BAB 2

LANDASAN TEORI

2.1 *Text Preprocessing*

Preprocessing digunakan untuk mempersiapkan teks sebelum digunakan dalam pengujian atau pelatihan dengan tujuan untuk mengurangi *noise* yang ada pada data sehingga dapat meningkatkan kinerja *classifier* dan mempercepat proses klasifikasi (Sivakumar dan Gunasundari, 2017). Teknik yang digunakan pada *preprocessing* dapat dilihat pada poin-poin di bawah ini:

2.1.1 *Case Folding*

Case folding merupakan salah satu proses dari *text preprocessing*. Pada proses ini akan dilakukan perubahan pada huruf besar menjadi huruf kecil. Selain itu, proses penghapusan angka dan simbol khusus yang tidak terlalu penting pada klasifikasi seperti tanda seru (!), koma (,), garis miring(/), lebih besar (>), lebih kecil (<) dan sebagainya juga dilakukan (Hermawan dan Ismiati, 2020).

Tabel 2.1 Contoh *Case Folding* (Indraloka dan Santosa, 2017)

Sebelum *Case Folding*	JAKARTA, COVID-19 merupakan virus mematikan.
Sesudah *Case Folding*	jakarta covid19 merupakan virus mematikan

2.1.2 *Tokenizing*

Tokenizing menjelaskan proses mengenai pemecahan teks menjadi kata-kata dengan menggunakan spasi sebagai pemisah yang bertujuan agar setiap kata dapat berdiri sendiri tanpa adanya ikatan dengan kata yang lain (Nata dan Yudiastra, 2017).

Tabel 2.2 Contoh *Tokenizing* (Indraloka dan Santosa, 2017)

Sebelum *Tokenizing*	saya suka makan nasi			
Sesudah *Tokenizing*	saya	suka	makan	nasi

2.1.3 *Stopword Removal*

Pada suatu kalimat biasanya terdapat beberapa kata-kata yang sudah tidak memiliki arti yang relevan seperti “ini”, “itu”, dan sebagainya. Maka dari itu, kata-kata yang sering muncul namun tidak berpengaruh besar tersebut dihapus pada tahap ini untuk meningkatkan kinerja klasifikasi (Indraloka dan Santosa, 2017).

Tabel 2.3 Contoh *Stopword Removal* (Indraloka dan Santosa, 2017)

Sebelum *Stopword Removal*	tidak *bisa* masuk *ke* tempat *ini*
Sesudah *Stopword Removal*	tidak masuk tempat

2.1.4 *Stemming*

Stemming merupakan proses merubah kata menjadi kata dasar dengan cara menghilangkan imbuhan yang ada pada kata tersebut seperti “di-”, “-nya”, dan sebagainya. Tujuan dari *stemming* adalah untuk menghemat waktu dalam melakukan klasifikasi (Violos dkk., 2018).

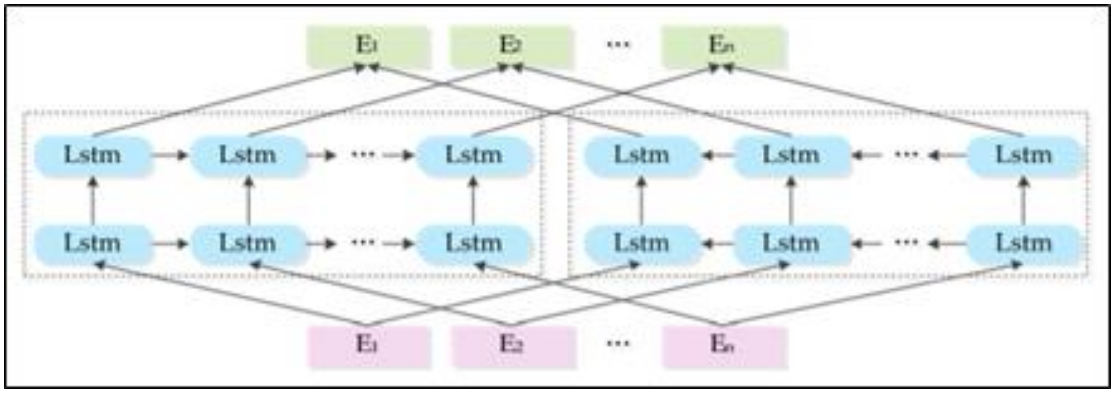
Tabel 2.4 Contoh *Stemming* (Indraloka dan Santosa, 2017)

Sebelum *Stemming*	saya suka memakan nasi
Sesudah *Stemming*	saya suka makan nasi

2.2 *Embedding from Language Models*

Embedding from Language Models (ELMo) merupakan cara baru untuk merepresentasikan kata dalam vektor. Representasi kata asli ELMo dihasilkan dengan mengambil seluruh input ke dalam persamaan untuk menghitung *embeddings* kata. ELMo bekerja berdasarkan pada *bidirectional* LSTM (biLSTM) yang dilatih dari banyak kalimat, dilihat dengan menggunakan dua arah, yaitu maju dan mundur sehingga dapat memberikan makna yang spesifik dan biLSTM merupakan perbaikan dari LSTM yang lemah terhadap dokumen yang panjang (Zhang, Geng dan Chen, 2020).

Pada ELMo terdapat *Recurrent Neural Network* (RNN) dan *Long Short Term Memory* (LSTM) dimana RNN untuk mempersatukan setiap kata dalam teks secara bergantian. RNN dapat menggunakan memori untuk menyimpan urutan teks dan dipercaya dapat menggunakan memori tersebut tanpa batasan namun dalam prakteknya, RNN hanya dapat melihat beberapa saja. LSTM yang ada pada ELMo digunakan untuk mengatur berapa banyak informasi yang perlu diterima, yang tidak digunakan atau yang perlu digunakan kembali (Peters dkk., 2018). Berikut merupakan struktur model ELMo (Zhang, Geng dan Chen, 2020).



Gambar 2.1 Struktur Embedding from Language Models (Zhang, Geng dan Chen, 2020)

Dapat terlihat pada Gambar 2.1, ELMo menggunakan 2 layer yang disatukan dan pada setiap lapisan memiliki 2 *phase*, yang dikenal dengan *forward pass* untuk perhitungan pada kata dan kata sebelumnya dan *backward pass* untuk kata dan kata setelahnya. Berikut merupakan persamaan yang digunakan dalam prediksi biLSTM yang dilatih dengan menggunakan banyak kalimat dapat dilihat pada persamaan 2.1.

$$\sum_{k=1}^{N}(\log p(t_k \mid t_1...,t_{k-1};\Theta_x,\overleftarrow{\Theta}_{LSTM},\Theta_s + \log p(t_k \mid t_{k+1}...,t_N;\Theta_x,\overleftarrow{\Theta}_{LSTM},\Theta_s)) \quad ...(2.1)$$

Kemudian, ELMo akan menggabungkan setiap biLSTM dari k dengan memperkecil semua representasi setiap lapisan dalam R_k yang merepresentasikan k dengan menggunakan persamaan yang tertera pada persamaan 2.2.

$$R_k = \left\{ X_k^{LSTM}, \overrightarrow{h}_{k,j}^{LSTM}, \overleftarrow{h}_{k,j}^{LSTM} \mid j = 1,...,L \right\} = \left\{ h_{k,j}^{LSTM} \mid j = 1,...,L \right\} \quad ...(2.2)$$

Untuk mengukur lapisan atas yang dipilih ELMo serta bobot pada setiap lapisan biLSTM persamaan yang digunakan dapat dilihat pada persamaan 2.3.

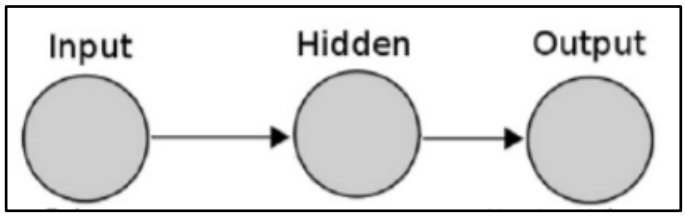
$$ELMo_k^{task} = \gamma^{task} \sum_{j=0}^{L} s_j^{task} h_{k,j}^{LSTM} \quad ...(2.3)$$

Dimana γ^{task} merepresentasikan parameter skalar yang digunakan untuk skala vektor ELMo berdasarkan task dan s_j^{task} merupakan bobot softmax untuk setiap lapisan (Zhang, Geng dan Chen, 2020).

2.3 *Neural Network*

Neural Network atau Jaringan Syaraf Tiruan (JST) merupakan suatu paradigma pengolahan data yang terinspirasi oleh sistem saraf yang ada pada otak manusia, saling terhubung satu dengan yang lain dalam bentuk grafik diarahkan. Struktur dari sistem pengolahan informasi terdiri dari sejumlah besar elemen pemrosesan yang saling berhubungan atau yang dikenal dengan neuron direpresentasikan sebagai node pada suatu bagan yang bekerja bersama-sama untuk menyelesaikan suatu permasalahan tertentu. *Neural Network* memiliki cara kerja yang sama dengan otak manusia dimana perilaku yang berguna harus dipelajari, diakui dan diterapkan antara objek dan *frame* benda di dunia nyata (Freeman dkk., 1992). Lapisan-lapisan pada *Neural Network* terbagi menjadi tiga, yaitu *input layer*, *hidden layer*, dan *output layer* (Sutojo, 2010). Lapisan-lapisan tersebut memiliki tanggung jawab terhadap fungsinya masing-masing

untuk saling melengkapi sistem (Vivian dkk., 2012). Skema dari ketiga lapisan tersebut dapat dilihat pada Gambar 2.2.



Gambar 2.2 Skema Dasar Neural Network (Vivian dkk., 2012)

Multilayer Perceptron (MLP) merupakan pengembangan lebih lanjut dari *perceptron* lapisan tunggal yang tidak memiliki lapisan *hidden layer* didalamnya dan merupakan topologi paling umum dari NN dimana *perceptron* yang saling terhubung membentuk beberapa lapisan dengan satu atau lebih *hidden layer*. Pada MLP terdapat metode yang sudah banyak digunakan yaitu *back-propagation*. Dalam metode ini terdapat empat langkah, yaitu *initialization* untuk menentukan nilai awal bobot dan *threshold* yang dilakukan secara acak namun dalam batasan tertentu, lalu *activation* untuk memberikan masukan dan nilai yang diharapkan, setelah itu *weight training* untuk menghasilkan nilai yang dihasilkan sebenarnya yang akan dibandingkan dengan nilai yang diharapkan sebelumnya, dan langkah selanjutnya adalah *iteration* yang dimana langkah *activation* dan *weight training* akan diulang hingga kondisi yang ditentukan (Negnevistky, 2005).

2.4 Evaluasi Performa

Evaluasi dilakukan dengan tujuan untuk mengetahui nilai performa dari sistem ini akan dilakukan dengan perhitungan *Accuracy*, *Precision*, *Recall* dan F-*Measure*.

Accuracy diukur berdasarkan jumlah kata yang benar diprediksi sebagai entitas dari jumlah kata yang terdapat pada artikel berita. Persamaan *Accuracy* dapat dilihat pada persamaan 2.4.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad \ldots(2.4)$$

Dimana *True Positive* (TP) didapatkan dari perhitungan kata yang benar diprediksi sebagai sebuah entitas, *True Negative* (TN) didapatkan dari perhitungan kata yang telah benar diprediksi bukan sebagai entitas, *False Positive* (FP) didapatkan dari perhitungan kata yang telah salah diprediksi, dan *False Negative* (FN) yang didapatkan dari perhitungan kata entitas yang salah diprediksi.

Precision digunakan untuk mengukur ketepatan hasil *classifier*, yang dapat dihitung dengan TP dibagi dengan total dari penjumlahan TP dengan FP. Persamaan *Precision* dapat dilihat pada persamaan 2.5.

$$Precision = \frac{TP}{(TP + FP)} \quad \ldots(2.5)$$

Recall untuk digunakan mengukur kelengkapan hasil *classifier*, yang dapat diukur dengan perhitungan TP dibagi dengan total penjumlahan TP dan FN. Persamaan *recall* dapat dilihat pada persamaan 2.6.

$$Recall = \frac{TP}{(TP + FN)} \quad \ldots(2.6)$$

F-*Measure* digunakan untuk mengukur rata-rata harmonic dari *precision* dan *recall*. Pengukuran F-*Measure* dapat dilihat pada persamaan 2.7.

$$F1 = 2 . \frac{(Precision\ x\ Recall)}{(Precision + Recall)} \quad \ldots(2.7)$$