BAB 2
LANDASAN TEORI

2.1 *Machine Learning*

Machine Learning merupakan studi tentang berbagai algoritma yang membantu membuat komputer belajar dan bekerja dengan sendiri- nya, dengan penerapan *machine learning* maka sistem akan dengan sendirinya membuat keputusan terbaik berdasarkan pola data pada sistem, tanpa banyaknya campur tangan dari manusia [9]. *Machine learning* memiliki karakteristik sebagai berikut :

1. Bagian dari *artificial intelligence*
2. Lebih besar dan umum daripada *deep learning*
3. Masih membutuhkan panduan lewat algoritma AI dengan bantuan tangan manusia sebagai pemrograman dan pembuat metode komputasi-nya.

Pada *machine learning* memiliki tiga metode atau tipe pada penerapan algoritmanya yaitu *Supervised Learning, Unsupervised learning* dan *Reinforcement learning.* Berikut penjelasannya :

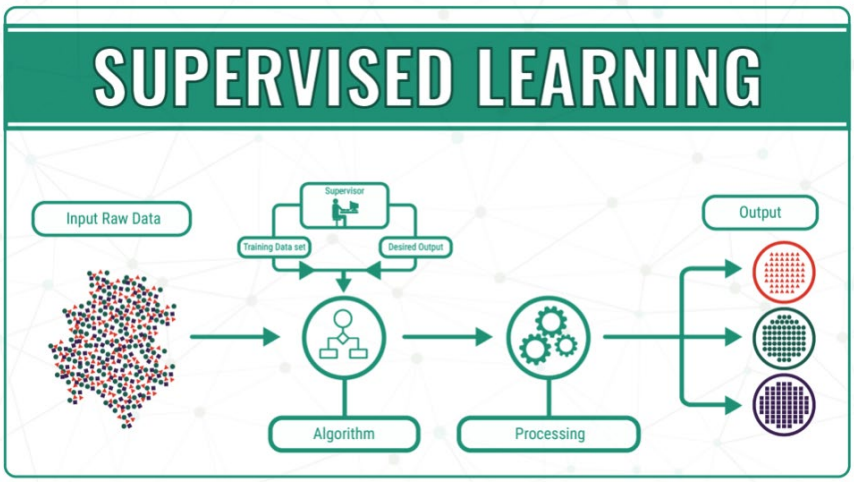
1. Supervised Learning

   Supervised learning merupakan sebuah metode dari *machine learning,* pada *supervised learning* ini sudah memiliki nilai dari label data sebagai *training* dari algoritma dan juga nilai keluaran data yang dapat diamati [10]. Teknik *Supervised machine learning,* terbagi menjadi tiga kategori yaitu :

   a. Similarity-based
   b. Model-based
   c. Probabilistic-approaches

   Tujuan dari pemodelan algoritma *supervised learning* adalah melatih algoritma untuk berjalan secara otomatis memetakan data *input*

ke data *output*. Pemodelan *supervised learning* memiliki tahapan dan proses-nya sebagai berikut :



Gambar 2.1 Proses Supervised Learning

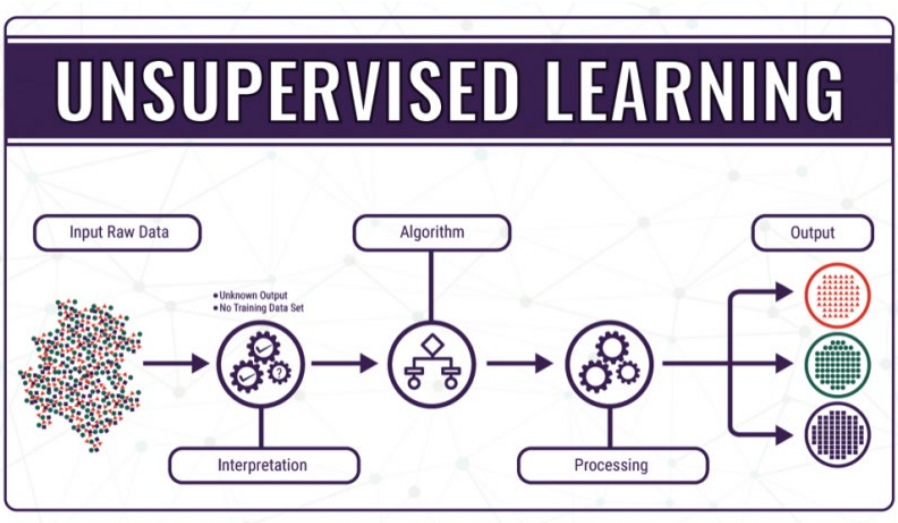
Pada gambar diatas proses *supervised learning* memiliki beberapa tahapan dengan detail penjelasan menurut [11].

a. *Input data raw*, pada tahap ini melakukan input data mentah yang akan diproses dan memastikan bahwa *dataset* memiliki label pada datanya.
b. Algoritma, pada tahap ini melakukan pemilihan algoritma yang akan dipakai, menyesuaikan dengan kasus yang akan dipecahkan, serta melakukan *split data training* dan *data test*.
c. *Processing*, pada tahap ini melakukan pemrosesan data dengan algoritma yang telah dipilih sesuai dengan pemecahan kasus yang ada.
d. *Output*, merupakan tahap terakhir yaitu hasil dari evaluasi penilaian yang didapatkan dengan pemecahan algoritma yang digunakan.

2. Unsupervised Learning

Unsupervised learning merupakan pemodelan dari *machine learning,* pada *unsupervised learning* ini tidak memiliki sebuah label data dalam pengamatan data-nya, tujuan dari pemodelan ini untuk

melatih algoritma menemukan sebuah pola, korelasi atau klister dalam sebuah *dataset* [12]. Pemodelan *unsupervised learning* memiliki tahapan dan prosesnya sebagai berikut



Gambar 2.2 Proses Unsupervised Learning

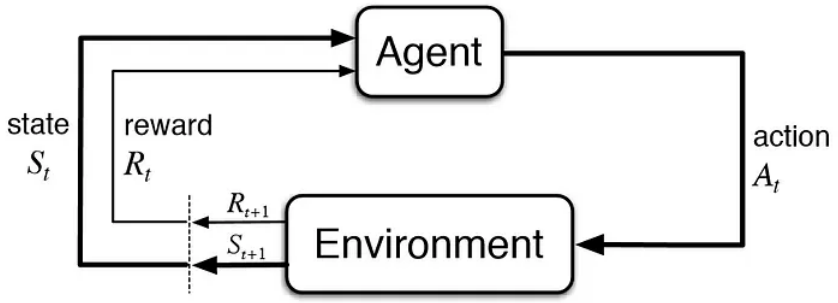
Pada gambar diatas, *unsupervised learning* memiliki beberapa tahapan dengan detail penjelasan:

a. *Input data raw* pada tahap ini melakukan input data mentah yang akan diproses dan memastikan bahwa *dataset* tidak memiliki label dalam datanya.
b. *Interpretation* melakukan pengolahan *data knowledge* pada data yang akan diproses.
c. Algoritma pada tahap ini melakukan pemilihan algoritma yang sesuai dengan *study case* untuk mendapatkan hasil yang optimal.
d. *Processing* pada tahap ini algoritma akan melakukan proses algoritma dengan hasil yang optimal.
e. *Output* melakukan evaluasi data yang telah didapatkan apakah data sudah sesuai atau tidaknya.

3. Reinforcement Learning

Reinforcement learning merupakan sebuah metode dari *machine learning* berbasis umpan balik dimana agen belajar berperilaku di

lingkungan dengan melakukan tindakan dan melihat hasil tindakannya. Dalam kasus ini agen belajar secara otomatis menggunakan umpan balik tanpa data yang berlabel. Maka algoritma yang akan diimplementasikan berdasarkan pengalamannya saja atau berdasarkan umpan balik yang didapatkan [13]. Tujuan dari *reinforcement learning* adalah untuk pembelajaran agen seperti melakukan interaksi yang baik dengan lingkungannya sehingga agen akan mendapatkan nilai yang baik dalam performa algoritmanya. Berikut merupakan alur dari sebuah *reinforcement learning*



Gambar 2.3 Reinforcement Learning

2.2 Myers-Briggs Type Indicator (MBTI)

Tipe kepribadian MBTI ditentukan dengan cara mengisi kuesioner yang berisi sejumlah pertanyaan singkat. MBTI mengadopsi dikotomi preferensi yang diajukan oleh Carl Jung yaitu *Extroversion Introversion* (E/I), *Sensing-Intuition* (S/N), dan *Thinking-Feeling* (T/F), Myers dan Briggs kemudian menambahkan preferensi keempat yaitu *Judging Perceiving* (J/P) [2]. Berdasarkan empat domain tersebut, sebagai contoh jika hasil evaluasi psikologis individu menunjukkan cenderung pada *Introversion* (I), *Intuition* (N), *Feeling* (F), dan *Judging* (J), maka individu tersebut akan diklasifikasikan sebagai tipe kepribadian INFJ. Dengan demikian, terdapat total 16 tipe kepribadian yang dihasilkan dari kombinasi empat domain ini. Karakteristik dikotomi preferensi ini dipaparkan lebih lanjut pada Tabel 2.1.

Tabel 2.1 Karakteristik Preferensi MBTI

Extroversion (E) Mendapat energi dari bertemu dengan orang lain, menikmati tugas-tugas yang bervariasi, bertempo cepat, dan baik dalam multitasking.	**Introversion (I)** Seringkali lebih menyukai bekerja sendiri atau dalam kelompok kecil, bertempo semaunya, dan berfokus pada satu tugas dalam satu waktu.
Sensing (S) Berpikir realistis dengan fokus pada fakta dan informasi terperinci, menerapkan pengetahuan umum dan pengalaman terdahulu untuk menyelesaikan masalah secara praktis.	**Intuition (N)** Berpikir imajinatif dengan fokus pada kemungkinan kemungkinan dan hal yang lebih besar, baik dalam mengenali pola dan nilai, serta mencari solusi kreatif terhadap sebuah masalah.
Thinking (T) Menentukan keputusan berdasarkan analisis logis, objektif menilai kelebihan dan kekurangan, dan menghargai kejujuran, konsistensi, serta keadilan.	**Feeling (F)** Menentukan keputusan berdasarkan perasaan, sensitif dan kooperatif, mempertimbangkan orang lain dalam pengambilan keputusan.
Judging (J) Terorganisir dan tertata, senang membuat rencana dan menjalaninya, serta tak keberatan mengikuti aturan.	**Perceiving (P)** Senang dengan pilihan terbuka, bertindak secara spontan, fleksibel dengan jadwal dan aturan.

2.3 *Preprocessing*

Preprocessing adalah langkah penting dalam pengolahan data teks sebelum digunakan dalam model pembelajaran mesin. Proses ini bertujuan untuk membersihkan dan menyesuaikan data teks agar sesuai dengan kebutuhan analisis dan model yang akan dibangun. Beberapa teknik pra-proses teks yang umum digunakan meliput *cleaning*, *case folding*, *tokenization*, *stopword removal*, dan *lemmatization*. Berikut penjelasan dari setiap tahapannya :

2.3.1. *Cleaning*

Cleaning ini mencakup penghapusan tanda baca dan karakter khusus dari teks, karena elemen-elemen ini sering kali tidak memiliki kontribusi signifikan Dengan menghapus tanda baca dan karakter khusus, teks menjadi lebih bersih dan seragam, memudahkan model dalam mengidentifikasi pola dan memahami isi teks dengan lebih baik.

Sebagai contoh, kalimat "Hello, world! How's everything?" akan diubah menjadi "Hello world Hows everything", sehingga mengurangi gangguan dari simbol-simbol yang tidak relevan dan meningkatkan efisiensi analisis selanjutnya.

2.3.2. *Case Folding*

Proses mengubah semua karakter huruf pada sebuah kalimat menjadi huruf kecil atau huruf besar disebut *case folding*. *Case folding* yang dilakukan pada penelitian ini yaitu mengubah seluruh *dataset* menjadi huruf kecil. Huruf kapital biasanya terdapat pada setiap awal kalimat seperti “Hello World”, menggunakan *case folding* kalimat tersebut berubah menjadi “hello world”. Tujuan utama *case folding* adalah agar kata “hello” tidak lagi mempunyai dua bentuk yaitu, “Hello”, dan “hello”, namun hanya memiliki satu bentuk huruf kecil saja

2.3.3. Tokenization

Tokenization adalah proses pemecahan teks menjadi unit-unit yang lebih kecil yang dikenal sebagai token. Sebagai contoh, kalimat “Hello world Hows everything” akan dipecah menjadi [‘Hello’, ‘world’, ‘hows’, ‘everything’]. Sehingga proses ini memungkinkan analisis dan pemrosesan teks secara lebih efektif dan efisien. Proses ini membantu dalam berbagai tugas *Natural Language Processing* (NLP) seperti pengindeksan dokumen, pemrosesan teks, analisis sentimen, dan penerjemahan mesin.

2.3.4. *Stopword Removal*

Stopword Removal merupakan tahap pengambilan kata-kata penting dan membuang kata-kata yang dianggap tidak penting. Cara untuk membuang kata yang tidak penting disebut *stopword removal*. *Stopword removal* bertujuan untuk menghilangkan kata-kata yang sering muncul namun tidak memiliki kontribusi dalam proses analisis data [14]. *Stopword removal* berusaha memperkecil dimensi data dan mempercepat waktu komputasi Contoh kata yang tidak penting di bahasa inggris seperti kata “and”, “the”, “an”, “are”.

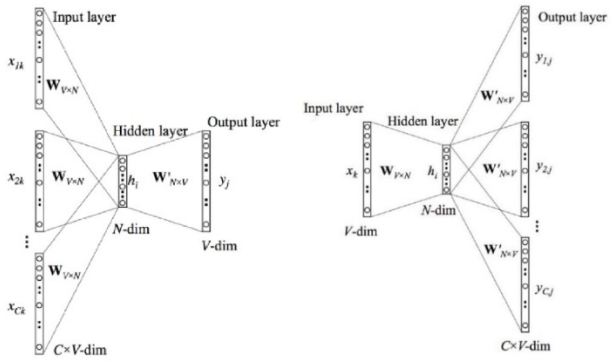
2.3.5. *Lemmatization*

Lemmatization adalah salah satu teknik dalam tahap praproses (*preprocessing*) teks yang bertujuan untuk mengubah berbagai bentuk kata menjadi bentuk dasarnya, yang disebut lemma. Proses ini mempertimbangkan konteks dan bagian dari tata bahasa (*part of speech*) dari kata-kata dalam kalimat. Dengan menggunakan *lemmatization*, kata-kata seperti “better”, “best”, dan “good” akan diubah menjadi bentuk dasar mereka, yaitu “good” [15]. Berbeda dengan *stemming*, yang memotong akhiran kata tanpa mempertimbangkan konteks dan sering menghasilkan bentuk yang tidak valid, *lemmatization* menggunakan kamus dan analisis morfologis untuk memastikan hasil adalah bentuk dasar yang sah dalam bahasa.

2.4 Word2Vec

Word embedding adalah proses konversi sebuah kata menjadi nilai bilangan (vektor) sehingga dapat diolah oleh komputer dan dapat menghitung kemiripan antar kata dan juga bekerja cepat pada data yang besar [6]. Model arsitektur Word2Vec bekerja menggunakan *Neural Network* (NN), mengkalkulasi *cosine similarity* antara vektor-vektor kata. Kata yang mirip akan memiliki nilai *cosine similarity* yang tinggi. Terdapat dua model arsitektur pada Word2Vec, yaitu model *Continuous Bag of Words* (CBOW)

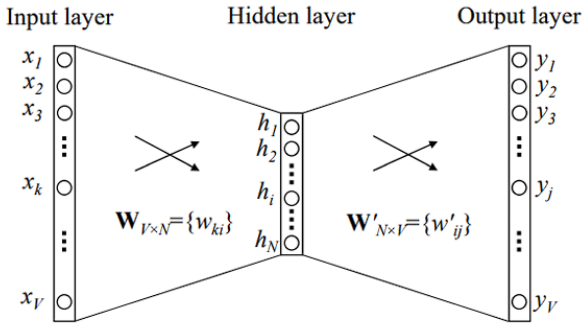
dan model *Skip-gram*. Cara kerja kedua model ini saling berkebalikan, masukan untuk CBOW adalah sejumlah kata dan luarannya berupa satu kata, sedangkan masukan untuk *Skip-gram* adalah satu kata dan luarannya berupa sejumlah kata [16]. Ilustrasi perbedaan arsitektur CBOW dan *Skip-gram* dapat dilihat pada Gambar 2.4



Gambar 2.4 Arsitektur CBOW dan Skip Gram

Continuous Bag-of-Words (CBOW) sering dipakai untuk aplikasi NLP yang mencoba untuk memprediksi sebuah target kata disekitar konteksnya biasanya terdiri dari beberapa kata yang berdekatan. CBOW tidak bergantung pada urutan kata-kata atau harus pada karakteristik probabilistiknya.

Jika hanya satu kata yang dijadikan konteks [17], maka arsitektur CBOW-nya dapat dilihat pada gambar 2.5.



Gambar 2.5 Arsitektur CBOW dengan Satu Kata Konteks

Pada gambar terdapat simbol k adalah konteks kata, j adalah target kata, V yang berarti jumlah *vocabulary*, dan N adalah jumlah *hidden layer*, *unit*

layer tersebut *fully connected*, dan *input layer* berupa vektor *one-hot encoding*, berarti untuk input kata konteks posisinya diisi nilai 1, dan sisanya 0 [17].

Dimulai dari perhitungan *hidden layer* (h), namun sebelumnya menentukan matriks bobot W dengan dimensi V × N. Setiap baris dari matriks bobot W merepresentasikan vektor berukuran N-dimensi dari kata terkait di *input layer*. Jadi baris i dari matriks bobot W adalah v_w^T. Perhitungannya perkalian matriks antara matriks bobot W *transpose* dengan *input layer* yang berupa *one-hot encoding*. Nilai *hidden layer* didapatkan dengan persamaan (2.1).

$$h = W^T x \coloneqq v_{w_I}^T \quad \text{.....(2.1)}$$

Keterangan: W^T adalah matriks bobot W *transpose*

x adalah *input layer*

v_{w_I} adalah representasi vektor dari input kata (wI)

Lalu setelah *hidden layer* dihitung, hitung *output* u, namun matriks bobotnya berbeda dari yang sebelumnya, yaitu bobot matriksnya adalah $W' = \{w'_{ij}\}$ ukurannya adalah N × V. Perhitungannya perkalian matriks antara kolom j bobot matriks W' dengan *hidden layer* pada persamaan (2.2).

$$u_j = {v'_{w_j}}^T h \quad \text{.....(2.2)}$$

Keterangan : ${v'_{w_j}}^T$ adalah kolom j dari matriks bobot W' *transpose*

h adalah *hidden layer*

Lalu menghitung *output* dari unit j di *output layer* dengan fungsi *softmax* untuk mendapatkan distribusi posterior kata-kata. Catatan bahwa vw adalah representasi baris dari matriks bobot W', v'_w representasi kolom dari

matriks bobot W' namun analisis berikutnya akan berubah. Nilai *output softmax* didapatkan dengan persamaan (2.3).

$$p(w_j|w_I) = y_j = \frac{\exp(u_j)}{\sum_{j'=1}^{V} \exp(u_{j'})} \quad \text{.....(2.3)}$$

Keterangan : u_j adalah *output* u unit ke-j

y_j adalah *output* dari unit j di *output layer*

V adalah jumlah *Vocabulary*

Setelah didapatkan hasil keluarannya, *update* perhitungan *hidden* dan bobot *output*. Dengan menerapkan *backpropagation* dan melakukan turunan. Tujuan dari pelatihan adalah untuk maksimalkan persamaan (2.3) probabilitas bersyarat mengamati *output* kata sebenarnya dari j* mengingat input kata konteks w_I berkaitan dengan bobot. Namun sebelum itu perlu menghitung *loss function* atau *error* yang tujuannya untuk meminimumkan nilai E dengan persamaan (2.4).

$$\max p(w_o|w_I) = u_{j^*} - \log \sum_{j'=1}^{V} exp(u_{j'}) := -E \quad \text{.....(2.4)}$$

Keterangan : w_I adalah input konteks kata

w_O adalah *output* kata sebenarnya atau target kata

u_{j^*} adalah *output* u kata sebenarnya

$u_{j'}$ adalah *output* u kata dalam *vocabulary* selain target kata

E adalah *loss function*

Lalu menghitung turunan bobot untuk memperbarui bobot pada *hidden* dan *output layer*. Turunan parsial E yang berkaitan dengan input unit ke-j dari u_j dapat dihitung dengan persamaan (2.5).

$$\frac{\partial E}{\partial u_j} = y_j - t_j := e_j \quad \text{.....(2.5)}$$

Keterangan : t_j adalah $1(j = j^*)$ akan bernilai 1 ketika posisi unit ke-j adalah nilai sebenarnya dari kata *output*, jika sebaliknya bernilai 0

y_j adalah *output* dari unit j di *output layer*

e_j adalah prediksi *error* di *output layer*

Selanjutnya menghitung turunan parsial E pada w'_{ij} untuk memperoleh gradien yang terdapat pada bobot *hidden* dan *output*. Perhitungannya dengan persamaan (2.4.6).

$$\frac{\partial E}{\partial w'_{ij}} = \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial w'_{ij}} = e_j \otimes h_i \quad \text{.....(2.6)}$$

Keterangan : e_j adalah prediksi *error*

h_i adalah unit ke-i pada *hidden layer*

$\otimes$ adalah *outer product* atau *tensor product*

Menggunakan *stochastic gradient descent* untuk memperoleh bobot baru (W'). Sebagai catatan bahwa harus menulusuri setiap kemungkinan kata yang terdapat pada *vocabulary*. Periksalah *output* dari probabilitas y_j dan bandingkan dengan *output* yang diharapkan (t_j) 0 atau 1, jika $y_j > t_j$ maka akan melakukan pengurangan proporsi pada vektor *hidden layer* (h) dari v'_{wj}, (sehingga membuat v'_{wj} lebih jauh dari v_{w_I}), jika $y_j < t_j$ yang mana benar hanya jika $t_j = 1(w_j = w_O)$, maka akan menambahkan beberapa h ke v'_{w_O}, sehingga membuat v'_{w_O} mendekat ke v_{w_I}. Dan jika nilai y_j terlalu dekat ke t_j, maka berdasarkan perhitungan baru, perubahan bobot baru akan sangat sedikit atau kecil. Perhitungan memperbarui bobot W' dilakukan dengan persamaan (2.7).

$$w_{ij}'^{(baru)} = w_{ij}'^{(lama)} - \eta \cdot e_j \cdot h_i \quad \text{.....(2.7)}$$

Keterangan : η adalah *learning rate*

w_{ij}' adalah bobot W' dengan i baris dan j kolom

h_i adalah *hidden layer*

e_j adalah prediksi *error*

Selanjutnya melakukan pembaruan perhitungan bobot antara input dan *hidden layer* (W). Mengambil nilai turunan parsial dari E pada *output* dari *hidden layer*. Perhitungan turunan E terhadap *hidden layer* dapat dilakukan dengan persamaan (2.4.8).

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^{V} \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial h_i} = \sum_{j=1}^{V} e_j \cdot w_{ij}' \coloneqq EH_i \quad \text{.....(2.8)}$$

Keterangan : h_i adalah *hidden layer*

w_{ij}' adalah matriks bobot W'

h_i adalah prediksi *error* dari kata ke-j pada *output layer*

EH adalah sebuah vektor N-dim, jumlah vektor *output* dari semua kata yang ada pada *vocabulary*, dibobotkan oleh prediksi *error*nya.

Perhitungan dari turunan E terhadap bobot W sama dengan perhitungan *tensor product* antara x dan EH dilakukan dengan persamaan (2.9)

$$\frac{\partial E}{\partial W} = x \otimes EH = xEH^T \quad \text{.....(2.9)}$$

Keterangan : x adalah input vektor bukan nol

EH^T adalah sebuah vektor N-dim, jumlah vektor *output* dari semua kata yang ada pada *vocabulary*, dibobotkan oleh prediksi *error*nya.

$\otimes$ adalah *outer product* atau *tensor product*

Lalu perhitungan untuk mendapatkan bobot baru v_{w_I}, dikarenakan hanya satu nilai x yang bukan nol dan nilai $\frac{\partial E}{\partial W}$ ini sama dengan nilai EH^T jadi menghitung bobot baru W dengan persamaan (2.4.10)

$$v_{w_I}^{(baru)} = v_{w_I}^{(lama)} - \eta\, EH^T \quad \text{.....(2.10)}$$

Keterangan : v_{w_I} adalah baris bobot W, input vektor hanya merupakan kata konteks dan hanya baris dari W yang turunannya bernilai bukan nol

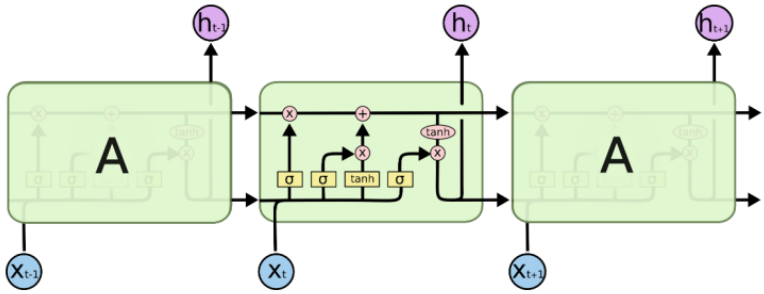
η adalah *learning rate*

EH^T adalah sebuah vektor N-dim, jumlah vektor *output* dari semua kata yang ada pada *vocabulary*, dibobotkan oleh prediksi *error*nya *transpose*.

2.5 Long-Short Term Memory (LSTM)

LSTM adalah variasi *Recurrent Neural Network* (RNN) yang dikembangkan dengan tujuan menghindari masalah dependensi jangka panjang pada RNN, pertama kali dikenalkan pada tahun 1997 oleh Hochreiter & Schmidhuber. Dalam RNN, *network loop* hanya menggunakan satu *layer* yaitu *tanh*, sedangkan pada LSTM menggunakan satu *layer*. LSTM terdiri dari tiga gerbang yaitu *forget gate*, berguna untuk menentukan informasi mana yang akan dihapus dari sel; *input gate*, berguna untuk menentukan nilai masukan mana yang akan diperbarui pada memory state; dan *output gate*, berguna untuk menentukan luaran apa yang akan dihasilkan berdasarkan masukan dan memori yang ada dalam sel [18].

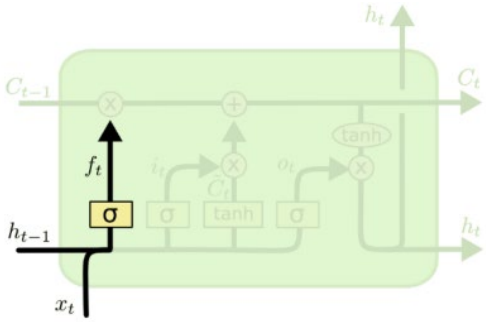
LSTM dapat mendeteksi data yang akan disimpan dan data yang tidak digunakan untuk dipangkas, karena LSTM memiliki 4 *layer* neuron yang biasa disebut *gates* untuk mengatur memori pada setiap neuron [19]. Komponen utama dari struktur LSTM adalah *forget gate*, *input gate*, *cell state*, dan *output gate*.



Gambar 2.6 Jaringan LSTM

Berdasarkan gambar 2.6 jaringan LSTM berikut dapat dijabarkan :

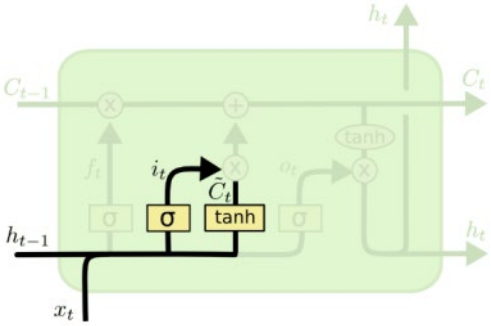
1. Kotak berwarna kuning merupakan *neural network layer* yaitu jaringan saraf yang dipelajari.
2. Anak panah tanpa cabang atau garis penggabungan adalah *vector transfer* yaitu garis yang membawa seluruh vektor, dari keluaran satu neuron ke masukan lainnya.
3. Lingkaran berwarna merah muda adalah *pointwise operation* yaitu operasi yang dilakukan seperti penambahan vektor.
4. Garis yang mengalami penggabungan adalah *concatenate* yaitu proses penggabungan yang menandakan rangkuman.
5. Garis bercabang adalah *Copy* yaitu proses penyalinan nilai yang kemudian dibawa ke lokasi yang berbeda.
6. Notasi xt menyatakan masukan untuk setiap unit pada *time step t dan ni*lainya berupa vektor.
7. Notasi ht menyatakan *hidden* state yang dihasilkan oleh setiap unit pada *time step t dan ni*lainya berupa vektor.
8. Notasi Ct menyatakan *cell state* yang dihasilkan oleh setiap unit pada *time step t dan ni*lainya berupa vektor.



Gambar 2.7 *Forget Gate* LSTM

Persamaan 2.11 adalah persamaan untuk *output forget gate*. Persamaan ini berfungsi untuk memutuskan apakah informasi akan disimpan atau dilupakan dari *cell state* Ct-1 yang merupakan hasil perkalian dengan *cell state*. Keputusan ini dibuah oleh fungsi *sigmoid*. Fungsi *sigmoid* yang menerima input h_{t-1} dan x_t akan menghasilkan angka antara nol dan satu. Angka 1 artinya *cell state* akan disimpan. Angka 0 artinya *cell state* akan dilupakan. Proses ini digambarkan dalam gambar 2.7.

$$f_t = \sigma\left(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f\right) \quad (2.11)$$

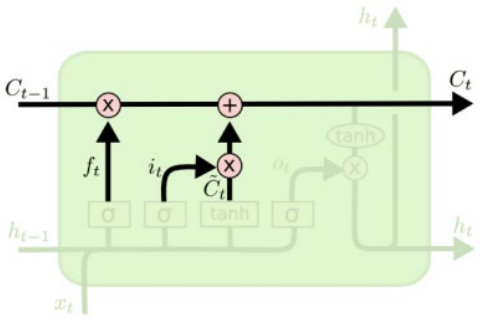


Gambar 2.8 *Input Gate* LSTM

Persamaan 2.12 adalah persamaan gerbang input LSTM. Secara sederhana, persamaan ini menciptakan kandidat baru untuk *cell state*. *Input gate* memilik dua bagian. Pertama yaitu *sigmoid* yang akan menentukan nilai yang mana yang akan diperbarui. Selanjutnya yaitu fungsi aktivasi *tanh* yang menciptakan kandidat baru yang ditambahkan dengan *cell state*. Pada langkah

selanjutnya akan digabungkan untuk membuat pembaruan *cell state*. Proses ini dapat dilihat pada gambar 2.8.

$$i_t = \sigma(W_i \cdot x_i + U_i \cdot h_{t-1} + b_i) \quad (2.12)$$

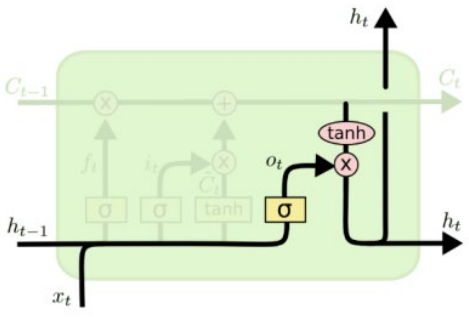


Gambar 2.9 *Cell State* LSTM

Cell state kemudian diperbarui berdasarkan persamaan 2.14. Sederhananya, *cell state* akan menggabungkan *cell state* hasil dari *forget gate* dan kandidat baru hasil dari *input gate*. *Cell state* hasil dari *forget gate* yang memutuskan apakah akan dilupakan atau diteruskan didapat dengan mengalikan ft dengan *cell state* sebelumnya. Calon kandidat baru *cell state* didapat berdasarkan persamaan 2.13. Proses ini dapat dilihat pada gambar 2.9.

$$\tilde{C}_t = tanh(W_c \cdot x_c + U_c \cdot h_{t-1} + b_c) \quad (2.13)$$

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C} \quad (2.14)$$



Gambar 2.10 *Output Gate* LSTM

Setelah *cell state* diperbarui, *output gate* dapat dihitung menggunakan persamaan 2.15 untuk menghasilkan *hidden* state. *Hidden* state didasarkan pada *cell state* namun disaring kembali. Pertama, fungsi aktivasi *sigmoid* akan

memutuskan bagian mana dari *cell state* yang akan dijadikan *output*. Kemudian *cell state* dimasukkan dalam fungsi aktivasi *tanh* yang menghasilkan nilai antara -1 dan 1 kemudian dikalikan dengan hasil dari *sigmoid* yang akan menghasilkan *output*.

$$o_t = \sigma(W_o \cdot x_o + U_o \cdot h_{t-1} + b_0) \quad (2.15)$$

$$h_t = o_t \cdot tanh\,(C_t) \quad (2.16)$$

2.6 Fungsi Aktivasi

Fungsi aktivasi adalah fungsi yang digunakan dalam jaringan saraf untuk menghitung jumlah input dan bias yang terbobot. Hal ini memanipulasi data yang disajikan melalui beberapa pemrosesan gradien, biasanya *Gradient Descent* dan kemudian menghasilkan *output* untuk jaringan saraf, yang berisi parameter dalam data. Fungsi aktivasi dapat berupa linier atau non-linier tergantung pada fungsi yang merepresentasikannya, dan digunakan untuk mengontrol *output* dari jaringan saraf luar.

2.6.1. *Sigmoid Function*

Sigmoid merupakan fungsi aktivasi non-linier yang digunakan dalam jaringan saraf. Fungsi *Sigmoid* mengubah *range* nilai input x menjadi nilai antara 0 dan 1. Fungsi *Sigmoid* ditunjukkan oleh persamaan 2.17. $S(x)$ akan menghasilkan sebuah kurva dalam rentang 0-1 pada sumbu y. Jika x adalah bilangan 17 positif sangat besar, maka nilai e^{-x} memiliki nilai mendekati 0 yang menghasilkan *output* mendekati 1. Sedangkan jika x adalah bilangan negatif sangat besar, maka nilai e^{-x} memiliki nilai yang besar dan menghasilkan *output* mendekati 0.

$$f(x) = \frac{1}{(1+e^{-x})} \quad (2.17)$$

Keterangan : $fsigmoid(x) =$ fungsi *sigmoid* dari x

e = epsilon

2.6.2. *Rectified Linear Unit (ReLU)*

Fungsi ReLU merupakan salah fungsi aktivasi yang telah terbukti menjadi fungsi yang sering digunakan dalam *Deep Learning*. Hal ini dikarenakan fungsi ReLU menawarkan kinerja dan generalisasi yang lebih baik dalam *Deep Learning* dibandingkan dengan fungsi aktivasi *sigmoid* dan *tanh*. Aktivasi ReLU akan memberikan keluaran 0 ketika $x < 0$ dan merepresentasikan fungsi linier ketika $x \geq 0$. Fungsi aktivasi ReLU melakukan operasi ambang batas untuk setiap 18 elemen input di mana nilai kurang dari nol diatur ke nol sehingga ReLU dinotasikan sebagai persamaan berikut:

$$f(x) = max(0, x) = \begin{cases} x_i, if\ x_i \geq 0 \\ 0, if\ x_i < 0 \end{cases} \quad \text{.....(2.18)}$$

Range dari fungsi aktivasi ReLU adalah [0, ∞], semua input bernilai negatif pada fungsi aktivasi ReLU akan menjadi nilai 0. Fungsi ini memperbaiki nilai input yang kurang dari nol sehingga diubah menjadi nol dan menghilangkan masalah gradien (*vanishing gradient*) yang diamati pada jenis aktivasi sebelumnya.

2.6.3. *Hyperbolic Tangent* Function *(Tanh)*

Fungsi aktivasi *tanh* merupakan salah satu fungsi aktivasi yang digunakan untuk kasus multi klasifikasi. Kelemahan fungsi aktivasi *tanh* adalah tidak menyelesaikan *vanishing gradient* yang umum terjadi pada *sigmoid*. Fungsi aktivasi *tanh* ditunjukkan oleh persamaan 2.19. Jika x merupakan bilangan negatif sangat besar, maka nilai *tanh* akan mendekati -1, ketika nilai x merupakan bilangan positif sangat besar, maka nilai *tanh* mendekati 1.

$$f(x) = \left(\frac{e^x - e^{-x}}{e^x + e^{-x}}\right) \quad \text{.....(2.19)}$$

Keterangan: $ftanh(x)$ = fungsi *tanh* dari x

e = epsilon

2.6.4. *Softmax Function*

Fungi aktivasi *Softmax* adalah jenis dari fungsi aktivasi yang digunakan dalam komputasi saraf. Ini digunakan untuk menghitung distribusi probabilitas dan vektor bilangan real. Fungsi *Softmax* biasanya digunakan sebagai *output* dari model klasifikasi untuk merepresentasikan distribusi kemungkinan terhadap sejumlah kelas Fungsi *Softmax* menghasilkan *output* yang kisaran nilai antara 0 dan 1, dengan jumlah probabilitas sama dengan 1. Fungsi *softmax* dihitung menggunakan persamaan berikut:

$$f(xi) = \frac{exp(x_i)}{\sum_j exp(x_i)} \quad \text{.....(2.20)}$$

Keterangan : x = nilai input lapisan sebelumnya

i, j = indeks unit dan lapisan

Fungsi *softmax* digunakan dalam model multi kelas di mana fungsi ini mengembalikan probabilitas setiap kelas. Fungsi *softmax* sebagian besar muncul di hampir semua lapisan *output* dari arsitektur *Deep Learning*. Perbedaan utama antara fungsi aktivasi *Sigmoid* dan *Softmax* adalah *Sigmoid* digunakan dalam klasifikasi biner sedangkan *Softmax* digunakan untuk tugas klasifikasi *multivariant* atau multi label.

2.7 Stochastic Gradient Descent

Gradient descent adalah suatu teknik yang digunakan untuk optimisasi model *deep learning* dengan cara memperbarui parameter dan secara bertahap menurunkan nilai *error* atau *loss*. Namun, cara ini sangat lambat karena mengharuskan melewati iterasi keseluruhan *dataset* sebelum melakukan pembaruan pada parameter.

Minibatch stochastic gradient descent hanya cukup mengambil beberapa sampel acak, yaitu beberapa contoh dari data pelatihan, untuk menghitung gradient descent dan rata-rata *loss* pada *minibatch* terhadap parameter model. *Minibatch Stochastic Gradient Descent* adalah alat yang

biasa digunakan untuk optimisasi neural network. *Minibatch stochastic gradient descent* mampu menyesuaikan kecepatan konvergensi dan membuat komputasi menjadi lebih efisien. *Minibatch* lebih efisien dan cepat dibandingkan dengan *stochastic gradient descent* [20]

2.8 *Oversampling*

Oversampling adalah salah satu metode yang digunakan untuk menyeimbangkan distribusi data (*imbalanced data*) dengan meningkatkan jumlah data minoritas [21]. Metode ini dapat mempengaruhi hasil perhitungan algoritme pembelajaran mesin, karena di beberapa kasus algoritme tidak mementingkan data kelas minoritas bahkan ada yang mengabaikan data tersebut. Salah satu pendekatan yang dapat dilakukan adalah dengan menggunakan random sampling. Pendekatan ini menduplikasi data minoritas untuk meratakan jumlah keseluruhan data di semua kelas. Data dipilih secara acak sampai distribusi data yang diinginkan tercapai [22]. Metode seperti ini tidak menambahkan fitur baru terhadap *dataset* sehingga cocok diimplementasikan pada data yang besar dan kompleks karena waktu eksekusi yang cepat. Perubahan distribusi data juga hanya diterapkan ke *training dataset*.

2.9 Synthetic Minority Over Sampling Technique (SMOTE)

Synthetic Minority Oversampling Technique (SMOTE) adalah teknik statistika untuk menambahkan jumlah data di kelas minoritas agar seimbang dengan kelas mayoritas [23]. Berbeda dengan teknik random *oversampling*, SMOTE tidak hanya menduplikasi data, melainkan menggunakan algoritma kNN untuk mensintesis data baru berdasarkan data yang telah dimiliki. Cara kerja SMOTE yaitu sebuah data dari kelas minoritas dipilih secara acak, kemudian menggunakan algoritme kNN untuk mencari data tetangga dan menghubungkan kedua data tersebut menggunakan garis [24]. Data sintesis diambil dari kombinasi *convex* kedua data yang telah dipilih.

Nearest neighbor dipilih berdasarkan jarak Euclidian antara kedua data. Misalkan diberikan dua data dengan p dimensi yaitu $X^t = [x_1, x_2, \ldots., x_n]$ dan $Y^t = [y_1, y_2, \ldots., y_n]$ maka jarak Euclidian $d(x, y)$ adalah

$$X_{knn} = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2} \quad \text{.....(2.21)}$$

Secara umum, rumus menentukan data sintetis sebagai berikut:

$$X_{syn} = X_i + (X_{knn} - X_i)\ x\ \delta \quad \text{.....(2.22)}$$

Keterangan : X_{syn} adalah data sintetis hasil dari replikasi

X_i adalah data yang akan direplikasi

X_{knn} adalah data yang memiliki jarak terdekat dari data yang akan direplikasi

δ adalah bilangan random antara 0 dan1

Jika bilangan random mendekati 0, maka data sintetis akan sama dengan data minoritas asal. Jika mendekati 1 maka data sintetis akan sama dengan tetangga terdekat.

SMOTE memiliki beberapa varian dengan proses ekstrapolasi data tidak hanya memanfaatkan algoritme kNN saja, tetapi juga memanfaat algoritme lainnya [3]. Adapun varian tersebut antara lain Borderline SMOTE, K-Means SMOTE, dan SVM SMOTE.

1. Borderline SMOTE

   Metode Borderline SMOTE hanya mengekstrapolasi data yang memiliki data tetangga berasal dari kelas mayoritas dan minoritas. Data tersebut dapat dikatakan berada di area “borderline”. Metode ini mengklasifikan data di kelas minoritas menjadi dua, yaitu *noise* point, dan border point. Pendekatan SMOTE ini tidak mempertimbangkan *noise* point dan hanya menghasilkan data berdasarkan border point

2. K-Means SMOTE

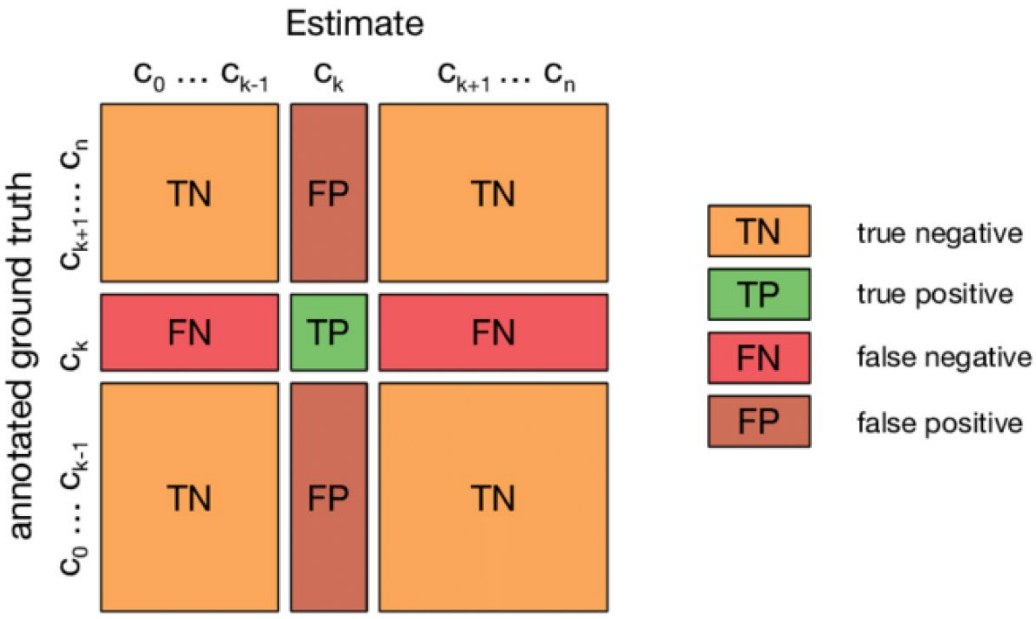
   K-Means SMOTE adalah metode *oversampling imbalanced data* yang membantu proses klasifikasi dengan menghasilkan data kelas minoritas baru secara aman dan di area yang penting sebagai input. Metode ini berusaha menghindari sintesis data *noise* (yang tidak terlalu berguna). Tahapan yang dilakukan antara lain : Melakukan *clustering* terhadap semua data menggunakan algoritme K-Means *Clustering*, Memilih *cluster* yang memiliki jumlah data dari kelas minoritas terbanyak, Membuat data baru berdasarkan sampel yang didapat ke *cluster* yang memiliki data kelas minoritas sedikit

3. SVM SMOTE

   Metode ini menggunakan algoritme SVM untuk mengidentifikasi contoh misklasifikasi di batas keputusan (*decision boundary*). Batas keputusan didefinisikan menggunakan *support vector* dan data yang berada disekitar *support vector* tersebut menjadi fokus dalam pembuatan data sintesis baru. Selain itu metode ini juga lebih mengutamakan area dengan data dari kelas minoritas yang lebih sedikit dan kemudian mengekstrapolasi ke arah batas keputusan.

2.10 *Confussion Matrix*

Confussion matrix adalah informasi yang disimpan dalam bentuk matriks untuk mengetahui performansi model *Machine Learning* maupun *Deep Learning* yang dipakai, dan dapat digunakan sebagai tumpuan dari performansi klasifikasi algoritma yang dipakai pada tahap evaluasi [5]. Tugasnya adalah untuk memprediksi label dari data input menjadi kelas spesifik, yang dikodekan secara numerik. Nilai yang dipakai yaitu *True Positive* (TP), *True Negative* (TN), *False Positive* (FP), dan *False Negative* (FN) [25]. Dapat dilihat pada gambar 2.11 berikut.



Gambar 2.11 Multilabel *Confussion Matrix*

Keterangan:

True Positif (TP) : Nilai Prediksi dari model yang cocok dengan nilai aktual, model yang diprediksi mendapatkan nilai positif, dan nilai aktualnya bernilai positif.

False Negatif (FN) : Nilai Prediksi dari model adalah negatif, sementara nilai aktual adalah positif.

False Positif (FP) : Nilai Prediksi dari model adalah positif sementara nilai aktual nya negatif

True Negatif (TN) : Nilai prediksi dari model cocok dengan nilai aktual, model yang diprediksi mendapatkan nilai negatif, dan nilai aktualnya bernilai negatif.

Perhitungan performansi yang dipakai pada penelitian ini adalah *Accuracy*. *Accuracy* didefinisikan sebagai rasio dari total jumlah nilai yang diprediksi secara benar oleh model dengan jumlah total sampel [25]. Perhitungan akurasi dapat dilakukan dengan persamaan

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN} \quad \text{.....(2.23)}$$

2.11 Studi Literatur

Penelitian ini memiliki keterkaitan dengan penelitian-penelitian sebelumnya, sehingga terdapat hubungan dalam hal kesamaan dan perbedaan dalam objek yang diteliti. Ringkasan penelitian-penelitian terdahulu dapat dilihat pada tabel 2.3 berikut.

No.	Judul	Metode	Hasil
1	Survey Analysis of Machine Learning Methods for Natural Language Processing for MBTI Personality Type Prediction [26]	LSTM	38
2	Deteksi Kepribadian MBTI pada Diskusi Agama Islam [2]	SVM	82
3	Penerapan SMOTE untuk Mengatasi Imbalance Class dalam Klasifikasi Kepribadian MBTI Menggunakan Naive Bayes [3]	Naive Bayes	68
4	Peningkatan Akurasi pada Prediksi Kepribadian MBTI Pengguna Twitter Menggunakan Augmentasi Data [4]	Random Forest	70
5	Prediksi Tipe Kepribadian MBTI Artis K-Pop Berdasarkan Caption Instagram Menggunakan Word2Vec dan Long-Short Term Memory (LSTM) [5]	LSTM	84
6	MBTI Personality Prediction Using Machine Learning and SMOTE for Balancing Data Based on Statement Sentences [7]	LSTM	83