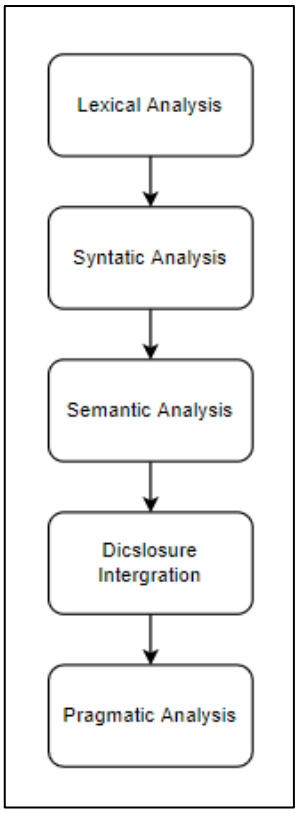
BAB 2

LANDASAN TEORI

2.1. *Natural Language Processing*

Liddy dkk(2011) Teori yang menjelaskan mengenai teknik komputasi untuk menganalisis dan merepresentasi sebuah teks yang terjadi secara natural pada satu atau lebih level dari analisis linguistik yang memiliki tujuan untuk memproses *human-like language* yang dapat digunakan pada sebuah aplikasi. Teks yang terjadi secara natural merupakan kumpulan teks yang dihasilkan melalui sebuah pembicaraan ataupun tulisan. *Natural Language Processing* (NLP) tersebut akan memberikan kemampuan kepada mesin agar dapat mengerti bahasa manusia atau juga disebut dengan *Natural Language Understanding* (NLU). Terdapat berbagai tahapan-tahapan analisis teks menggunakan NLP menurut Nitin Indurkhya dan Fred J Damerau yaitu *tokenization, lexical analysis, syntactic analysis, semantic analysis, discourse integration* dan *pragmatic analysis*. Tahap *tokenization* merupakan tahap awal dan juga krusial. Proses *tokenization* adalah memecah kalimat menjadi kata-kata yang esensial dalam kalimat tersebut dan juga menghapus tanda baca. Tahap *lexical analysis* akan menganalisis dan mengidentifikasi struktur kata dari bahasa pada kalimat yang sesuai dengan koleksi kata yang dimiliki. Selanjutnya adalah tahap *syntactic analysis* yang menganalisis susunan kata-kata pada kalimat sesuai dengan aturan bahasa natural. *Semantic analysis* merupakan identifikasi struktur sintatik dari urutan kata-kata yang

ditentukan berdasarkan arti dari kalimat tersebut. *Discourse intergration* menjelaskan bahwa setiap kalimat memiliki hubungan antar satu dengan lainnya dan memiliki makna yang saling terhubung. *Pragmatic analysis* adalah analisis pada kalimat dengan melakukan interpretasi arti sesungguhnya pada sebuah kalimat. Langkah-langkah dalam menerapkan NLP terdapat pada gambar 2.1



Gambar 2.1 Langkah-Langkah Penerapan NLP

Pada penelitian ini, proses penerapan NLP pada *model machine learning* menggunakan *library* NLTK, BeautifulSoup dan Sastrawi. Ketiga *library* tersebut memiliki fungsi masing-masing yaitu NLTK bertujuan untuk memproses kalimat menjadi pecahan *token-token*. *Library* Sastrawi memiliki tujuan untuk melakukan proses *stemming* pada setiap kata masukan dan menghapus *stopwords* pada paragraf. *Stemming* merupakan proses transformasi sebuah kata menjadi kata asal sebagai contoh pada kata makanan, maka kata asal dari kata tersebut adalah makan. BeautifulSoup memiliki fungsi sebagai *library* yang akan membersihkan paragraf berita yang memiliki *tag* HTML agar tidak diproses, karena *tag* HTML tidak memiliki informasi yang dapat menjelaskan sebuah konteks kalimat.

2.2. *Recurrent Neural Network* (RNN)

RNN adalah variasi dari *artificial neural network* (ANN) yang memiliki *node-node* yang saling terhubung untuk membentuk urutan dari *directed graph*. RNN memiliki *memory* untuk memproses masukan yang berurutan. *Memory* tersebut memberi kemampuan kepada RNN untuk mengurutkan dan memproses data-data masukan yang berurutan seperti teks, suara, video dan lainnya. RNN memiliki siklus yang memberi masukan kepada setiap aktivasi pada jaringan dari masukan sebelumnya untuk mempengaruhi prediksi pada proses waktu tertentu. Nilai aktivasi disimpan pada internal jaringan yang dapat memegang informasi yang bersifat kontekstual dengan jangka waktu yang sementara. Ajao dkk(2018) RNN memiliki efektifitas dalam waktu dan

mengurutkan makna dari sebuah kalimat yang bertujuan dalam memprediksi suatu kategori dengan keterbatasan masalah yang disebut dengan *exploding gradient dan vanishing gradient* yang dapat diatasi dengan bantuan arsitektur *Long Short-Term Memory*. Berikut perhitungan yang diterapkan pada RNN yang terdapat pada Rumus 2.1

$$Y_t = W_{hy} h_t \tag{2.1}$$

Dalam model RNN tersebut Y_t merupakan *output state*, W_{hy} adalah *weight* dari *output state* dan h_t adalah fungsi aktivasi yaitu softmax. Rumus perhitungan fungsi aktivasi terdapat pada rumus 2.2

$$h_t = \text{softmax}\,(W_{hh}\, h_{t-1} + W_{xh} X_t) \tag{2.2}$$

Pada rumus 2.2, W merupakan *weight*, h adalah *single hidden vector*, W_{hh} adalah *weight* pada *hidden state* sebelumnya, W_{xh} adalah *weight* pada *input state* sekarang dan h_t adalah nilai *vector* pada *hidden layer.*

2.3 Long Short-Term Memory (LSTM)

LSTM merupakan satuan pada RNN yang memiliki sel, masukan, keluaran dan *forget gates*. Sel berfungsi untuk menyimpan ingatan-ingatan yang diproses sebelumnya dan *gates* bertujuan untuk mengatur jalur informasi yang keluar dan masuk pada sebuah sel. LSTM memberikan alternatif untuk menyelesaikan masalah pada RNN saat proses *back propagation* yaitu *vanishing/exploding gradient.* Masalah *Exploding gradient* pada RNN dapat diatasi dengan membatasi faktor dengan memangkas gradient yang ada. *Vanishing gradient* tersebut terjadi

saat keluaran yang dihasilkan oleh lemparan masukan memiliki jumlah yang besar pada *hidden layer*. Masalah pada RNN ini disebabkan pada lapisan yang mengeluarkan hasil ditentukan oleh *back propagation* pada setiap nilai awal yang menghasilkan beberapa *weight* oleh karena itu semakin awal memori tersebut dihasilkan maka akan semakin cepat dilupakan oleh layer-layer lainnya. Pada layer yang memiliki perubahan gradient kecil maka layer tersebut tidak melakukan proses pembelajaran lagi dan terjadi pada layer-layer awal. LSTM memberikan kemampuan dengan memberi *gate* pada sebuah layer yang dapat menentukan bahwa data tersebut penting pada suatu runtutan kata. LSTM memiliki berbagai *gate* yang membentuk konstruksi pada layer yaitu :

1. *Forget Gate*

   Gate yang memiliki fungsi untuk menentukan informasi yang seharusnya dibuang atau tetap dipertahankan. Sari dkk(2020) Informasi dari masukan sebelumnnya dijadikan masukan pada kondisi sekarang melalui fungsi sigmoid untuk membuat keputusan. Perhitungan matematis *forget gate* terdapat pada rumus 2.3

$$f_t = \sigma\left(W_i.[h_{t-1}, x_t] + b_f\right) \qquad (2.3)$$

   W_i pada rumus 2.3 merupakan matriks yang terdiri dari *weight*, σ adalah fungsi sigmoid, x_t merupakan masukan untuk sel LSTM yang berasal dari *word embedding*, h_t adalah nilai *vector* pada *hidden layer* dan b_f merupakan nilai *bias* pada model LSTM

2. *Input Gate*

Gerbang yang bertugas untuk melakukan pembaruan pada *cell state*. Diawali dengan menerima masukan dari *hidden layer* sebelumnya dan menjadi masukan bagi fungsi sigmoid. Setelah nilai tersebut bertransformasi menjadi 0 sampai 1 dimana 0 berarti tidak penting dan 1 berarti penting, setelah itu hasil dari sigmoid tersebut akan diproses dengan mengalikan nilai keluaran sigmoid dan tanh. Hasil akhir dari perkalian tersebut yang merupakan keluaran sigmoid akan menjadi penentu apakah informasi tersebut dibuang atau tetap dipertahankan. Perhitungan matemtis untuk lapisan sigmoid yang sama digunakan pada *forget gate* terdapat pada rumus 2.4

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i) \quad (2.4)$$

W_i pada rumus 2.4 merupakan matriks yang terdiri dari *weight*, σ adalah fungsi sigmoid, x_t merupakan masukan untuk sel LSTM yang berasal dari *word embedding*, h_t adalah nilai *vector* pada *hidden layer* dan b_i merupakan nilai *bias* pada model LSTM. Selanjutnya lapisan tanh akan membuat vektor dari kandidat baru C_t dengan perhitungan matematis terdapat pada rumus 2.5

$$C_t = tanh(W_c.[h_{t-1}, x_t] + b_c) \quad (2.5)$$

Perbedaan pada *gate* lainnya adalah terdapat 2 lapisan dalam *input gate* yang dimana hasil perhitungan lapisan sigmoid memiliki pengaruh terhadap *cell state*.

3. *Output Gate*

Gate yang menentukan *hidden state* selanjutnya. *Hidden state* mengandung informasi dari masukan sebelumnnya yang memiliki fungsi untuk melakukan prediksi melalui proses perhitungan pada fungsi sigmoid. Selanjutnya diberikan nilai baru pada cell state kepada fungsi tanh dan berakhir pada mengalikan fungsi tanh dengan keluaran fungsi sigmoid yang telah diproses dan menghasilkan *hidden state* untuk langkah selanjutnya pada *neural network*. Rumus untuk menghasilkan nilai *hidden state* terdapat pada rumus 2.6

$$h_t = \sigma(W_i.[h_{t-1}, x_t] + b_o) * \tanh(C_t) \qquad (2.6)$$

W_i pada rumus 2.6 merupakan matriks yang terdiri dari *weight*, σ adalah fungsi sigmoid, x_t merupakan masukan untuk sel LSTM yang berasal dari *word embedding*, h_t adalah nilai *vector* pada *hidden layer* dan b_o merupakan nilai *bias* pada model LSTM. Hasil dari lapisan sigmoid akan dikali dengan fungsi tanh pada *Cell State*.

4. *Cell State*

Nilai pada *cell state* didapat dari perkalian pada *forget vector*. Sari dkk(2020) Setelah itu keluaran pada *input gate* merupakan *cell state*

baru pada neural network untuk model. Perhitungan matematis *cell state* terdapat pada rumus 2.7

$$C_t = f_t * C_{t-1} + i_t * \tanh(W_c . [h_{t-1}, x_t] + b_c \quad (2.7)$$

f_t pada rumus 2.7 adalah nilai dari *forget gate*, C_{t-1} merupakan nilai *cell state* pada situasi sebelumnya dan i_t adalah nilai *input gate*. W_c adalah matriks dari weight, h_t adalah nilai *vector* pada *hidden layer*, x_t merupakan masukan untuk sel LSTM yang berasal dari *word embedding* dan b_c merupakan nilai *bias* pada model LSTM

2.4. *Softmax Activation*

Merupakan fungsi matematika yang mengubah nilai vektor menjadi vektor probabilitas. Setiap vektor probabilitas memiliki nilai yang proporsional untuk rentang pada setiap vektor. Fungsi softmax ini umum digunakan pada model pembelajaran mesin seperti *neural network* dengan menormalisasi keluaran dan mengkonversi menjadi *weighted sum values*. Setiap nilai dari keluaran fungsi softmax merupakan prediksi dari kemungkinan sebuah fitur masuk dalam sebuah kelas yang tersedia. Brownlee(2020) Fungsi aktivasi lain seperti linear dan sigmoid tidak cocok dalam memproses klasifikasi dengan banyak kelas dan softmax merupakan aktivasi yang dapat memberi hasil yang lebih akurat. Rumus matematis dalam menghitung probabilitas pada softmax *activation* terdapat pada rumus 2.8

$$probability = \frac{\exp(value)}{sum\ v\ in\ list \exp(v)} \quad (2.8)$$

Berikut simulasi perhitungan probabilitas menggunakan softmax *activation*, misalnya dalam sebuah *list* terdapat tiga nilai yaitu [1,4,7] dan nilai 1 pada *list* mau diubah menjadi probabilitas, maka perhitungannya sebagai berikut :

P(1)= exp(1) / (exp(1) + exp(4) + exp(7))

P(1)= 2.71828182846 / (2.71828182846 + 54.5981500331 + 1096.63315843)

P(1) = 2.71828182846 / 1,153.94959029156

P(1) = 0.00235563308079444927829435804767

P(4) = exp(4) / (exp(1) + exp(4) + exp(7))

P(4) = 54.5981500331 / 1,153.94959029156

P(4) = 0.047314155221724273851125568 4193

P(7) = exp(7) / (exp(1) + exp(4) + exp(7))

P(7) = 1096.63315843 / 1,153.94959029156

P(7) = 0.95033021169748127687058007353303

P(7) memiliki nilai paling besar dibanding yang lain dan menjadikan P(7) sebagai hasil dari fungsi softmax *activation.*

2.5. FastText

FastText merupakan pre trained model untuk melakukan proses *word embeddings* yang diciptakan oleh facebook pada tahun 2016. FastText melakukan pemetaan pada teks yang akan diproses sebelum digunakan oleh model tertentu dengan mengubah setiap kata masukan menjadi vektor sesuai *pre trained model*

FastText, karena sebuah model tidak dapat menerima masukan berupa karakter ataupun kata. Tujuan utama dari proses ini adalah memberikan pertimbangan mengenai struktur dari sebuah kata yang berguna untuk merepresentasi berbagai kata yang memiliki bentuk morfologi yang berbeda dan dapat dimengerti arti dari setiap kata tersebut. Santos dkk(2017) FastText memiliki kemampuan yang sungguh baik untuk sebuah kata yang memiliki bentuk morfologi yang beraneka dan dengan kemampuan tersebut maka setiap bentuk morfologi tersebut dipelajari sebagai bentuk kata yang berbeda-beda. FastText memecah setiap kata pada dataset menjadi beberapa *n-grams* seperti pada kata 'India'. *Tri-grams* pada kata 'india' yaitu 'ind','ndi','di'. Jumlah dari seluruh n-grams tersebut akan menjadi vektor *word embedding* pada kata tersebut. Dengan metode tersebut maka proses *embedding* akan memiliki kemampuan untuk mengerti sebuah kata dengan berbagai variasi namun memiliki arti yang sama. Setelah *n-grams* pada kata telah terbentuk maka, model *skip-gram* telah dilatih untuk dipelajari oleh *embedding layer*. Fastext memiliki keunggulan yaitu dapat merepresentasi sebuah kata yang langka dengan tepat karena nilai n-grams kata tersebut akan mirip dengan n-grams dengan kata yang ada.

2.6 *Confusion Matrix*

Narkhede(2018)*Confusion Matrix* adalah sebuah alat pengukuran untuk mengevaluasi performa dari *machine learning* untuk melakukan klasifikasi. Ping Shung(2018) dalam menentukan nilai dari tiga indikator tersebut terdapat beberapa rumus perhitungan yaitu pada rumus 2.9 merupakan rumus *accuracy*. Rumus 2.10 merupakan perhitungan *precision*. Rumus 2.11 merupakan perhitungan *recall* dan rumus 2.12 untuk perhitungan *f1-score*.

$$Accuracy = \frac{TP+TN}{TP+FP+FN+TN} \quad (2.9)$$

$$Precision = \frac{TP}{TP+FP} \quad (2.10)$$

$$Recall = \frac{TP}{TP+FN} \quad (2.11)$$

$$F1 - Score = 2x \frac{Precision\ x\ Recall}{Precision+Recall} \quad (2.12)$$

TP adalah *True Positive* yang memiliki arti bahwa hasil prediksi kategori sama dengan kategori sesungguhnya. TN adalah *True Negative* yang berarti bahwa hasil prediksi kategori diprediksi dengan benar bahwa suatu berita bukan kategori tersebut, contohnya jika *model* memprediksi suatu berita dengan kategori tekno, tetapi hasil prediksi adalah bisnis dan benar bahwa berita tersebut adalah bisnis dan bukan tekno, maka itu disebut *true negative*. FP adalah *False Positive* yang memiliki arti bahwa hasil prediksi kategori salah memprediksi kategori dengan nilai aktual, contohnya saat *model* memprediksi bahwa berita ini adalah tekno tetapi kategori aktualnya adalah bisnis. FN adalah *False Negative* berarti *model* memprediksi sebuah kategori salah dan hasil prediksi tersebut salah. *Accuracy* merupakan ketepatan prediksi yang bernilai *true*. *Precision* merupakan pengujian untuk mengidentifikasi kelas positif yang diprediksi dengan tepat. *Recall* menghitung dari banyaknya *model* memprediksi hasil prediksi yang positif dengan nilai aktual yang positif. *F1-score* bertujuan untuk melihat apakah *model* memiliki distribusi kelas yang tidak merata dan menampilkan keseimbangan dari *precision* dan *recall.*