dalam membedakan antara ujaran yang bersifat kasual dan yang mengandung kebencian atau ancaman. *Embedding* kata dengan model pretrained menjadi langkah penting dalam meningkatkan akurasi dan ketajaman sistem deteksi, serta memperluas cakupan pemahaman konten dalam bahasa Indonesia.

## **BAB VII**

# **STUDI KASUS: KLASIFIKASI UJARAN KEBENCIAN**

Pada Bab ini merupakan kelanjutan dari Studi Kasus pada Bab 6 sebelumnya. Pada bagian ini, dataset pada proses sebelumnya telah mengalami ekspansi dari teks pendek menjadi teks yang lebih panjang. Oleh sebab itu, pada bagian ini kita dijelaskan proses pembentukan model untuk sistem klasifikasi ujaran kebencian. Pembentukan model menggunakan *embedding* dari IndoBERT. IndoBERT adalah model bahasa berbasis Transformer yang dikembangkan khusus untuk Bahasa Indonesia. Model ini didasarkan pada arsitektur Transformer yang dikenal karena kemampuannya dalam pemrosesan bahasa alami. Namun, IndoBERT telah diadaptasi dan dilatih khusus untuk memahami dan menghasilkan teks dalam bahasa Indonesia.

Model IndoBERT telah melalui tahap pelatihan yang intensif menggunakan sejumlah besar data teks dalam bahasa Indonesia. Hal ini memungkinkan model ini untuk memahami konteks dan makna kata-kata dalam teks Bahasa Indonesia dengan lebih baik. IndoBERT telah menjadi salah satu alat yang penting dalam pemrosesan bahasa alami dan analisis teks berbahasa Indonesia, digunakan dalam berbagai aplikasi seperti analisis sentimen, ekstraksi informasi, terjemahan mesin, dan tugas-tugas pemrosesan teks lainnya. Terdapat dua bagian besar yang menjadi proses klasifikasi ini yaitu pembagian data serta pembentukan model dan evaluasi. Berdasarkan tahapan ini akan menghasilkan model terbaik yang akan digunakan untuk proses pengujian dengan data yang bersumber dari media sosial menggunakan aplikasi sederhana untuk pengujian identifikasi kalimat ujaran kebencian.



**Gambar 7. 1** Alir Proses Klasifikasi Ujaran Kebencian

## 7.1 Data

Sumber data dalam yang digunakan diambil dari penelitian yang sudah ada (Ibrohim & Budi, 2019) dan diperoleh melalui situs GitHub yang dapat di unduh di <https://github.com/okkyibrohim/id-multi-label-hate-speech-and-abusive-language-detection>. Jumlah data sebanyak 13.169 tweet yang terdiri atas 12 label yaitu HS, Abusive, HS\_Individual, HS\_Group, HS\_Religion, HS\_Race, HS\_Physical, HS\_Gender, HS\_Other, HS\_Weak, HS\_Moderate, HS\_Strong.

## 7.2 Data Preprocessing

Pra-pemrosesan data tekstual atau data preprocessing merupakan langkah awal yang diperlukan untuk memproses data teks mentah yang asalnya tidak terstruktur menjadi data yang lebih terstruktur, sehingga data tersebut dapat digunakan secara efektif dalam tahap selanjutnya. Proses pra-pemrosesan data melibatkan beberapa tahapan, seperti *case folding*, pembersihan data, normalisasi, tokenisasi, penghapusan *stop word*, dan *stemming*. Tujuan dari pra-pemrosesan yaitu menghilangkan *noise*, menghasilkan bentuk kata yang seragam, dan mengurangi volume kata (Affandes, 2015). Untuk menerapkan data preprocessing dapat menggunakan skrip berikut.

### # Case Folding

```
import re
# Contoh kalimat
sentence = "Ini Contoh Dalam DUnIa NYAta."
# Memisahkan kata-kata
words = sentence.split()
# Melakukan case folding pada setiap kata
folded_words = [word.lower() for word in words]
# Menggabungkan kembali kata-kata setelah case folding
folded_sentence = ' '.join(folded_words)
print("Kalimat Awal:", sentence)
print("Hasil Case Folding:", folded_sentence)
Output: Kalimat Awal: Ini Contoh Dalam DUnIa NYAta.
        Hasil Case Folding: ini contoh dalam dunia nyata.
```

Dalam contoh ini, kalimat "Ini Contoh Dalam DUnIa NYAta." diubah menjadi "ini contoh dalam dunia nyata." setelah proses case folding.

### # Normalization

```
from nltk.stem import WordNetLemmatizer
# Contoh kata-kata
words = ["running", "flies", "better", "dogs", "cats"]
# Inisialisasi objek WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
# Lemmatisasi kata-kata
normalized_words = [lemmatizer.lemmatize(word) for word in words]
```

```

print("Original Words:")
print(words)
print("\nNormalized Words:")
print(normalized_words)
Output: Kalimat Awal:['running', 'flies', 'better',
                     'dogs', 'cats']
Setelah Normalized:['running', 'fly', 'better',
                     'dog','cat']

```

Dalam contoh ini, WordNetLemmatizer digunakan untuk melakukan lemmatization pada kata-kata. Proses ini akan mengembalikan kata-kata dalam bentuk dasar mereka. Misalnya, "running" akan diubah menjadi "run", "flies" menjadi "fly", "better" tetap "better" (karena "better" sudah dalam bentuk dasar), "dogs" tetap "dog", dan "cats" tetap "cat".

## # Tokenization

```

# Contoh teks
text = "Ini adalah contoh kalimat sederhana."
# Tokenisasi menggunakan split
tokens = text.split()
# Menampilkan hasil
print("Original Text:", text)
print("Tokenized Text:", tokens)
output: Kalimat Awal: Ini adalah contoh kalimat sederhana.
Setelah Tokenized: ['Ini', 'adalah', 'contoh',
                     'kalimat', 'sederhana.']

```

Dalam contoh ini, metode `split` digunakan untuk memisahkan kata-kata dalam teks berdasarkan spasi. Setiap kata menjadi elemen dalam list `tokens`. Metode ini sederhana dan berguna jika pemisahan kata didasarkan pada spasi atau karakter pemisah tertentu dalam teks.

## # Stop Word Removal

```

from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
# Contoh teks
text = "Ini adalah contoh kalimat dengan beberapa kata
pengisi (stop-words) yang perlu dihilangkan."
# Mendownload daftar stop-word dari NLTK
stop_words = set(stopwords.words('indonesian'))
# Tokenisasi kata-kata dalam teks

```

```

words = word_tokenize(text)
# Menghapus stop-word
filtered_words = [word for word in words if word.lower() not
in stop_words]
# Menampilkan hasil
print("Original Text:", text)
print("Text After Stop-word Removal:", '
''.join(filtered_words))
Output: Kalimat Awal: Ini adalah contoh kalimat dengan
beberapa kata yang perlu dihilangkan
Setelah Stop-word Removal: contoh kalimat dihilangkan

```

Dalam contoh ini, kita menggunakan daftar *stop-word* Bahasa Indonesia dari NLTK (stopwords.words('indonesian')). Kemudian, kita melakukan tokenisasi kata-kata dalam teks menggunakan `word_tokenize` dari NLTK, dan menghasilkan list kata-kata. Setelah itu, kita membuat list baru (`filtered_words`) yang hanya berisi kata-kata yang bukan stop-word.

## # Stemming

```

from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
# Contoh teks
text = "Berlari, berlari, dan terus berlari adalah aktivitas
yang sangat menyenangkan bagi para pelari."
# Inisialisasi objek PorterStemmer
porter = PorterStemmer()
# Tokenisasi kata-kata dalam teks
words = word_tokenize(text)
# Melakukan stemming pada setiap kata
stemmed_words = [porter.stem(word) for word in words]
print("Original Text:", text)
print("Text After Stemming:", ''.join(stemmed_words))

```

Output: Kalimat Awal: Berlari, berlari, dan terus berlari  
adalah aktivitas yang sangat menyenangkan bagi  
para pelari.  
Setelah Stemming: berlari , berlari , dan teru  
berlari adalah aktivita yang sangat  
menyenangkan bagi para pelari.

Dalam contoh ini, kita menggunakan `PorterStemmer` dari NLTK untuk melakukan stemming pada kata-kata dalam teks. Setiap

kata dalam teks diubah menjadi bentuk dasarnya (atau kata dasar). *Stemming* membantu dalam mengurangi kata-kata ke bentuk dasarnya sehingga kata-kata dengan akar kata yang sama dapat diperlakukan dengan seragam.

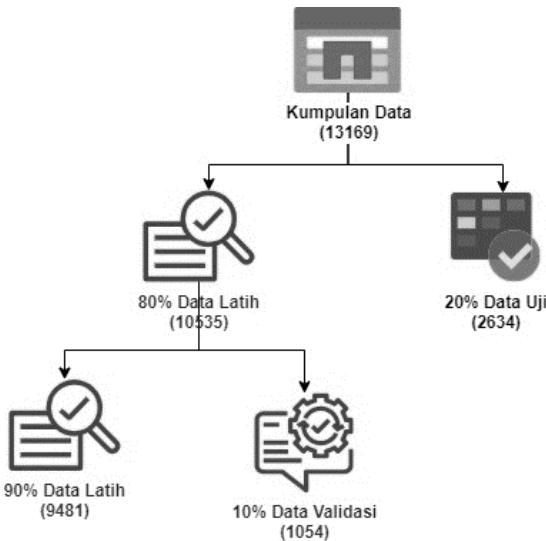
### 7.3 Pembagian Data

Pembagian data dilakukan ketika proses persiapkan data yang akan kita gunakan telah melalui pre-processing. Dalam kasus ini, format data yang digunakan dalam format Comma Separated Values (.csv) seperti pada Gambar 7.2.

|   | A                                  | B          | C             | D        | E           | F       | G           | H         | I        | J       | K           | L         | M |
|---|------------------------------------|------------|---------------|----------|-------------|---------|-------------|-----------|----------|---------|-------------|-----------|---|
| 1 | Tweet                              | HS_Abusive | HS_Individual | HS_Group | HS_Religion | HS_Race | HS_Physical | HS_Gender | HS_Other | HS_Weak | HS_Moderate | HS_Strong |   |
| 2 | - disaat semua cowok berusaha m    | 1          | 1             | 1        | 0           | 0       | 0           | 0         | 0        | 1       | 1           | 0         | 0 |
| 3 | RT USER: USER siapa yang telat ng  | 0          | 1             | 0        | 0           | 0       | 0           | 0         | 0        | 0       | 0           | 0         | 0 |
| 4 | 41. Kadang aku berfikir, kenapa ak | 0          | 0             | 0        | 0           | 0       | 0           | 0         | 0        | 0       | 0           | 0         | 0 |
| 5 | USER USER Kaum cebong kapir ud     | 1          | 1             | 0        | 1           | 1       | 0           | 0         | 0        | 0       | 0           | 1         | 0 |
| 6 | USER Ya bani taplak dkk \xf0\x9f\x | 1          | 1             | 0        | 1           | 0       | 0           | 0         | 0        | 1       | 0           | 1         | 0 |

**Gambar 7. 2** Data dalam Format CSV

Pembagian data dalam proses pembentukan model klasifikasi ujaran kebencian dibagi menjadi tiga yaitu data latih, data validasi, dan data uji. Proses pembagian ini bertujuan untuk menghasilkan data latih dan data validasi yang akan digunakan dalam pelatihan model, serta data uji yang akan digunakan untuk menguji kinerja model. Umumnya, pembagian data untuk membangun model klasifikasi dibagi kedalam: data latih dan data uji. 80% untuk data latih dan 20% untuk data uji. Sedangkan untuk validasi pelatihan biasanya menggunakan 10% dari data latih. Skema pembagian data ditunjukkan pada Gambar 7.3.



**Gambar 7. 3** Pembagian Data 80% Data Latih dan 20% Data Uji

Berikut contoh dari potongan *script* untuk pembagian data menggunakan bahasa python.

### #Pembagian Data

```

# penentuan X dan y
X = data[['Tweet']]
y = data.drop(['Tweet'],axis = 1).values
#proses splitting
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

```

## 7.4 Pembentukan Model Deteksi Ujaran Kebencian

Pada tahap ini, pembentukan model dilakukan dengan menggunakan *pre-trained* IndoBERT yang melibatkan beberapa langkah penting. IndoBERT adalah model bahasa berbasis Transformer yang telah dilatih sebelumnya pada data bahasa Indonesia.

#### 7.4.1 Pemahaman tentang *Pre-trained Model*

Pertama-tama, kita perlu memahami apa itu IndoBERT serta variasi dari turunannya dan bagaimana itu dilatih sebelumnya. Pastikan kita memiliki akses ke model dan berkas konfigurasi yang sesuai. Misalnya, dengan menggunakan model *pre-trained* IndoBERT, maka kita bisa mulai dengan mengakses *pre-trained* yang sudah ada di halaman website <https://huggingface.co/indolem>. Disini kita dapat menemukan model-model yang bisa kita gunakan seperti IndoBERT dan IndoBERT-Tweet.



Gambar 7. 4 Pilihan Model *Pre-trained* IndoBERT

Untuk penerapan dalam bentuk pengkodean, dapat dilihat pada contoh *script* pembuatan model *embedding* IndoBERT pada pembahasan selanjutnya.

#### 7.4.2 Memuat Model *Embedding* BERT dan Konfigurasi

Kita perlu mengurai teks menjadi token-token yang sesuai dengan token-token yang digunakan oleh IndoBERT. Ini melibatkan tokenisasi teks dan pengkodean token untuk membentuk input yang dapat diberikan ke model. Selanjutnya, kita dapat memuat *pre-trained* IndoBERT beserta berkas konfigurasinya. Kita dapat menggunakan library seperti *Hugging Face Transformers* untuk memuat model ini. Misalnya, untuk IndoBERT dapat menggunakan konfigurasi model dengan *indobenchmark/indobert-base-p1* atau *indobenchmark/indobert-large-p2* seperti pada potongan Script berikut.

## #Load Model IndoBERT

```
model_class_indobert, tokenizer_class_indobert,  
pretrained_weights_indobert = (tfm.BertModel,  
tfm.BertTokenizer, 'indobenchmark/indobert-base-p2')  
tokenizerIndobert =  
tokenizer_class_indobert.from_pretrained(pretrained_weights_  
indobert)  
modelIndobert =  
model_class_indobert.from_pretrained(pretrained_weights_indo  
bert)
```

Output :

```
Downloading (...)solve/main/vocab.txt: 100%  
229k/229k [00:00<00:00, 1.81MB/s]  
Downloading (...)cial_tokens_map.json: 100%  
112/112 [00:00<00:00, 6.69kB/s]  
Downloading (...)okenizer_config.json: 100%  
2.00/2.00 [00:00<00:00, 119B/s]  
Downloading (...)lve/main/config.json: 100%  
1.53k/1.53k [00:00<00:00, 101kB/s]  
Downloading pytorch_model.bin: 100%  
498M/498M [00:05<00:00, 48.1MB/s]
```

Dari contoh potongan *script* diatas, inisialisasi model dan tokenizer dilakukan menggunakan *library* *Transformers* dari *Hugging Face*, khususnya untuk model IndoBERT (Indonesian BERT). Berikut adalah penjelasan singkat:

- Inisialisasi Variabel: `model_class_indobert, tokenizer_class_indobert,` dan `pretrained_weights_indobert` adalah variabel yang menentukan kelas model, kelas tokenizer, dan bobot model yang akan digunakan. Dalam contoh ini, kelas model dan tokenizer dipilih dari pustaka *Transformers*, dan bobot model adalah '`indobenchmark/indobert-base-p2`'.
- Inisialisasi Tokenizer dan Model: `tokenizerIndobert` dan `modelIndobert` adalah objek tokenizer dan model yang diinisialisasi dengan menggunakan kelas-kelas dan bobot yang telah ditentukan sebelumnya. Objek ini akan digunakan untuk tokenisasi teks dan pemrosesan model dalam tugas-tugas NLP yang melibatkan bahasa Indonesia.

- Output: Setelah inisialisasi, *script* menunjukkan proses pengunduhan beberapa file yang diperlukan untuk tokenizer dan model BERT. File yang diunduh termasuk `vocab.txt`, `special_tokens_map.json`, `tokenizer_config.json`, `config.json`, dan `pytorch_model.bin` yang merupakan bobot model itu sendiri.

### 7.4.3 *Fine-tuning* (Opsiional)

Jika kita memiliki data tugas khusus yang cukup, kita dapat melanjutkan dengan *fine-tuning* model IndoBERT pada data tersebut. Cara ini dapat meningkatkan kinerja model untuk tugas tertentu. Misalnya, kita dapat melakukan pengaturan parameter berdasarkan *Optimizer*, *Batch\_Size*, *Epoch*, atau *Learning rate*. Berikut pada Gambar 7.5 contoh konfigurasi parameter untuk Fine-Tunning.

```
param_grid = {
    'optimizer': [adam, rmsprop, sgd],
    'epochs': [20, 30, 35],
    'batch_size': [8,16,32],
    'lr': [0.001, 0.0001, 2e-05, 3e-05, 5e-05]
}
```

**Gambar 7.5** Konfigurasi parameter untuk Fine-Tunning

### 7.4.4 Pelatihan Model

Jika kita sudah melakukan *fine-tuning* dan mendapatkan hasil pengaturan parameter yang optimal, maka kita perlu melatih model dengan data yang disiapkan sebelumnya. Setelah pelatihan, kita harus menguji model pada data validasi atau pengujian untuk mengukur kinerjanya. Berikut contoh *script* pendefinisian dan trainin model.

## # Define Model

```
model = Sequential()
# Configuring the parameters
model.add(InputLayer((100,768)))
model.add(Conv1D(128,5, activation="relu"))
model.add(GlobalMaxPooling1D())
model.add(Dense(64, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(13, activation="sigmoid"))
model.summary()
```

Pada potongan *script* diatas membentuk sebuah model *neural network* pada model CNN dengan arsitektur yang terdiri dari lapisan-lapisan konvolusi, *max pooling*, *dense*, dan *dropout* untuk tugas tertentu yang mungkin melibatkan data sekuensial. Atau kita dapat menambah jumlah lapisan lainnya untuk mengoptimalkan pelatihan model.

*Script* diatas merupakan contoh pembuatan dan konfigurasi model *neural network* menggunakan *library* bernama ‘Keras’. Berikut adalah penjelasan singkat:

- Inisialisasi Model: `model = Sequential()` adalah langkah pertama dalam membuat model neural network menggunakan Keras. Sequential adalah tipe model Keras yang memungkinkan kita membuat model secara sekuensial, satu lapisan (layer) setelah yang lain.
- Penambahan Lapisan Input:  
`model.add(InputLayer((100, 768)))` menambahkan lapisan input dengan dimensi (shape) (100, 768). Ini menunjukkan bahwa model ini diharapkan menerima input dengan panjang 100 dan dimensi 768.
- Penambahan Lapisan Konvolusi: `model.add(Conv1D(128, 5, activation="relu"))`. Misalnya, kita menggunakan lapisan konvolusi 1D dengan 128 filter, panjang kernel 5, dan fungsi aktivasi ReLU. Lapisan ini berguna untuk mengekstraksi fitur dari data sekuensial.

- Penambahan Lapisan GlobalMaxPooling1D: `model.add(GlobalMaxPooling1D())` menambahkan lapisan global max pooling 1D, yang mengambil nilai maksimum dari setiap fitur sepanjang dimensi sekuensial.
- Penambahan Lapisan Dense: `model.add(Dense(64, activation="relu"))` menambahkan lapisan dense (fully connected) dengan 64 unit dan fungsi aktivasi ReLU.
- Penambahan Dropout: `model.add(Dropout(0.5))`. Misalnya, kita menggunakan lapisan dengan tingkat *dropout* 0.5. *Dropout* digunakan untuk mencegah overfitting dengan secara acak "menghapus" sebagian unit selama pelatihan. Umumnya, semakin tinggi nilai *dropout* dalam suatu model *neural network*, semakin besar kemungkinan beberapa unit atau neuron di lapisan tersebut dinonaktifkan atau "drop out" selama proses pelatihan. *Dropout* adalah suatu metode regularisasi yang membantu mencegah overfitting dalam model. *Overfitting* terjadi ketika model terlalu "menghafal" data pelatihan dan tidak dapat menggeneralisasi dengan baik untuk data baru.
- Penambahan Lapisan Dense Output: `model.add(Dense(12, activation="sigmoid"))`. Misalnya, dengan menambahkan lapisan *dense output* dengan 12 unit dan fungsi aktivasi sigmoid. Jumlah unit dapat disesuaikan tergantung jumlah kelas atau output yang diinginkan.
- Ringkasan Model: `model.summary()` mencetak ringkasan arsitektur model, termasuk detail jumlah parameter pada setiap lapisan.

## # Pelatihan Model

```
from keras.optimizers import Adam
from tensorflow import keras
adam = keras.optimizers.Adam(learning_rate=0.001)
#compile model
model.compile(optimizer=adam, loss='binary_crossentropy',metrics=['accuracy'])
# proses melakukan training
lr_adjust=
tf.keras.callbacks.ReduceLROnPlateau(monitor="val_loss",
factor=0.5, patience=2, verbose=1, mode="auto")
history = model.fit(train_features, y_train,
validation_data=(validation_features,
y_val),
epochs=15, batch_size=32,
callbacks=[lr_adjust], verbose=1)
```

Script ini adalah contoh konfigurasi dan pelatihan model menggunakan pustaka Keras. Berikut adalah penjelasan singkat:

- Inisialisasi Optimizer: Optimizer yang paling populer digunakan adalah Adam. *Adaptive Moment Estimation* (Adam) adalah salah satu algoritma optimisasi yang umum digunakan dalam pelatihan *neural network*. Algoritma ini dikembangkan untuk menggabungkan keuntungan dari algoritma *stokastik gradient descent* (SGD) dan algoritma momentum. Adam merupakan optimizer yang diinisialisasi dengan *learning rate*. *Learning rate* (tingkat pembelajaran) adalah parameter yang mengontrol seberapa besar perubahan yang akan diterapkan pada bobot jaringan selama proses pelatihan. Ini adalah salah satu *hyperparameter* kritis dalam algoritma pembelajaran mesin, terutama dalam konteks pelatihan jaringan saraf tiruan. Parameter dari learning rate dapat disesuaikan sesuai dengan rekomendasi. Rentangnya dimulai dari 0.1 sampai 1.0.
- Compile Model: `model.compile(optimizer=adam, loss='binary_crossentropy', metrics=['accuracy'])` mengonfigurasi model untuk pelatihan. Optimizer Adam dengan learning rate 0.001 dipilih,

fungsi loss 'binary\_crossentropy' digunakan (cocok untuk tugas klasifikasi biner), dan metrik akurasi diukur selama pelatihan.

- Proses Pelatihan : `tf.keras.callbacks.ReduceLROnPlateau` adalah callback yang akan mengurangi learning rate jika tidak terjadi perbaikan pada metrik validasi (`val_loss`) dalam beberapa epoch. Ini membantu mengoptimalkan proses pelatihan. `model.fit` melakukan pelatihan model. Data pelatihan (`train_features` dan `y_train`) dan data validasi (`validation_features` dan `y_val`) digunakan selama 15 epoch dengan ukuran batch sebesar 32. Callback `lr_adjust` digunakan untuk penyesuaian *learning rate*.

Pada *script* ini menunjukkan bagaimana mengonfigurasi model dengan optimizer Adam, fungsi loss binary crossentropy, dan metrik akurasi. Misalnya, model tersebut dilatih menggunakan data pelatihan dan validasi selama 15 *epoch* dengan *batch size* 32, sambil memonitor perubahan dalam `val_loss` untuk mengatur learning rate menggunakan *callback* `ReduceLROnPlateau`.

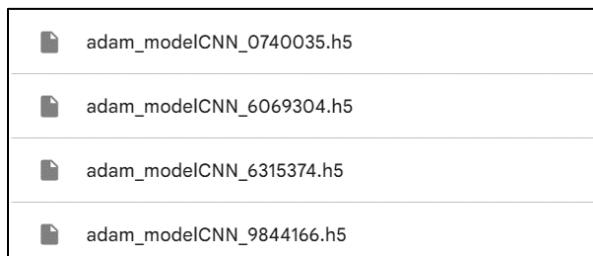
#### 7.4.5 Penyimpanan Model

Setelah kita mendapatkan hasil yang optimal dari proses pelatihan model sebelumnya, selanjutnya adalah menyimpan model tersebut agar dapat digunakan kembali di masa depan. Pada umumnya model akan disimpan dalam format .h5 dan akan digunakan pada saat pengujian sistem.

#### #Penyimpanan Model

```
model.save(f"/content/drive/MyDrive/eksperimen/model/semantic/BERT/80/indobert/CNN/CNN_model_semantic.h5")
```

Berdasarkan script tersebut, maka model akan disimpan dalam format .h5 dan kemudian dapat digunakan sebagai prediksi di proses selanjutnya. Berikut pada Gambar 7.6 hasil dari model yang terbentuk.



**Gambar 7. 6** Contoh Hasil Model Dari Proses Pelatihan

#### 7.4.6 Evaluasi Model

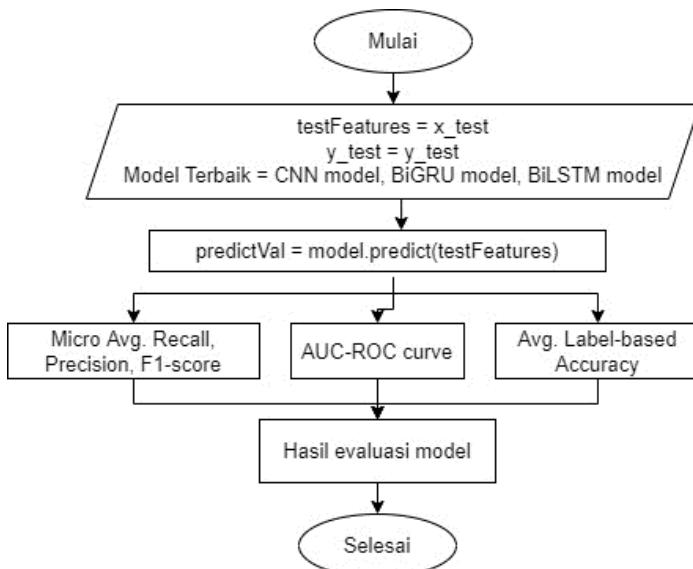
Tahap ini merupakan langkah penting dalam penelitian, yang berfokus pada klasifikasi data uji menggunakan model yang telah dihasilkan dari proses pelatihan sebelumnya. Selama tahap pengujian, model akan memberikan nilai klasifikasi untuk setiap data uji yang tersedia. Perlu dicatat bahwa karakteristik dari kumpulan data yang digunakan dalam penelitian ini menghasilkan lima jenis output yang berbeda, termasuk penentuan apakah sebuah kalimat merupakan ujaran kebencian atau tidak, apakah mengandung kata-kata kasar atau tidak, identifikasi target dari ujaran kebencian tersebut, penentuan kategori ujaran kebencian, dan penilaian tingkatan ujaran kebencian dalam kalimat tersebut.

Dalam kasus ini, setiap model akan dibandingkan secara adil melalui pengujian dan evaluasi. Contohnya, proses klasifikasi dilakukan dengan menggunakan model CNN, BiGRU, dan BiLSTM serta *embedding* dari IndoBERT. Hal ini dilakukan untuk mendapatkan perbandingan yang akurat dengan menggunakan

model terbaik dari hasil *fine-tuning* parameter pada saat proses pelatihan.

Selanjutnya, nilai-nilai klasifikasi yang dihasilkan dari proses pengujian akan menjadi fokus utama dalam tahap evaluasi. Evaluasi ini akan menggunakan metrik-metrik seperti *precision*, *recall*, *f1-score*, dan akurasi, maupun kurva AUC. Metrik-metrik ini memungkinkan kita untuk mengukur sejauh mana model mampu mengidentifikasi dan mengklasifikasikan berbagai aspek terkait domain atau studik kasus dalam data uji. Dengan demikian, tahap evaluasi ini tidak hanya membantu dalam mengevaluasi kualitas dan performa model, tetapi juga memberikan wawasan yang mendalam tentang kemampuannya dalam memahami kompleksitas dan keragaman, misalnya pola kalimat dalam ujaran kebencian.

Model terbaik dari proses pelatihan kemudian digunakan sebagai masukkan pada proses pengujian. Misalnya, berdasarkan analisis data, sebanyak 2625 data digunakan pada proses pengujian. Proses pengujian menggunakan ilustrasi seperti pada Gambar 7.7 yang umumnya diinisialisasi sebagai  $X_{test}$  dan  $y_{test}$ . Kemudian hasil prediksi disimpan pada variabel  $yPredTest$  sebagai variabel hasil prediksi. Setiap label dari data asli akan dibandingkan dengan label hasil prediksi untuk dilakukan evaluasi menggunakan confusion matrix.



**Gambar 7. 7** Alir Pengujian dan Evaluasi Model

Untuk itu, evaluasi dilakukan untuk mengukur kinerja model berdasarkan data pengujian, sebagai contoh pada Tabel 7.1.

**Tabel 7. 1** Contoh Studi Kasus Confusion Matrix

|              |         | Kelas Prediksi |            |
|--------------|---------|----------------|------------|
|              |         | Negatif        | Positif    |
| Kelas Aktual | Negatif | TN (887)       | FP (231)   |
|              | Positif | FN (156)       | TP ( 1351) |

Metode evaluasi *Micro-Averaged F1-measure* digunakan sebagai metode evaluasi dengan menghitung nilai *Precision* dan *Recall* sehingga menghasilkan nilai *F1-Score* yang merupakan performa model. Sebagai contoh dari Tabel 7.3, pada label *Non Hate Speech* (Non HS) misalnya dapat dihitung nilai *Micro-Averaged F1-measure* dengan menghitung total *TP* (*True Positive*), *FP* (*False*

*Positive*), dan *FN (False Negative)*. Contoh perhitungannya berdasarkan pada pembahasan di Bab 5 sebelumnya dapat dihitung sebagai berikut:

$$\begin{aligned} \text{Precision} &= \text{TP} / (\text{TP} + \text{FP}) \\ &= 1351 / (1351 + 231) \\ &= 1351 / 1582 \\ &= \mathbf{0.853 (85,3\%)} \end{aligned}$$

$$\begin{aligned} \text{Recall} &= \text{TP} / (\text{TP} + \text{FN}) \\ &= 1351 / (1351 + 156) \\ &= 1351 / 1507 \\ &= \mathbf{0.896 (89,6\%)} \end{aligned}$$

$$\begin{aligned} \text{Micro-average F1-measure} &= 2 \times \text{Precision} \times \text{Recall} / (\text{Precision} + \text{Recall}) \\ &= 2 \times 0.853982300884956 \times 0.896483078964831 / \\ &\quad (0.853982300884956 + 0.896483078964831) \\ &= \mathbf{0.874 \text{ atau } 87,4\%.} \end{aligned}$$

Berdasarkan hasil evaluasi metrik kinerja model klasifikasi ujaran kebencian memberikan hasil yang sangat memuaskan, dengan presisi mencapai 85,3%, recall sebesar 89,6%, dan F1 Score mencapai 87,4%. Angka-angka tersebut menggambarkan keberhasilan model dalam mengatasi tantangan klasifikasi ujaran kebencian secara efektif. Presisi yang tinggi memungkinkan model untuk mengidentifikasi dengan akurat ujaran-ujaran yang termasuk dalam kategori kebencian, sementara recall yang tinggi menunjukkan kemampuan model untuk tidak melewatkkan ujaran kebencian yang sebenarnya. F1 Score yang mencapai 87,4% menunjukkan bahwa model mencapai keseimbangan yang baik antara presisi dan recall, hal ini sangat penting dalam konteks penanganan ujaran kebencian di lingkungan online. Hasil evaluasi ini memberikan keyakinan bahwa model dapat memberikan kontribusi positif terhadap upaya menciptakan platform online yang

lebih aman dan mengurangi dampak negatif dari ujaran kebencian. Meskipun perlu mempertimbangkan konteks spesifik dan sifat ujaran kebencian yang beragam, angka-angka yang tinggi ini menjadi pijakan yang kuat dalam menilai efektivitas model dalam mengatasi masalah kritis ini.

## **BAB VIII**

## **PENUTUP**

Dalam menyusun buku ini, kami berharap telah memberikan pandangan mendalam tentang teori dan konsep dari *machine Learning* dan deep learning serta implementasi pada kasus Sistem Deteksi Ujaran Kebencian dalam bahasa Indonesia (iHate-detection). Dari pengumpulan data hingga penerapan teknik pemrosesan bahasa alami yang canggih, panduan ini dirancang untuk membantu pembaca memahami kompleksitas serta tantangan yang terlibat dalam menghadapi konten negatif dalam bahasa Indonesia.

Penting untuk diingat bahwa deteksi ujaran kebencian adalah langkah krusial dalam menjaga lingkungan online yang aman dan inklusif. Dalam masyarakat digital yang semakin terhubung, upaya untuk mengidentifikasi dan merespons ujaran negatif memiliki dampak yang mendalam pada kualitas interaksi dan komunikasi kita. Dengan mengintegrasikan teknik seperti back-translation, ekspansi kata, disambiguasi kata, dan penggunaan model pretrained, kita dapat membangun sistem yang lebih responsif, akurat, dan efektif dalam menjaga keberlangsungan ruang maya yang positif.

Semoga buku ini memberikan tambahan ilmu yang berharga dalam upaya untuk mengembangkan sistem deteksi ujaran kebencian yang andal dan canggih berbasis pada pendekatan ekspansi semantik dan deep learning. Kami berharap bahwa informasi dan strategi yang telah kami bagikan akan membantu dalam menghadapi tantangan yang kompleks ini dan menjadikan internet tempat yang lebih aman bagi semua orang. Teruslah berinovasi dan berkolaborasi untuk menciptakan lingkungan online yang lebih baik, lebih inklusif, dan lebih peduli.



## DAFTAR PUSTAKA

- Affandes, M. (2015). Penerapan Metode Support Vector Machine (SVM) Menggunakan Kernel Radial Basis Faunction (RBF) Pada Klasifikasi Tweet. *SITEKIN: Jurnal Sains, Teknologi Dan Industri*, 12(2), 189–197.
- Alaa, Z., Tiun, S., & Abdulameer, M. (2016). Cross-language plagiarism of Arabic-English documents using linear logistic regression. *Journal of Theoretical and Applied Information Technology*, 83(1), 20–33. <https://www.jatit.org/volumes/Vol83No1/3Vol83No1.pdf>
- Amrutha, B. R., & Bindu, K. R. (2019). Detecting hate speech in tweets using different deep neural network architectures. *2019 International Conference on Intelligent Computing and Control Systems, ICCS 2019*, 923–926. <https://doi.org/10.1109/ICCS45141.2019.9065763>
- Ayo, F. E., Folorunso, O., Ibharalu, F. T., & Osinuga, I. A. (2020). Machine learning techniques for hate speech classification of twitter data: State-of-The-Art, future challenges and research directions. *Computer Science Review*, 38, 100311. <https://doi.org/10.1016/j.cosrev.2020.100311>
- Ayo, F. E., Folorunso, O., Ibharalu, F. T., & Osinuga, I. A. (2021). A probabilistic clustering model for hate speech classification in twitter. *Expert Systems with Applications*, 173. <https://doi.org/10.1016/j.eswa.2021.114762>
- Azhar, A. N., & Khodra, M. L. (2020). Fine-tuning pretrained multilingual bert model for indonesian aspect-based sentiment analysis. *2020 7th International Conference on Advance Informatics: Concepts, Theory and Applications (ICAICTA)*, 1–6.

- Bouhriz, N., & Benabbou, F. (2016). Word sense disambiguation approach for Arabic text. *International Journal of Advanced Computer Science and Applications*, 7(4).
- Brownlee, J. (2017). Deep learning for natural language processing. In *Machine Learning Mystery, Vermont, Australia* (Vol. 322).
- Deepa, M. D. (2021). Bidirectional encoder representations from transformers (BERT) language model for sentiment analysis task. *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, 12(7), 1708–1721.
- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv Preprint ArXiv:1810.04805*. <https://doi.org/https://doi.org/10.48550/arXiv.1810.04805>
- Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, 1, 4171–4186. <https://doi.org/https://doi.org/10.48550/arXiv.1810.04805>
- Dong, L., Wei, F., Tan, C., Tang, D., Zhou, M., & Xu, K. (2014). Adaptive recursive neural network for target-dependent twitter sentiment classification. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 49–54.
- Ekedahl, J., & Golub, K. (2004). Word sense disambiguation using WordNet and the Lesk algorithm. *Projektarbeten 2004*, 17.
- Farhan, Y. H., Mohd, M., & Noah, S. A. M. (2020). Survey of Automatic

Query Expansion for Arabic Text Retrieval. *Journal of Information Science Theory and Practice*, 8(4), 67–86.  
<https://doi.org/10.1633/JISTaP.2020.8.4.6>

Goyal, M., Reeves, N. D., Rajbhandari, S., & Yap, M. H. (2018). Robust methods for real-time diabetic foot ulcer detection and localization on mobile devices. *IEEE Journal of Biomedical and Health Informatics*, 23(4), 1730–1741.

Ibrohim, M. O., & Budi, I. (2019). Multi-label Hate Speech and Abusive Language Detection in Indonesian Twitter. *Proceedings of the Third Workshop on Abusive Language Online*, 46–57.  
<https://doi.org/10.18653/v1/w19-3506>

Kalchbrenner, N., Grefenstette, E., & Blunsom, P. (2014). A Convolutional Neural Network for Modelling Sentences. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 655–665.

Koto, F., Lau, J. H., & Baldwin, T. (2021). INDOBERTWEET: A Pretrained Language Model for Indonesian Twitter with Effective Domain-Specific Vocabulary Initialization. *EMNLP 2021 - 2021 Conference on Empirical Methods in Natural Language Processing, Proceedings*, 10660–10668.  
<https://doi.org/10.18653/v1/2021.emnlp-main.833>

Koto, F., Rahimi, A., Lau, J. H., & Baldwin, T. (2020). IndoLEM and IndoBERT: A Benchmark Dataset and Pre-trained Language Model for Indonesian NLP. *Proceedings of the 28th International Conference on Computational Linguistics*, 757–770.

Kowsari, K., Meimandi, K. J., Heidarysafa, M., Mendu, S., Barnes, L., & Brown, D. (2019). Text classification algorithms: A survey. *Information (Switzerland)*, 10(4), 1–68.

<https://doi.org/10.3390/info10040150>

- Kulmanov, M., Smaili, F. Z., Gao, X., & Hoehndorf, R. (2021). Semantic similarity and machine learning with ontologies. *Briefings in Bioinformatics*, 22(4), bbaa199.  
<https://doi.org/https://doi.org/10.1093/bib/bbaa199>
- Kumar, A., & Sachdeva, N. (2020). Multi-input integrative learning using deep neural networks and transfer learning for cyberbullying detection in real-time code-mix data. *Multimedia Systems*.  
<https://doi.org/10.1007/s00530-020-00672-7>
- Laatar, R., Aloulou, C., & Belghith, L. H. (2018). Word embedding for Arabic word sense disambiguation to create a historical dictionary for Arabic language. *2018 8th International Conference on Computer Science and Information Technology (CSIT)*, 131–135.  
<https://doi.org/https://doi.org/10.1109/csit.2018.8486159>
- Liu, Z., Yu, W., Chen, W., Wang, S., & Wu, F. (2010). Short text feature selection for micro-blog mining. *2010 International Conference on Computational Intelligence and Software Engineering*, 1–4.
- Luthfiarta, A., Zeniarja, J., & Salam, A. (2013). Algoritma Latent Semantic Analysis (LSA) pada peringkas dokumen otomatis untuk proses clustering dokumen. *Seminar Nasional Teknologi Informasi & Komunikasi Terapan*, 16.
- Mahmoud, A., & Zrigui, M. (2017). Semantic similarity analysis for paraphrase identification in Arabic texts. *Proceedings of the 31st Pacific Asia Conference on Language, Information and Computation*, 274–281. <https://aclanthology.org/Y17-1037>
- Manning, C. D. (2009). *An introduction to information retrieval*. Cambridge university press.
- Mathew, A., Amudha, P., & Sivakumari, S. (2020). Deep Learning

- Techniques: An Overview. *International Conference on Advanced Machine Learning Technologies and Applications*, 599–608.
- Miao, Y. L., Ji, Y. C., & Peng, E. Lou. (2019). Application of CNN-BiGRU model in chinese short text sentiment analysis. *PervasiveHealth: Pervasive Computing Technologies for Healthcare*, 510–514. <https://doi.org/10.1145/3377713.3377804>
- Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems*, 26.
- Muzakir, A., Adi, K., & Kusumaningrum, R. (2023). Advancements in Semantic Expansion Techniques for Short Text Classification and Hate Speech Detection. *Ingenierie Des Systemes d'Information*, 28(3), 545. <https://doi.org/https://doi.org/10.18280/isi.280301>
- Naskar, S. K., & Bandyopadhyay, S. (2007). Word sense disambiguation using extended wordnet. *2007 International Conference on Computing: Theory and Applications (ICCTA'07)*, 446–450. <https://doi.org/https://doi.org/10.1109/ICCTA.2007.134>
- Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1532–1543. <https://doi.org/http://dx.doi.org/10.3115/v1/D14-1162>
- Prihatini, P. M. (2016). Implementasi Ekstraksi Fitur Pada Pengolahan Dokumen Berbahasa Indonesia. *Jurnal Manajemen Teknologi Dan Informatika (MATRIX)*, 6(3), 174–178.
- Qiu, H., Fan, C., Yao, J., & Ye, X. (2020). Chinese Microblog Sentiment Detection Based on CNN-BiGRU and Multihead Attention

Mechanism. *Scientific Programming*, 2020.

<https://doi.org/10.1155/2020/8865983>

Riyaddulloh, R., & Romadhony, A. (2021). Normalisasi Teks Bahasa Indonesia Berbasis Kamus Slang Studi Kasus: Tweet Produk Gadget Pada Twitter. *EProceedings of Engineering*, 8(4).

Salminen, J., Almerekhi, H., Milenković, M., Jung, S.-G., An, J., Kwak, H., & Jansen, B. J. (2018). Anatomy of online hate: Developing a taxonomy and machine learning models for identifying and classifying hate in online news media. *12th International AAAI Conference on Web and Social Media, ICWSM 2018*, 330–339.

Saputro, I. W., & Sari, B. W. (2020). Uji Performa Algoritma Naïve Bayes untuk Prediksi Masa Studi Mahasiswa. *Creative Information Technology Journal*, 6(1), 1–11.

Sethy, A., & Ramabhadran, B. (2008). Bag-of-word normalized n-gram models. *Ninth Annual Conference of the International Speech Communication Association*.

Sharma, D. K., Pamula, R., & Chauhan, D. S. (2021). Semantic approaches for query expansion. *Evolutionary Intelligence*, 14(2), 1101–1116.  
<https://doi.org/10.1007/s12065-020-00554-x>

Sindhu, C., Som, B., & Singh, S. P. (2021). Aspect-Oriented Sentiment Classification using BiGRU-CNN model. *Proceedings - 5th International Conference on Computing Methodologies and Communication, ICCMC 2021, Iccmc*, 984–989.  
<https://doi.org/10.1109/ICCMC51019.2021.9418242>

Soyusiawaty, D., Jones, A. H. S., & Lestariw, N. L. (2020). The stemming application on affixed javanese words by using nazief and adriani algorithm. *IOP Conference Series: Materials Science and Engineering*, 771(1), 12026.

- Sundermeyer, M., Schlüter, R., & Ney, H. (2012). LSTM neural networks for language modeling. *Thirteenth Annual Conference of the International Speech Communication Association*.
- Tala, F. (2003). *A study of stemming effects on information retrieval in Bahasa Indonesia*.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 5998–6008.
- Verma, T., Renu, R., & Gaur, D. (2014). Tokenization and filtering process in RapidMiner. *International Journal of Applied Information Systems*, 7(2), 16–18.
- Xie, J., Chen, B., Gu, X., Liang, F., & Xu, X. (2019). Self-Attention-Based BiLSTM Model for Short Text Fine-Grained Sentiment Classification. *IEEE Access*, 7, 180558–180570. <https://doi.org/10.1109/ACCESS.2019.2957510>
- Zhang, & LeCun, Y. (2015). Text understanding from scratch. *ArXiv Preprint ArXiv:1502.01710*.
- Zhang, Y., & Wallace, B. (2017). A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification. *Proceedings of the The 8th International Joint Conference on Natural Language Processing*, 253–263. <http://arxiv.org/abs/1510.03820>



## BIODATA PENULIS



Dr.(c) Ari Muzakir, S.Kom, M.Cs, saat ini tengah menempuh pendidikan pada Program Doktor Sistem Informasi di Universitas Diponegoro, yang berlokasi di Semarang, 50241, Indonesia. Tidak hanya sebagai Mahasiswa, saat ini juga berkontribusi sebagai Dosen tetap di Fakultas Sains dan Teknologi, Universitas Bina Darma di Palembang, 30761, Indonesia. Perjalanan akademiknya di Universitas Bina Darma dimulai sejak ia meraih gelar Sarjana dalam Ilmu Komputer pada tahun 2009. Kemudian pada tahun 2010 yang dilanjutkan dengan pencapaian gelar Magister pada bidang Ilmu Komputer, Universitas Gadjah Mada di Yogyakarta, Indonesia. Saat ini fokus pada peminatan riset di bidang ilmu komputer, khususnya Artificial Intelligence (AI).



Prof. Dr. Kusworo Adi, S.Si., M.T., saat ini selain sebagai Dosen sekaligus menjabat sebagai Wakil Dekan Bidang Sumberdaya, Bisnis dan Komunikasi, Fakultas Sains dan Matematika di Universitas Diponegoro, Semarang, 50271, Indonesia. Pendidikan formal dimulai di Universitas Diponegoro, di mana dia meraih gelar Sarjana pada bidang Fisika pada tahun 1996. Keinginannya untuk terus belajar membawanya menyelesaikan gelar Magister tahun 2002 dan Doktor pada tahun 2010 pada bidang Teknik Elektro dan Informatika di Institut teknologi Bandung (ITB), Bandung, Indonesia. Dalam dunia riset, memiliki kepakaran khusus di bidang Pencitraan dan Pengolahan Citra.



Dr. Retno Kusumaningrum, S.Si., M.Kom., saat ini sebagai dosen tetap di Program Studi S1 Informatika, Universitas Diponegoro, Semarang, 50271, Indonesia. Pendidikan tingginya dimulai dari kampus yang sama, Universitas Diponegoro, di mana ia meraih gelar sarjana dalam bidang Matematika pada tahun 2003. Kemudian melanjutkan jenjang pendidikan Magister Ilmu Komputer tahun 2010 dan program Doktor Ilmu Komputer tahun 2014 di Fakultas Ilmu Komputer, Universitas Indonesia, Depok, Indonesia. Dalam meraih gelar Doktor, ia berhasil mendapatkan beasiswa *Sandwich-Like Program* di Computer Vision and Pattern Recognition Group, School of Systems Engineering, University of Reading, United Kingdom. Dengan bidang kepakaran yang kuat, saat ini fokus pada riset-riset di bidang Machine Learning, Natural Language Processing, Computer Vision, Pattern Recognition, dan Topic Modelling.

Buku ini dirancang sebagai bahan pengantar atau pendukung mata kuliah machine Learning dan text mining untuk berbagai jurusan, terutama tingkat sarjana, dan bisa juga digunakan pada tingkat pascasarjana. Namun, perlu ditekankan bahwa buku ini sebatas merupakan pelengkap, bukan sumber informasi utama. Isinya mencakup materi dasar mengenai machine Learning, deep learning, pendekatan semantik, dan studi kasus yang dirancang sedemikian rupa sehingga pembaca dapat memperoleh intuisi yang kuat. Penting untuk diingat bahwa buku ini tidak merinci setiap aspek secara mendalam, sehingga pembaca tetap perlu merujuk sumber informasi lain untuk pemahaman yang lebih komprehensif.

Dalam buku ini memberikan gambaran konsep dan teori yang terkait dengan Artificial Intelligence (AI) dan Natural Language Processing (NLP). Metode dan algoritma disajikan secara deskriptif dan didukung oleh ilustrasi agar pembaca dapat menggambarkan inti dari topik yang dibahas. Selain itu, buku ini menghadirkan studi kasus penerapan beberapa metode dan algoritma dalam konteks ujaran kebencian dalam bahasa Indonesia. Solusi untuk masalah deteksi ujaran kebencian disajikan dengan menggunakan pendekatan perluasan semantik dan algoritma deep learning.

Namun, seperti halnya buku lainnya, buku ini juga memiliki beberapa kekurangan, seperti kesalahan tipografi dan kesalahan lainnya. Oleh karena itu, pembaca disarankan untuk membaca dengan cermat, menginterpretasikan variabel dalam persamaan, dan merujuk pada sumber-sumber lain untuk mendapatkan pemahaman yang lebih komprehensif.

diterbitkan oleh  
**UNDIP**  
**PRESS**

