TUGAS AKHIR – KS141501

NORMALISASI TEKS BAHASA INDONESIA PADA MEDIA SOSIAL BERDASARKAN FASTTEXT EMBEDDINGS

BAHASA INDONESIA TEXT NORMALIZATION IN SOCIAL MEDIA BASED ON FASTTEXT EMBEDDINGS

AHMAD ARIF SAMUDRO
NRP 0521 14 4000 0118

Dosen Pembimbing
1. Renny Pradina K., S.T., M.T., SCJP

Departemen Sistem Informasi
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2019

Halaman ini sengaja dikosongkan

TUGAS AKHIR – KS141501

NORMALISASI TEKS BAHASA INDONESIA BERDASARKAN FASTTEXT EMBEDDINGS

AHMAD ARIF SAMUDRO
NRP 05211440000118

Dosen Pembimbing
1. Renny Pradina K., S.T., M.T., SCJP

Departemen Sistem Informasi
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2019

Halaman ini sengaja dikosongkan

TUGAS AKHIR – KS141501

BAHASA INDONESIA TEXT NORMALIZATION IN SOCIAL MEDIA BASED ON FASTTEXT EMBEDDINGS

AHMAD ARIF SAMUDRO
NRP 05211440000118

Supervisor
1. Renny Pradina K., S.T., M.T., SCJP

Departement of Information Systems
Faculty of Information Technology and Communication
Institut Teknologi Sepuluh Nopember
Surabaya 2019

Halaman ini sengaja dikosongkan

LEMBAR PENGESAHAN

NORMALISASI TEKS BAHASA INDONESIA PADA MEDIA SOSIAL BERDASARKAN FASTTEXT EMBEDDINGS

TUGAS AKHIR

Disusun Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Departemen Sistem Informasi
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh:

AHMAD ARIF SAMUDRO
NRP. 0521 14 4000 0118

Surabaya, 17 Januari 2019

KEPALA
DEPARTEMEN SISTEM INFORMASI

Mahendrawathi ER, ST, M.Sc, Ph.D
NIP 19761011 200604 2 001

Halaman ini sengaja dikosongkan

LEMBAR PERSETUJUAN

NORMALISASI TEKS BAHASA INDONESIA PADA MEDIA SOSIAL BERDASARKAN FASTTEXT EMBEDDINGS

Disusun Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Departemen Sistem Informasi
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh:

AHMAD ARIF SAMUDRO
NRP. 0521 14 4000 0118

Disetujui Tim Penguji: Tanggal Ujian: 17 Januari 2019
Periode Wisuda: Maret 2019

Renny Pradina K., S.T., M.T., SCJP	**(Pembimbing I)**
Faizal Johan Atletiko, S.Kom., M.T	**(Penguji I)**
Radityo Prasetianto Wibowo, S.Kom, M.Kom	**(Penguji II)**

Halaman ini sengaja dikosongkan

NORMALISASI TEKS BAHASA INDONESIA PADA MEDIA SOSIAL BERDASARKAN FASTTEXT EMBEDDINGS

Nama Mahasiswa : Ahmad Arif Samudro

NRP : 05211440000118

Departemen : Sistem Informasi FTIK-ITS

Dosen Pembimbing :

1. Renny Pradina K., S.T., M.T., SCJP

ABSTRAK

Menurut survei Asosiasi Penyelenggara Jasa Internet Indonesia (APJII) pengguna internet di Indonesia pada tahun 2017 adalah 143,76 jiwa, dan 87,13 persen diantara mereka menggunakannya untuk kebutuhan media sosial sebagai wadah bertukar informasi[1]. Namun seiring berjalannya waktu, komunikasi tertulis pada media sosial cenderung menggunakan ejaan dan tata bahasa yang tidak formal, dengan alasan agar komunikasi terjadi lebih natural, santai, dan terasa akrab[2]. Hal ini tentunya akan sedikit mengganggu penyebaran informasi karena banyaknya perbedaan ejaan kata pada media sosial Twitter dan Facebook. Banyaknya informasi yang simpang siur di platform media sosial merupakan suatu sumber data yang sangat berharga bagi para pegiat NLP. Namun masih banyak kendala yang ditemui dalam praktik NLP itu sendiri, dimana komputer sulit memahami bahasa manusia yang tidak baku baik secara ejaan maupun semantiknya. Dan untuk meminimalisir masalah tersebut, diperlukan serangkaian aktivitas untuk menormalisasi teks, diantaranya dengan model word embedding FastText.

Penggabungan model word embedding dengan algoritma Levensthein's distance, dan Jaro-Winkler distance dapat menghasilkan sistem normalisasi yang cukup baik.

Hasil terbaik dari training model yang didapatkan adalah model FastText ke-3 dengan akurasi pengujian awal sebesar 45,6%. Dimana parameter penting yang perlu diperhatikan adalah learning algorithm dan context window. Pengujian data sampel pada model ini menghasilkan akurasi terbaik sebesar 84,65% pada threshold 55% dan 60%.

Kata Kunci: ***Natural Language Processing, Word Embedding, Word2Vec, Fasttext, Social Media.***

BAHASA INDONESIA TEXT NORMALIZATION IN SOCIAL MEDIA BASED ON FASTTEXT EMBEDDINGS

Student Name : **Ahmad Arif Samudro**

NRP : **05211440000118**

Department : **Information Systems FTIK-ITS**

Supervisors :

1. Renny Pradina K., S.T., M.T., SCJP

ABSTRACT

According to a survey of the Indonesian Internet Service Providers (APJII) internet users in Indonesia in 2017 were 143.76 people, and 87.13 percent of them used it for social media needs as a place to exchange information [1]. But over time, written communication on social media tends to use informal spelling and grammar, arguing that communication occurs more naturally, relaxed, and feels familiar [2]. This of course will slightly disrupt the spread of information because of the many differences in the spelling of words on Twitter and Facebook social media. The amount of confusing information on social media platforms is a very valuable data source for NLP activists. But there are still many obstacles encountered in the practice of NLP itself, where computers find it difficult to understand human language that is not standard both in spelling and semantics. And to minimize these problems, a series of activities is needed to normalize text, including the word embedding model FastText.

Combining the word embedding model with the Levensthein's distance algorithm, and Jaro-Winkler distance can produce a fairly good text normalization system.

The best results from the training model obtained were the 3rd FastText model with initial testing accuracy of 45.6%. Where the important parameters to consider are learning algorithm and context window. The testing of sample data in this model produces the best accuracy of 84.65% on the threshold of 55% and 60%.

Keywords: ***Natural Language Processing, Word Embedding, Word2Vec, Fasttext, Social Media.***

KATA PENGANTAR

Puji syukur kepada Allah SWT yang telah melimpahkan rahmat dan anugerah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir dengan judul "Normalisasi Teks Bahasa Indonesia pada Media Sosial Berdasarkan *FastText Embeddings*" sebagai salah satu syarat kelulusan pada Departemen Sistem Informasi, Fakultas Teknologi Informasi dan Komunikasi, Institut Teknologi Sepuluh Nopember dengan tepat waktu. Semoga apa yang tertulis dalam buku Tugas Akhir ini dapat bermanfaat kepada para pembacanya, dan dapat memberikan kontribusi dalam perkembangan ilmu pengetahuan khususnya bagi bangsa Indonesia.

Dalam penyusunan Tugas Akhir ini tentunya sangat banyak bantuan yang penulis terima dari berbagai pihak baik dalam bentuk doa, semangat, arahan, kritik, saran, dan berbagai bantuan lainnya. tanpa mengurangi rasa hormat penulis secara khusus ingin menyampaikan ucapan terima kasih kepada:

1. Ibu Siti Suparni dan Bapak Sugeng Samudro, kedua orangtua penulis yang tidak henti-hentinya mengirimkan doa, memberi semangat, dorongan, dan bantuan hingga penulis dapat berada pada titik ini.
2. Ibu Renny Pradina K., S.T, M.T, SCJP selaku dosen pembimbing yang telah membimbing dan membantu penulis selama pengerjaan tugas akhir ini.
3. Bapak Faizal Johan Atletiko, S.Kom., M.T dan Bapak Radityo Prasetianto Wibowo, S.Kom, M.Kom selaku dosen penguji yang telah memberikan masukan-masukan guna menyempurnakan Tugas Akhir ini.
4. Ibu Mahendrawati ER, ST, M.Sc, Ph.D selaku dosen wali penulis yang selalu memotivasi dan memberikan bantuan moral selama masa kuliah penulis.
5. Seluruh Dosen Departemen Sistem Informasi ITS yang telah memberikan ilmu pengetahuan yang bermanfaat dan pengalaman yang berharga bagi penulis.

6. Nadhif dan Hendro, yang telah banyak membantu dan menemani penulis dalam proses pengerjaan tugas akhir ini.
7. Keluarga besar E-Home yang telah menemani kehidupan kampus penulis selama tiga tahun. Memberikan warna pada masa kuliah dengan berbagai tipe mahasiswa yang dapat ditemui.
8. Member UKM Cinta Lab yang menjadi teman mengerjakan, bertukar pikiran, dan berbagi keluh kesah selama pengerjaan tugas akhir.
9. Serta seluruh pihak-pihak lain yang tidak dapat disebutkan satu per satu yang telah banyak membantu penulis selama perkuliahan hingga dapat menyelesaikan tugas akhir ini.

Penulis sadar bahwa Tugas Akhir ini masih jauh dari kata sempurna, sehingga saran dan kritik yang membangun dari pembaca merupakan *feedback* yang berarti untuk perbaikan ke depan. Semoga Tugas Akhir ini dapat bermanfaat bagi perkembangan ilmu pengetahuan dan semua pihak.

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR KODE

BAB I
PENDAHULUAN

Bab ini akan menguraikan gambaran umum penelitian meliputi latar belakang masalah, perumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, dan relevansi terhadap pengerjaan tugas akhir. Tujuan dan manfaat dilakukannya penelitian diharapkan dapat memberikan gambaran awal mengenai penelitian.

1.1 Latar Belakang

Internet, terutama pada sektor media sosial, kini seolah-olah merupakan kebutuhan sehari-hari bagi masyarakat Indonesia. Survey Asosiasi Penyelenggara Jasa Internet Indonesia pada tahun 2017 menunjukkan pengguna internet di Indonesia sejumlah 143,76 jiwa dari total populasi penduduk indonesia 262 juta jiwa. Ini menunjukkan peningkatan sebanyak 10,5 juta jiwa dibanding tahun sebelumnya yaitu 132,7 juta jiwa, dengan 87,13% diantaranya menggunakan internet untuk kebutuhan media sosial[1].

Media sosial merupakan salah satu sarana komunikasi antar pengguna internet, terutama pada situs seperti *Facebook* dan *Twitter*. Namun seiring berjalannya waktu, komunikasi tertulis pada media sosial cenderung menggunakan ejaan dan tata bahasa yang tidak formal, dengan alasan agar komunikasi terjadi lebih natural, santai, dan terasa akrab[2]. Hal ini tentunya akan sedikit mengganggu penyebaran informasi karena banyaknya perbedaan ejaan kata pada media sosial *Twitter* dan *Facebook*.

Kendati demikian, peristiwa penggunaan tata bahasa dan ejaan yang tidak formal ini dapat menjadi keuntungan bagi kalangan akademisi. Banyak diantaranya yang menggunakan peristiwa ini sebagai bahan penelitian di bidang pemrosesan data teks. Data tersebut akan diproses dengan suatu metode yang disebut dengan *Natural Language Processing* (NLP). NLP adalah suatu proses komputasi yang digunakan pada komputer agar dapat

menganalisa, memahami, dan memperoleh makna dari bahasa manusia yang digunakan dalam kehidupan sehari-hari.

Pada praktiknya, NLP memiliki banyak kendala. Banyak dialami kesulitan misalnya dalam penandaan kelas kata, segmentasi teks, identifikasi konteks kalimat yang memiliki penuturan berbeda, disambiguitas makna kata, ambiguitas sintaksis, dan masukan yang tidak sempurna atau tidak teratur[3]. Jika tidak memperhatikan faktor-faktor di atas, maka pada tahap selanjutnya representasi kata akan menjadi tidak akurat, yang akan berujung pada tidak efektifnya proses normalisasi kata.

Untuk meminimalisir terjadinya kesalahan tersebut, perlu dilakukan tahapan-tahapan tambahan seperti pra-proses data tersebut sebelum proses data dimulai. Namun dengan banyaknya ragam variasi kata dalam bentuk tidak baku untuk suatu konteks yang sama, hal ini akan sulit dilakukan. Semisal untuk kata “semangat”, banyak variasi tidak baku lainnya yang sering digunakan dalam kehidupan sehari-hari seperti “mangat”, “mangats”, “cemungudh”, dan lain sebagainya. Jika tidak dilakukan pra-proses data, maka bisa jadi setiap kata tersebut akan direpresentasikan sebagai kata yang berbeda. Atas dasar inilah penulis mengajukan penelitian mengenai normalisasi teks dengan model *FastText*, yang nantinya akan dibandingkan dengan normalisasi menggunakan model *Word2Vec*. Dengan model *FastText*, pada dasarnya kita akan mendapatkan rekomendasi kata-kata atau label yang memiliki probabilitas kedekatan paling besar dengan input yang diberikan.

Diharapkan dengan adanya penelitian ini, penggunaan NLP untuk normalisasi teks tidak baku terutama dalam Bahasa Indonesia dapat menjadi lebih efektif lagi.

1.2 Perumusan Masalah

Berdasarkan latar belakang diatas, maka rumusan masalah yang akan diteliti pada tugas akhir ini adalah:

1. Bagaimana cara mengakuisisi data posting dari situs *Twitter?*
2. Apa saja pra-proses data yang diperlukan untuk memaksimalkan efektifitas pemrosesan data?
3. Bagaimana pembuatan model FastText dan Word2Vec untuk data yang telah diakuisisi?
4. Bagaimana perbandingan model FastText dengan Word2Vec?
5. Model mana yang lebih baik dalam merepresentasikan kata?

1.3 Batasan Masalah

Batasan pemasalahan dalam pengerjaan tugas akhir ini adalah :

1. Studi kasus yang digunakan pada penelitian ini adalah post Berbahasa Indonesia pada situs *Twitter*.
2. Data yang digunakan pada pengerjaan tugas akhir ini adalah data yang diakuisisi pada pengerjaan tugas akhir ini, yaitu posting dari akun *Twitter*, serta data yang telah diakuisisi pada penelitian sebelumnya.
3. Penelitian ini adalah eksperimen implementasi *FastText* dan *Word2Vec* menggunakan data *post* dari media sosial *Twitter*.
4. Proses pembenaran kata belum memperhatikan waktu komputasi.

1.4 Tujuan

Berdasarkan hasil perumusan masalah dan batasan masalah yang telah disebutkan sebelumnya, maka tujuan yang ingin dicapai dari tugas akhir ini adalah untuk membandingkan model mana yang lebih efisien dalam merepresentasikan kata diantara kedua jenis model *word embedding* yang akan dibuat, dan

mencari konfigurasi terbaik dari model tersebut untuk mendapatkan hasil yang maksimal.

1.5 Manfaat

Manfaat yang diharapkan dapat diperoleh dari tugas akhir ini adalah:

1. Memberikan *insight* terkait pemilihan model *word embedding* dan metode *training*-nya.
2. Menyediakan model yang dapat menjadi acuan untuk penelitian-penelitian selanjutnya.
3. Menyediakan dataset yang dapat digunakan sebagai bahan data *training* pada penelitian-penelitian selanjutnya.

1.6 Relevansi

Tugas akhir ini berkaitan dengan mata kuliah Sistem Cerdas, Analisa dan Desain Perangkat Lunak, Konstruksi dan Pengembangan Perangkat Lunak, dan Penggalian Data dan Analitika Bisnis.

BAB II
TINJAUAN PUSTAKA

Bab ini akan menjelaskan mengenai penelitian sebelumnya dan dasar teori yang dijadikan acuan atau landasan dalam pengerjaan tugas akhir ini.

2.1 Penelitian Sebelumnya

Pada subbab ini dijelaskan tentang referensi penelitian yang berkaitan dengan tugas akhir. Pada bagian ini memaparkan acuan penelitian sebelumnya.

1. *Efficient Estimation of Word Representation in Vector Space*, oleh Tomas Mikolov dkk pada tahun 2013[4] Menunjukkan bahwa sangat memungkinkan untuk melatih vektor kata berkualitas tinggi dengan arsitektur model yang cukup sederhana, yaitu *Continuous Bag-of-Words* dan *Skip-gram* dibandingkan dengan model *neural network* yang sedang populer pada saat itu. Dan dengan kerumitan komputasi yang jauh lebih kecil, maka memungkinkan juga untuk mengomputasi vektor kata yang *high-dimensional* dari dataset yang sangat besar dengan akurat.
2. *Distributed Representations of Words and Phrases and their Compositionality,* oleh Tomas Mikolov dkk pada tahun 2013[5] merupakan beberapa tambahan yang cukup signifikan terutama untuk model *Continuous Skip-gram* yang kemudian mampu memiliki representasi vektor yang lebih berkualitas dan juga meningkatkan kecepatan training dataset. Penelitian ini juga mengenalkan pendekatan *hierarchical softmax* dan *negative sampling.* Lalu penulis menyatakan bahwa pemilihan algoritma untuk training dan *hyper-parameter*-nya sangat bergantung pada tugas yang akan diberikan. Karena solusi yang optimal akan berbeda-beda untuk tiap masalah yang berbeda pula.
3. *From Word Embeddings to Item Recommendation* oleh Makbule Gulcin Ozsoy pada tahun 2016. Penulis berhasil mengimplementasikan Skip-gram dan CBOW dari

Word2Vec untuk membuat rekomendasi *check-in* berikutnya pada *Location-based Social Network* (LSBN). Penulis menggabungkan teknik dari Word2Vec dengan metode rekomendasi yang populer, metode-metode berbasis *content-based* dan *collaborative filtering*. Hasilnya menunjukkan bahwa menggunakan metode dari NLP merupakan cara yang efektif, dan penggunaan teknik dari Word2Vec sangat menjanjikan dalam membuat rekomendasi.

4. *Bag of Tricks for Efficient Text Classification* oleh Armand Joulin dkk pada tahun 2016[6]. Penulis dan tim memperkenalkan *FastText*, suatu klasifier teks cepat yang terinspirasi dari *Word2Vec.* Metode ini dapat dilatih dengan lebih dari 1 miliar kata dalam kurang dari 10 menit menggunakan CPU *multicore* standar, dan dapat mengklasifikasi 500 ribu kalimat dari 312 ribu class dalam waktu kurang dari satu menit. Hal ini dapat dicapai karena metode pelatihannya menggunakan metode *hierarchical softmax* dan menggunakan fitur tambahan berupa *bag of N-grams* untuk menangkap informasi mengenai urutan kata-kata yang mendekati input.
5. *Normalisasi Teks Media Sosial Menggunakan Word2Vec, Levenshtein Distance, dan Jaro-Winkler Distance* oleh Stezar Priansya pada tahun 2017[7]. Penulis menemukan beberapa hal, diantaranya dengan berbagai konfigurasi training yang berbeda-beda, ditemukan model terbaik dengan parameter*learning algorithm* CBOW, iterasi sebanyak 2, 5 *minimum word frequency*, context window sebanyak 5, epoch sebanyak 10, layer size sebesar 500 dimensi, dan menggunakan *hierarchical softmax*. Akurasi yang dimiliki oleh model dengan konfigurasi ini adalah sebesar 25% ketika diujikan terhadap 100 kata non-kamus.

Efficient Estimation of Word Representation in Vector Space, oleh Tomas Mikolov dkk pada tahun 2013[4] Menunjukkan bahwa sangat memungkinkan untuk melatih vektor kata berkualitas tinggi dengan arsitektur model yang cukup

sederhana, yaitu *Continuous Bag-of-Words* dan *Skip-gram* dibandingkan dengan model *neural network* yang sedang populer pada saat itu. Dan dengan kerumitan komputasi yang jauh lebih kecil, maka memungkinkan juga untuk mengomputasi vektor kata yang *high-dimensional* dari dataset yang sangat besar dengan akurat.

6. *Distributed Representations of Words and Phrases and their Compositionality,* oleh Tomas Mikolov dkk pada tahun 2013[5] merupakan beberapa tambahan yang cukup signifikan terutama untuk model *Continuous Skip-gram* yang kemudian mampu memiliki representasi vektor yang lebih berkualitas dan juga meningkatkan kecepatan training dataset. Penelitian ini juga mengenalkan pendekatan *hierarchical softmax* dan *negative sampling.* Lalu penulis menyatakan bahwa pemilihan algoritma untuk training dan *hyper-parameter*-nya sangat bergantung pada tugas yang akan diberikan. Karena solusi yang optimal akan berbeda-beda untuk tiap masalah yang berbeda pula.
7. *From Word Embeddings to Item Recommendation* oleh Makbule Gulcin Ozsoy pada tahun 2016. Penulis berhasil mengimplementasikan Skip-gram dan CBOW dari Word2Vec untuk membuat rekomendasi *check-in* berikutnya pada *Location-based Social Network* (LSBN). Penulis menggabungkan teknik dari Word2Vec dengan metode rekomendasi yang populer, metode-metode berbasis *content-based* dan *collaborative filtering*. Hasilnya menunjukkan bahwa menggunakan metode dari NLP merupakan cara yang efektif, dan penggunaan teknik dari Word2Vec sangat menjanjikan dalam membuat rekomendasi.
8. *Bag of Tricks for Efficient Text Classification* oleh Armand Joulin dkk pada tahun 2016[6]. Penulis dan tim memperkenalkan *FastText,* suatu klasifier teks cepat yang terinspirasi dari *Word2Vec*. Metode ini dapat dilatih dengan lebih dari 1 miliar kata dalam kurang dari 10 menit menggunakan CPU *multicore* standar, dan dapat

mengklasifikasi 500 ribu kalimat dari 312 ribu class dalam waktu kurang dari satu menit. Hal ini dapat dicapai karena metode pelatihannya menggunakan metode *hierarchical softmax* dan menggunakan fitur tambahan berupa *bag of N-grams* untuk menangkap informasi mengenai urutan kata-kata yang mendekati input.

9. *Normalisasi Teks Media Sosial Menggunakan Word2Vec, Levenshtein Distance, dan Jaro-Winkler Distance* oleh Stezar Priansya pada tahun 2017[7]. Penulis menemukan beberapa hal, diantaranya dengan berbagai konfigurasi training yang berbeda-beda, ditemukan model terbaik dengan parameter*learning algorithm* CBOW, iterasi sebanyak 2, 5 *minimum word frequency*, context window sebanyak 5, epoch sebanyak 10, layer size sebesar 500 dimensi, dan menggunakan *hierarchical softmax*. Akurasi yang dimiliki oleh model dengan konfigurasi ini adalah sebesar 25% ketika diujikan terhadap 100 kata non-kamus.

2.2 Dasar Teori

Berisi teori-teori yang mendukung serta berkaitan dengan tugas akhir yang sedang dikerjakan.

2.2.1. *Natural Language Processing*

Natural Language Processing (NLP) adalah suatu proses komputasi yang digunakan pada komputer agar dapat menganalisa, memahami, dan memperoleh makna dari bahasa manusia yang digunakan dalam kehidupan sehari-hari. Proses komputasi ini digambarkan sebagai suatu rangkaian kata yang memiliki aturan-aturan tertentu. Dalam melakukan NLP, terdapat beberapa kendala, kendala yang paling umum dialami adalah terjadinya ambiguitas, karena bahkan pada manusia, analisa makna dan konteks dalam suatu kalimat sangat dipengaruhi oleh pemahaman dan pengetahuan masing-masing individu. Kendala lain dalam proses ini adalah varian kosa kata yang semakin besar dan luas. Seiring berjalannya waktu, bahasa akan mengalami perkembangan, baik itu berupa penambahan ataupun perubahan. Model dari NLP akan didasarkan pada segi

pengejaan kata, bagaimana kata-kata tersebut digabungkan menjadi satu kalimat, dan konteks kata yang ada pada kalimat tersebut. Dalam NLP terdapat beberapa disiplin ilmu[1]:

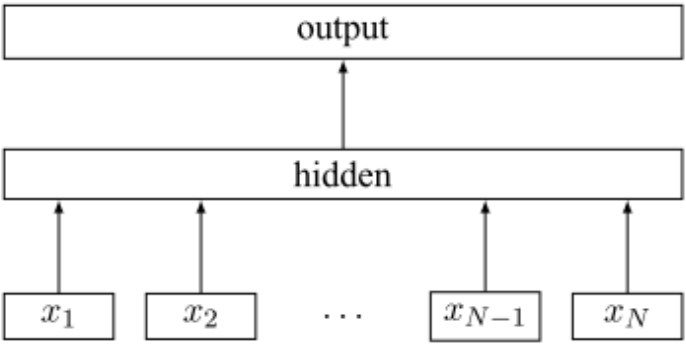
- Fonetik/fonologi: Pengetahuan terkait suara yang membentuk kata yang bermakna.
- Morfologi: Pengetahuan terkait bentuk-bentuk kata
- Sintaksis: Pengetahuan terkait sintaks/urutan kata dalam pembentukan kalimat.
- Semantik: Pengetahuan terkait makna kata dan pembentukan arti kalimat dari kata-kata penyusunnya.
- Pragmatik: Pengetahuan terkait konteks kata/kalimat yang berhubungan dalam suatu keadaan tertentu.

2.2.2. Word Embedding

Word embedding atau *distributed representation* adalah salah satu pendekatan dari *word representation*. Word embedding bersifat padat, berdimensi rendah, dan bernilai riil. Setiap dimensi dari *embedding* merepresentasikan fitur implisit dari kata tersebut, yang diharapkan dapat menangkap sifat-sifat sintaksis dan semantik yang berguna.[8]

2.2.3. FastText

FastText adalah suatu model *word embedding* yang dikembangkan oleh tim riset AI Facebook. Model ini terinspirasi dari penelitian oleh Mikolov dkk. dan berhasil menunjukkan bahwa model mereka mampu melakukan *training* pada 1 milyar kata dalam waktu 10 menit, dengan kualitas hasil yang tidak kalah dengan model-model lainnya[6]. Arsitektur model *FastText* mirip dengan arsitektur CBOW pada *Word2Vec* namun memiliki struktur hirarki dan merepresentasikan kata dalam bentuk *dense vector*. Dan diantara *input layer* dan *output layer* terdapat *hidden layer* seperti di bawah ini[6].



2.2.4. Hierarchical Softmax

Hierarchical Softmax adalah metode *training* yang awalnya dikenalkan oleh Mikolov dkk. untuk model Skip-gram pada model *Word2Vec*, dan metode training ini diadaptasi dan diaplikasikan pada model *FastText*. Model ini merupakan optimasi dari metode *full softmax*, sehingga mengurangi jumlah evaluasi *node* dari *W* menjadi sekitar $\log_2(W)$.

Metode ini menggunakan representasi *binary tree* dari *output layer* dengan kata-kata (*W*) sebagai *nodes*, dan pada setiap *node* akan merepresentasikan probabilitas kedekatan *child node*-nya secara eksplisit.[5][6]

2.2.5. Word2Vec

Word2Vec adalah suatu model untuk word embedding yang dibuat oleh Mikolov dkk. Dimana model ini mengimplementasikan arsitektur *Continuous-bag-of-words* dan *Skip-Gram* secara efisien untuk menghitung representasi vektor kata-kata. Representasi ini kemudian dapat digunakan di berbagai aplikasi NLP dan untuk penelitian lanjut.[5]

2.2.6. Levenshtein Distance

Levenshtein distance adalah salah satu metode pengukuran perbedaan antara dua *string* dalam bentuk jarak atau *distance*.

Ukuran ini didapat dengan mencari jumlah minimum dari operasi penyisipan, penghapusan, atau substitusi karakter dari suatu *string* awal menjadi *string* target[9]. Aturan dalam mencari nilai *edit distance* pada algoritma ini adalah untuk satu operasi penyisipan karakter diberi nilai 1, untuk satu operasi penghapusan karakter diberi nilai 1, dan satu operasi substitusi karakter diberi nilai 1. Apabila tidak terjadi perubahan karakter, diberi nilai 0. Sehingga semakin tinggi nilai *Levenshtein distance*, maka semakin besar perbedaan antara 2 *string* tersebut.

2.2.7. Jaro-Winkler Distance

Jaro-Winkler Distance juga merupakan salah satu algoritma pengukuran jarak atau *distance* antara dua *string*. Algoritma ini merupakan varian dari *Jaro distance metric*. Berbeda dengan nilai *Levensthein distance*, Semakin tinggi nilai ukuran ini, maka semakin mirip kedua *string* tersebut. Skor normalnya adalah 0 untuk tidak ada kesamaan, dan 1 menunjukkan sama persis[10].

2.2.8. Media Sosial

Media sosial adalah suatu media daring yang mewadahi penggunanya untuk saling berinteraksi, berbagi, dan menciptakan konten, diantaranya melalui blog, jejaring sosial, dan forum. Media sosial juga menggambarkan berbagai macam teknologi yang mengikat orang-orang ke dalam suatu kolaborasi, saling bertukar informasi, dan berinteraksi melalui pesan yang berbasis web (Michael Cross, 2003). Media sosial menjadi suatu sarana untuk berkomunikasi dan berkolaborasi di antara jaringan orang-orang, masyarakat, komunitas, dan organisasi yang saling terkait dan saling tergantung, serta diperkuat oleh kemampuan dan mobilitas teknologi (Tracy L. Truten). Media sosial yang banyak digunakan di indonesia diantaranya Facebook, Twitter, dan Instagram.

2.2.9. Twitter

Twitter adalah situs daring media sosial dan berita dimana penggunanya dapat berkirim dan berinteraksi melalui pesan yang disebut dengan '*Tweet'*. Situs ini dibuat pada bulan Maret 2006 oleh Jack Dorsey, Noah Glass, Biz Stone, and Evan Williams dan diluncurkan pada bulan Juli di tahun yang sama. Pengiriman pesan teks pada tweet dibatasi dengan 140 karakter saja, namun sejak 7 Nopember 2017 batas tersebut dinaikkan menjadi 280 karakter. Hingga Mei 2016 pengguna jejaring sosial Twitter di Indonesia mencapai 24,34 juta, dan 70% diantaranya aktif[11].

2.2.10. Web Crawling

Web crawling adalah suatu metode menggunakan program komputer untuk mengambil data berupa dokumen elektronik dari satu atau beberapa situs Web. Data yang telah diambil kemudian disiapkan untuk dapat diproses oleh program lain[12].

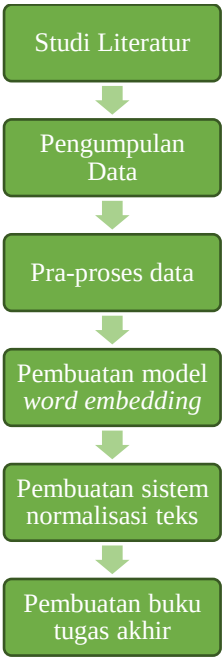
Halaman ini sengaja dikosongkan

BAB III
METODOLOGI

Pada bab metode penelitian akan dijelaskan mengenai tahapan – tahapan apa saja yang dilakukan dalam pengerjaan tugas akhir ini beserta deskripsi dan penjelasan tiap tahapan tersebut. Lalu disertakan jadwal pengerjaan tiap tahapan.

3.1. Tahapan Pelaksanaan Tugas Akhir

Pada sub bab ini akan menjelaskan mengenai metodologi dalam pelaksanaan tugas akhir. Metodologi ini dapat dilihat pada Gambar 3.1



Gambar 3.1 Metodologi Tugas Akhir

3.2. Uraian Metodologi

Pada bagian ini akan dijelaskan secara lebih rinci masing-masing tahapan yang dilakukan untuk penyelesaian tugas akhir ini.

3.2.1. Studi Literatur

Pada tahap ini dilakukan studi literatur agar dapat memahami konsep, metode, dan teknologi yang terkait dengan bahasan dan rumusan masalah tugas akhir ini agar dapat memberi solusi yang akan digunakan dalam penyusunan tugas akhir ini. Adapun literatur utama yang digunakan dalam tugas akhir ini adalah *Bag of Tricks for Efficient Text Classifier* oleh Armand Joulin dkk. dimana literatur tersebut membahas model utama yang akan digunakan dalam tugas akhir ini, yaitu model *FastText*.

3.2.2. Pengumpulan Data

Pada tahap ini dilakukan pengumpulan data dengan metode *crawling* dari beberapa akun di situs twitter. Data yang akan dikumpulkan berupa kiriman-kiriman teks dari media sosial tersebut. Secara umum, data teks yang akan dikumpulkan berasal dari akun yang sering melakukan posting, dan teks tersebut memiliki struktur bahasa baku maupun tidak baku.

3.2.3. Pra-proses Data

Data yang telah didapatkan dari proses sebelumnya kemudian masuk ke tahapan pra-proses data. Setiap data yang akan digunakan dalam proses *training* nanti akan masuk ke tahapan pra-proses data. Tahapan pra-proses data memiliki beberapa sub-proses di dalamnya, diawali dengan menggabungkan data agar menjadi satu dataset.

3.2.3.1 Penghapusan Data yang Duplikat

Besar kemungkinan data yang didapatkan dari media sosial merupakan data duplikat. *Multi-post* dan *post-spamming* adalah hal yang seringkali terjadi di platform media sosial *Twitter*.

Untuk itu perlu dilakukan penghapusan data agar tidak terjadi redundansi data dan data menjadi lebih reliable.

3.2.3.2 Menghapus Tanda Baca dan Simbol

Tanda baca dan simbol yang ada pada tiap-tiap data akan dihapus. Tujuannya adalah untuk menghilangkan karakter-karakter yang tidak bermakna dan memperkecil kemungkinan data memiliki penulisan yang berbeda dari kata yang sama karena terdapat perbedaan penulisan dengan tanda baca tertentu.

Tabel 3.2 Tabel sebelum dan sesudah penghapusan tanda baca dan simbol

Sebelum Penghapusan tanda baca	Setelah Penghapusan tanda baca
Paris Saint Germain (PSG) menjamu Real Madrid dalam leg kedua babak 16 Besar Liga Champions, Rabu (7/3/2018) WIB.	Paris Saint German PSG menjamu Real Madrid dalam leg kedua babak 16 Besar Liga Champions Rabu 7 3 2018 WIB.

3.2.3.3 Case Folding

Untuk melakukan pengoreksian, maka kalimat akan dicocokkan per kata. Untuk itu perlu dilakukan *case folding*, dimana huruf kapital akan dijadikan huruf kecil. Hal ini dapat menyebabkan sedikit ambiguitas, namun untuk normalisasi teks Bahasa Indonesia, hal ini tidak terlalu signifikan.

Tabel 3.3 Tabel sebelum dan sesudah case folding

Sebelum Case Folding	Setelah Case Folding
Paris Saint Germain (PSG) menjamu Real Madrid dalam leg kedua babak 16 Besar	paris saint german psg menjamu real madrid dalam leg kedua babak 16 besar liga

Liga Champions, Rabu (7/3/2018) WIB.	champions, rabu (7/3/2018) wib.

3.2.3.4 Penggabungan Baris yang Terpisah

Pada beberapa post akan ada teks yang memiliki baris-baris terpisah, dan untuk memudahkan pemrosesan data nanti, baris-baris tersebut harus digabungkan.

Tabel 3.3 Tabel sebelum dan sesudah penggabungan baris

Sebelum Penggabungan Baris	Setelah Penggabungan Baris
daftar travian sini boi https://www.travian.com/id?uc=id2_95 gausah main sampe kelar, sampe jadi desa 2 aja minimal, abis itu ditinggal gapapa deh	daftar travian sini boi https://www.travian.com/id?uc=id2_95 gausah main sampe kelar, sampe jadi desa 2 aja minimal, abis itu ditinggal gapapa deh

3.2.3.5 Tokenization

Tokenizing adalah tahapan dimana setiap korpus akan dipisahkan per kata menjadi satu *token*, dimana token-token tersebut akan merepresentasikan sebuah kata dalam vektor.

Tabel 3.4 Tabel sebelum dan sesudah Tokenization

Sebelum Tokenization	Setelah Tokenization
Panggil semua penggemar BLACKPINK! Gabunglah dengan kampanye ini untuk meminta konser	{Panggil, semua, penggemar BLACKPINK, Gabunglah, dengan, kampanye, ini, untuk, meminta, konser,

BLACKPINK ke Jakarta! Atas permintaan penggemar, MyMusicTaste telah membawa SHINee dan VIXX ke Jakarta. Bawa BLACKPINK ke kota anda melalui MyMusicTaste!	BLACKPINK, ke, Jakarta, Atas, permintaan, penggemar, MyMusicTaste, telah, membawa, SHINee, dan, VIXX, ke, Jakarta Bawa, BLACKPINK, ke, kota, anda, melalui, MyMusicTaste}

3.2.3.6 Kondisi Ideal

Setelah melalui tahapan pra-proses di atas, maka data teks berada dalam kondisi ideal untuk pemrosesan, namun akan dilakukan pengecekan bahasa sebelum pemrosesan dimulai.

Tabel 3.4 Tabel sebelum dan setelah pra-proses

Sebelum Pra-proses	Kondisi Ideal
Yaudah tau gitu lu gausah pake komen segala. Kalo gasuka tinggal unfriend aja drpd ngomel/komen/marah2 di status gw! Basi tau gak?!	{Yaudah, tau, gitu, lu, gausah, pake, komen, segala, Kalo, gasuka, tinggal, unfriend, aja, drpd, ngomel, komen, marah2, di, status, gw, Basi, tau, gak}

3.2.4. Pembuatan Model FastText dan Word2Vec

Pada tahap ini dibuat model untuk pemrosesan data lebih lanjut. Terdapat beberapa parameter kontrol yang akan digunakan dalam pembangunan model, seperti arsitektur yang digunakan (CBOW atau Skip-gram) dan metode training-nya (Negative Sampling/hierarchical softmax). Tujuannya adalah untuk

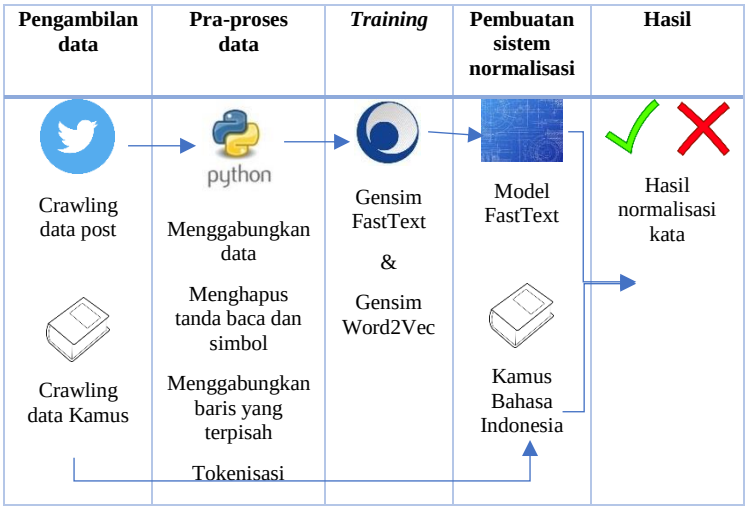
membuat model yang paling baik yang akan digunakan untuk pemrosesan data.

3.2.5.1 Pengujian Akurasi Model FastText dan Word2Vec

Pada tahap ini, model-model yang telah dibuat dengan parameter yang berbeda-beda akan diuji satu per satu akurasinya untuk menentukan model terbaik yang nantinya akan digunakan sebagai model untuk normalisasi teks.

3.2.5. Pembuatan Sistem Normalisasi Teks

Sistem normalisasi teks akan dibuat dengan gambaran umum seperti berikut:



Kata-kata yang akan dinormalisasi dicocokkan dengan kamus untuk melihat apakah kata tersebut merupakan kata baku. Jika kata tersebut ada pada kamus maka kata tersebut akan dicetak. Jika tidak, maka model dimuat kemudian dicari kata yang memiliki vektor paling dekat dengan kata tersebut. Setelah itu akan muncul hasil kandidat kata yang tepat untuk menormalisasi kata yang dimasukkan.

3.2.6.1 Pengujian dan Analisis Hasil Normalisasi Teks

Langkah selanjutnya adalah menguji sistem normalisasi teks dengan mengujikan data tes yang terdiri dari sejumlah post yang baku dan tidak baku. Pengujian dilakukan dengan memprediksi hasil luaran dari proses normalisasi kalimat tidak baku dengan kalimat baku yang dibuat secara manual. Keakuratan dihitung dari jumlah kata yang dinormalisasi dengan benar dibandingkan dengan jumlah kata dalam kalimat tersebut

3.2.6. Penyusunan Laporan Tugas Akhir

Tahapan terakhir adalah penyusunan laporan tugas akhir yang dibuat sesuai dengan format yang telah ditentukan. Pembuatan buku tugas akhir ini sebagai bentuk dokumentasi atas terlaksananya tugas akhir. Dan diharapkan akan dapat membantu dalam penelitian berikutnya. Tahapan penyusunan laporan tugas akhir dilakukan sejak awal hingga berakhirnya proses pengerjaan tugas akhir ini.

Halaman ini sengaja dikosongkan

BAB IV
PERANCANGAN

Pada bab ini perancangan terkait luaran dari tugas akhir ini akan dibahas.

4.1 Akuisisi Data

4.1.1 Perancangan Crawler Data Media Sosial

Dalam mengakuisisi data media sosial *twitter* diperlukan *crawler* yang dirancang berdasarkan API dari *twitter* yang membutuhkan akun *twitter* itu sendiri. Akun *twitter* yang digunakan untuk akuisisi data berjumlah 3 akun. Dari satu akun *twitter* dapat dibuat 3 *crawler*. Dimana masing-masing *crawler* akan mengambil data berdasarkan keyword acak berbahasa Indonesia. Inti *code* pada crawler ini mengacu pada *library tweepy* untuk Python.

4.1.2 Database untuk Data Media Sosial

Data yang telah diakuisisi awalnya disimpan dalam bentuk json pada file .txt secara *local*. Pada tahap ini data memiliki berbagai atribut, namun kemudian di*filter* untuk diambil hanya atribut *ID* dan *message* saja, karena berikutnya data akan dipindahkan pada *database* mySQL. *ID Tweet* diperlukan untuk mengurangi data yang duplikat dengan *query DISTINCT* dan *message* adalah data yang paling penting dan diperlukan untuk pemrosesan data pada tugas akhir ini.

4.1.3 Perancangan Crawler Data Kateglo

Untuk mengambil data kamus dari situs Kateglo (Kamus, Tesaurus, Glosarium) diperlukan *crawler* yang berbeda dengan *crawler* untuk situs *twitter*. Akuisisi data kamus dari situs Kateglo dilakukan dengan 2 tahap: tahap pertama adalah *scraping* untuk mengambil seluruh entri kamus, dan tahap kedua adalah penggunaan API Kateglo untuk mengambil *actual phrase* dari suatu kata yang akan digunakan sebagai *mapping* untuk normalisasi kata. Data kamus dan *mapping* yang telah diakuisisi disimpan secara *local*.

4.2 Rancangan Pra-proses Data

Perlu dilakukan pra-proses data yang telah diakuisisi sebelum *training* dilakukan, agar pemrosesan data menjadi lebih efektif dan menghasilkan olahan data yang lebih baik. Tahapan-tahapan pra-proses data yang dilakukan adalah sebagai berikut.

4.2.1 Penggabungan Dataset

Data yang diakuisisi pada awalnya berupa file-file .txt individu yang tersimpan berdasarkan kata kunci pencarian data. Untuk itu data-data tersebut perlu digabungkan menjadi satu data yang utuh pada database mySQL.

4.2.2 Penghapusan Tanda Baca dan Simbol

Adanya tanda baca pada data dapat mempersulit pemrosesan data karena adanya kemungkinan perbedaan penulisan suatu kata dengan penambahan tanda baca atau simbol tertentu, selain itu perlu dilakukan penghapusan tanda baca atau simbol yang kurang bermakna.

Tahap ini dilakukan dengan mengganti string berupa tanda baca, simbol, atau *emoticon* menjadi karakter spasi. Menggunakan *built-in function* dari Python dan *library* lain.

4.2.3 Penggabungan Baris yang Terpisah

Baris-baris yang terpisah pada data perlu digabungkan untuk mempermudah pemrosesan data pada tahap selanjutnya. Karena pada dasarnya program akan membaca data masukan secara baris per baris, sehingga untuk menghindari kesalahan dalam pembacaan data, tahap ini perlu dilakukan.

4.2.4 Tokenisasi

Tokenisasi dilakukan dengan memisahkan setiap kata pada korpus menjadi satu token per kata. Dimana token-token ini nanti akan merepresentasikan sebuah kata dalam vektor.

4.3 Perancangan *Training* Model

4.3.1 Perancangan *Training* Model FastText

Tahapan ini bertujuan untuk menghasilkan model FastText. Untuk itu diperlukan suatu *library* yaitu Gensim FastText yang

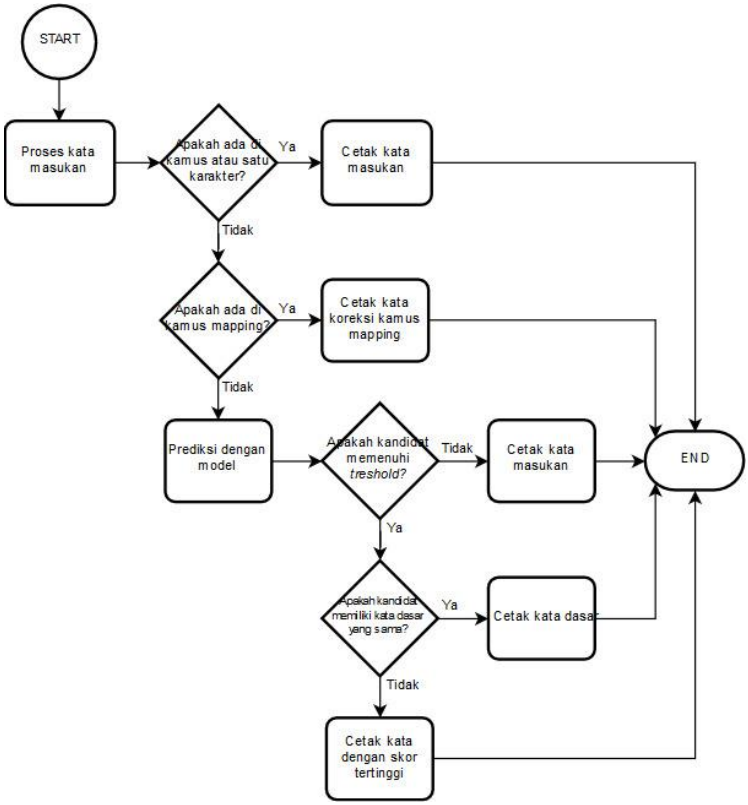
berbasis Python. Fitur dari *library* ini cukup lengkap, diantaranya melakukan *training* dengan parameter yang dapat kita ubah, menyimpan model hasil *training* dan memuat model tersebut, dan fitur yang cukup menarik dari *library* ini adalah dapat melakukan tugas-tugas NLP dasar, misalnya mencari kata-kata yang (memiliki vektor) berdekatan dengan kata masukan, dan mencari kata yang tidak sesuai diantara kelompok kata-kata yang dimasukkan.

Tahapan ini akan dilakukan beberapa kali dengan parameter *training* yang berbeda, kemudian dicari model yang terbaik. Adapun parameter yang dapat diganti adalah sebagai berikut:

1. *Sg* : parameter ini menentukan *learning algorithm* apa yang akan digunakan. Terdapat 2 pilihan *learning algorithm*, *skip-gram* atau *CBOW*.
2. *Size*: parameter ini menentukan dimensi dari *vector*.
3. *Window*: parameter ini menentukan jumlah kata sebelum dan sesudah kata tertentu yang digunakan sebagai pertimbangan konteks dalam satu kalimat.
4. *Min_count*: parameter ini menentukan jumlah minimum kemunculan suatu kata agar kata tersebut tidak diabaikan.
5. *Iter*: parameter ini menentukan jumlah iterasi dilakukannya training.
6. *Min_n*: parameter ini menentukan panjang minimum *character n-gram* yang akan digunakan untuk *training* representasi kata.
7. *Max_n*: parameter ini menentukan panjang maksimum *character n-gram* yang akan digunakan untuk *traning* representasi kata.

4.4 Perancangan Sistem Normalisasi Teks

Sistem normalisasi teks akan dibuat sesuai diagram alur di bawah ini.



Gambar 4.1 Diagram Alur Sistem Normalisasi Teks

Untuk setiap *token* yang akan dikoreksi, sistem akan mengecek apakah masukan ada di kamus utama atau merupakan 1 karakter. Jika iya, maka sistem akan mencetak kata masukan. Jika tidak, maka melanjutkan pengecekan ke kamus mapping. Jika ada pada kamus mapping, maka kata koreksi pada kamus mapping akan dicetak. Jika tidak, maka akan diprediksi oleh model. Apabila tidak ada hasil prediksi yang memenuhi *threshold*, maka kata masukan akan dicetak kembali. Jika ada yang memenuhi *threshold*, maka sistem akan mengecek apakah *token* memiliki kata dasar yang sama dengan hasil prediksi. Jika ada hasil prediksi yang memiliki kata dasar sama dengan *token*,

maka kata dasar akan dicetak. Jika tidak, maka sistem akan mencetak kata yang memiliki skor tertinggi.

4.5 Perancangan Evaluasi Model

Model-model yang telah terbentuk dari tahapan *training* sebelumnya akan dipilih yang terbaik melalui evaluasi model. Evaluasi dilakukan dengan melihat ketepatan model dalam menghasilkan kandidat pembenaran kata dari seratus kata tidak baku yang paling sering muncul. Untuk itu, perlu dilakukan pencarian kata-kata tersebut.

4.5.1 Pencarian Kata Tidak Baku yang Paling Sering Muncul

Salah satu tahapan pada *training* dengan menggunakan *Gensim FastText* sudah mengurutkan kemunculan kata dalam korpus mulai dari yang paling banyak kemunculannya ke yang paling sedikit. Sehingga untuk mencari kata tidak baku yang paling sering muncul, kata akan dicetak dari indeks terkecil, dan diperiksa terhadap data kamus *kateglo* yang telah diakuisisi, untuk melihat apakah kata tersebut termasuk kata baku atau tidak, hingga muncul 1000 kata yang tidak baku.

4.5.2 Pengujian Model dengan Seratus Kata Non-Kamus

Pengujian model dengan 100 kata non-kamus akan dilakukan untuk memilih model terbaik diantara model-model yang telah dibuat. Pengujian ini dilakukan dengan mengecek kemampuan model dalam menghasilkan prediksi kata yang bukan merupakan kata kamus. Seratus kata akan dimuat dan diuji terhadap model dengan perulangan. Model kemudian akan menghasilkan 500 kandidat kata terdekat, namun yang diambil hanya 10 kata yang merupakan kata kamus. Setelah 10 kata didapatkan, perulangan dihentikan. Akurasi ditentukan dengan mengecek 10 kandidat kata yang dimunculkan, apakah berhasil mengeluarkan koreksi kata yang benar. Jika koreksi kata yang benar dimunculkan, lalu diberi skor berdasarkan urutan kemunculan kata tersebut.

4.5.3 Pengujian Model Terbaik dengan Seribu Kata Non-Kamus

Setelah model terbaik dipilih, model tersebut akan diujikan kembali dengan 1000 kata non kamus, dimana kata-kata tersebut akan dimuat dan diuji seperti pengujian sebelumnya, namun kali ini dengan menambahkan skor *levenshtein distance* dan *jaro-winkler distance*. Kata yang benar akan ditandai berdasarkan beberapa kategori, kemudian dicari skor tertinggi dan terendahnya. Hal ini dilakukan untuk menentukan *threshold* pengujian. *Threshold* ini digunakan sebagai batas minimal yang harus dipenuhi agar suatu kata dapat disebut ‘berhasil dikoreksi’.

BAB V
IMPLEMENTASI

Pada bab ini dijelaskan hasil dari implementasi perancangan yang telah dijelaskan pada bab perancangan. Penjelasan dalam bagian implementasi meliputi lingkungan implementasi, penjelasan kode, dan pengujian aplikasi.

5.1 Lingkungan Implementasi

Pada bagian ini, dipaparkan spesifikasi perangkat yang digunakan dalam pembuatan dan implementasi penelitian. Perangkat yang dimaksud meliputi perangkat keras dan perangkat lunak pengembangan aplikasi. Tabel 5.1 dan Tabel 5.2 merupakan daftar perangkat yang digunakan dalam penelitian ini:

Tabel 5.1 Spesifikasi Perangkat Lunak

Hardware	Spesifikasi
Jenis	ASUS ROG GL503 VD
Processor	Intel Core i7-7700HQ
Memory	2x DDR4 8GB
Storage	- 128GB SSD - 1TB HDD

Tabel 5.2 Perangkat Lunak Pengembangan Aplikasi

Fungsi	Software / Teknologi
Sistem Operasi	Windows 10 Home Single Language 64-Bit
Bahasa Pemrograman	Python 3.6.4 x64
IDE	- Spyder 3.2.8

	- Visual Studio Code 1.24.1
Text Editor	- Sublime Text 3 - EmEditor - Notepad
Data Labeling	Microsoft Excel 365
Web Browser	- Google Chrome version 67 - Mozilla Firefox
Python Library	- beautifulsoup4 (bs4) 4.6.0 - emot 1.0 - gensim 3.4.0 - jsonpickle 1.0 - langdetect 1.0.7 - matplotlib 2.1.2 - nltk 3.2.5 - numpy 1.11.3 - pandas 0.22.0 - requests 2.18.4 - textdistance 3.0.3 - tweepy 3.7.0 - tweet-preprocessor 0.5.0

5.2 Implementasi Akuisisi Data

5.2.1 Pengumpulan data Twitter

Situs *twitter* memiliki API tersendiri yang dapat dihubungkan dengan kode python untuk mengambil data tweet. Pengambilan data tweet menggunakan library utama tweepy.

```
1     searchQuery = input('keyword ')
2     tweetsPerQry = 100
3     fName = searchQuery+".txt"
4     max_id = 999999999999999999999999999
```

Kode 5.1 Variabel pada *Crawler Twitter*

Kode 5.1 menunjukkan variabel yang digunakan pada *crawler* ini, searchQuery adalah kata kunci yang digunakan untuk melakukan *search* tweet. Variabel *tweetsPerQry* adalah jumlah tweet yang diambil dalam sekali query. Namun API membatasi nilai variabel ini, hanya 100 tweet yang dapat diambil untuk sekali query. Variabel *fName* adalah nama file yang akan

digunakan untuk penyimpanan hasil. Dan variabel max_id di *set* setinggi mungkin, karena pengambilan data dimulai dari tweet yang paling baru hingga yang paling lama.

```
1     with open(fName, 'w') as f:
2         while True:
3             try:
4                 if (max_id <= 0):
5                     if (not sinceId):
6                         new_tweets =
7     api.search(q=searchQuery, count=tweetsPerQry,
8     tweet_mode='extended')
9                     else:
10                        new_tweets =
11    api.search(q=searchQuery, count=tweetsPerQry,
12    since_id=sinceId, tweet_mode='extended')
13                else:
14                    if (not sinceId):
15                        new_tweets =
16    api.search(q=searchQuery, count=tweetsPerQry,
17    max_id=str(max_id - 1), tweet_mode='extended')
18                    else:
19                        new_tweets =
20    api.search(q=searchQuery, count=tweetsPerQry,
21    max_id=str(max_id - 1),
22    since_id=sinceId, tweet_mode='extended')
23                if not new_tweets:
24                    print("No more tweets found")
25                    break
26                for tweet in new_tweets:
27                    f.write(jsonpickle.encode(tweet._json,
28    unpicklable=False) +'\n')
29                tweetCount += len(new_tweets)
30                print("Downloaded {0}
31    tweets".format(tweetCount))
32                max_id = new_tweets[-1].id
33            except tweepy.TweepError as e:
34                print("some error : " + str(e))
```

Kode 5.2 Sistem utama crawler

Kode 5.2 adalah sistem utama crawler. Karena pengambilan tweet dibatasi 100 per query oleh API, maka digunakan perulangan *while (line 2-39)*, sehingga query dilakukan berulang-ulang, search dimulai dari max_ID, lalu dikurangi 1 setiap perulangan, hingga batas akhir *search tweet* yaitu tweet yang berusia 1 minggu. Setelah tweet didapatkan, lalu ditulis ke

dalam file .txt berbentuk json yang diberi nama berdasarkan kata kunci pencarian.

```
1     import json
2     import sys
3
4     class read(object):
5
6         counter = 1
7         searchQuery = input('keyword : ')
8         fName = str(searchQuery) + '.txt'
9         tweets = []
10
11        with open(fName) as lastdata:
12            with open('output_filter/' + fName, 'w') as
13    savefilter:
14                for line in lastdata:
15                    datatweet = json.loads(line)
16                    data = {}
17                    data['id'] = datatweet['id']
18                    data['message'] =
19    datatweet['full_text']
20
21                    json.dump(data,savefilter)
22                    savefilter.write('\n')
23
24                    print(searchQuery + " - " +
25    str(counter))
26                    counter = counter + 1
```

Kode 5.3 Potongan Kode untuk Menyaring *Tweet*

Data tweet yang telah di*crawl* sebelumnya perlu disaring karena tidak semua data dari tweet tersebut dibutuhkan. Data yang paling penting untuk diambil adalah *ID* dan *message* tweet tersebut. Setiap file hasil *crawling* dimuat lalu dibaca per baris. Kemudian *key* dan *value* dari *key ID* dan *message* akan diambil lalu dimasukkan ke dalam file *keyword.txt* pada foler filter_crawling dengan perulangan.

5.2.2 Pengumpulan data Kateglo

Pengumpulan data kamus dari situs Kateglo dilakukan dengan metode *scraping*, menggunakan library utama *requests* dan *beautifulsoup4*.

```
1     file = open('kamus.txt', 'w')
2
3
```

```
4     res =
5     requests.get('http://kateglo.com/?&mod=dictionary&srch=a
6     ll')
7     soup = bs4.BeautifulSoup(res.text, 'html.parser')
8
9     x = soup.find_all('dt')
10
11    for dt in x:
12        for a in dt.find_all('a'):
13            print(a.text)
14            file.write("\n"+a.text)
```

Kode 5.4 Potongan Kode mengambil kata kamus dalam halaman utama kamus Kateglo

Kode 5.4 menunjukkan proses pengambilan kata pada halaman pertama kamus Kateglo. Pertama-tama dilakukan *request* halaman sesuai url (baris 5-6), kemudian halaman tersebut di-*parse* (baris 7), dan dari hasil parsing akan terlihat bahwa kata kamus berada pada elemen <a> yang ada di dalam elemen <dt>. Sehingga konten elemen tersebut yang diambil (baris 9-13), dan dimasukkan ke dalam file kamus.txt.

```
1     for x in range (2, 1446):
2         res = requests.get('
3     http://kateglo.com/?&mod=dictionary&srch=all&op=1&p='+
4     str(x))
5         soup = bs4.BeautifulSoup(res.text, 'html.parser')
6
7         print(str(x))
8
9         x = soup.find_all('dt')
10        for dt in x:
11            for a in dt.find_all('a'):
12                print(a.text)
13                file.write("\n"+a.text)
14
15        while True:
16            try:
17                res.raise_for_status()
18            except Exception:
19                time.sleep(2)
20            break
```

Kode 5.5 Potongan Kode untuk Mengambil Kata Kamus untuk Halaman Kedua Hingga Terakhir

Pengambilan kata dari halaman pertama sedikit berbeda dengan halaman-halaman setelahnya karena url pada halaman

pertama memiliki pola yang berbeda dengan url pada halaman kedua hingga terakhir. Selain itu, *request* yang terlalu sering pada situs tersebut terkadang memunculkan error dimana halaman tidak dimuat. Sehingga perlu dilakukan pengecekan error. Apabila error terjadi, maka sistem akan menunggu selama 2 detik sebelum mencoba kembali dan melanjutkan *scraping*.

5.3 Implementasi Pra-proses data

5.3.1 Penggabungan Data Menjadi Satu Dataset

Data *twitter* yang telah didapatkan kemudian digabung menjadi satu dataset utuh di database MySQL.

```
1    with open(listname) as listsearch:
2        for line in listsearch:
3            namedir = line.strip()[:-5]
4            topic = line.strip()[9:-5]
5            with open('READ_'+ st +'_'+namedir+'.sql','a')
6    as saveFile:
7                with open(line.strip()) as json_data:
8                    for line in json_data:
9                        try:
10                           tweet = json.loads(line)
11                           if 'message' in tweet:
12                               id = tweet['id']
13                               text = tweet['message']
14
15                           abc = id,text
16                           print(text)
17                           print('INSERT INTO twitter
18   VALUES {};'.format(abc))
19                           saveFile.write('INSERT INTO
20   twitter VALUES {};'.format(str(abc)))
21                           saveFile.write('\n')
22                       except:
23
24                           continue
25
26                   print('\n' + 'COMPLETE READ : {}
27   '.format(namedir) + '\n')
```

Kode 5.6 Potongan Kode untuk Menghasilkan *Query SQL*

Kode 5.6 dijalankan untuk menghasilkan *query* SQL dimana kita akan memasukkan setiap ID dan *message tweet* ke database "twitter". Program akan membaca data tweet yang telah

dikumpulkan, lalu dibuat query untuk memasukkan value setiap barisnya ke database. Setelah itu, query dijalankan secara manual.

5.3.2 Penghapusan Data Duplikat

Terdapat banyak data duplikat jika mengambil dari situs *twitter* karena adanya fitur *retweet*. *Tweet* yang dihasilkan dari fitur tersebut tetap terdeteksi sebagai tweet baru dan dimasukkan ke dalam dataset. Oleh karena itu, dilakukan penghapusan data duplikat dengan menggunakan *query* sederhana di mySQL.

```
CREATE TABLE twitter_distinct LIKE twitter;
INSERT INTO twitter_distinct VALUES
   (SELECT * FROM twitter GROUP BY message);
```

Kode 5.7 *Query* untuk menghapus tweet yang duplikat

Proses yang dilakukan oleh *query* pada Kode 5.7 adalah membuat tabel baru dengan kolom yang sama seperti database awal yaitu ‘twitter’, kemudian melakukan *select* dengan *group by*. Hal ini akan mengambil 1 tweet dengan *message* yang sama lalu hasil seleksi tersebut dimasukkan ke tabel ‘twitter_distinct’. Data yang telah diproses kemudian di *export* kembali menjadi file .json.

5.3.3 Pra-proses Data Teks

Pra-proses berikutnya adalah pra-proses data teks, dimana telah dijelaskan pada bab sebelumnya, pra-proses teks yang dilakukan adalah *case folding*, penghapusan simbol dan karakter khusus, penghapusan url, dan *tokenizing*.

```
1    def process(self, text):
2            text = str(text).lower()
3            if text.isdigit():
4                return text
5            else:
6                text = str(text).strip()
7                text = tweetp.clean(text)
8                text = re.sub(r'[^a-zA-Z ]+', ' ', text)
9                text = ' '.join(text.split())
10               for term in self.custom_remover:
11                   text = re.sub(f'{term}', '', text)
12           return text
```

Kode 5.8 Potongan Kode Pra-proses Teks

Kode 5.8 menunjukkan *function* untuk pra-proses yang dilakukan. Pertama teks akan di *lowercase* seluruhnya, kemudian di *strip* untuk menghilangkan spasi yang berlebih. ”tweetp.clean()” adalah *function* dari library tweet-preprocessor, gunanya adalah untuk membersihkan data *tweet* dari URL, tagar, *mention*, *RT*, *FAV*, emoji, dan *smiley*. Kemudian dengan *regex* kita mengganti semua yang bukan karakter huruf menjadi spasi. Lalu ada ‘custom_remover’ yang gunanya untuk menghilangkan unsur-unsur lain dalam teks yang telah didefinisikan terlebih dahulu. Yang dihilangkan di sini adalah “http” dan “https”, untuk meminimalisir kemungkinan bahwa kedua unsur tersebut masih muncul dalam teks, walaupun telah dilakukan penghapusan url.

5.4 Pembuatan Model

5.4.1 Pembuatan Model FastText

```
class Fasttext():
  def __init__(self):
    self.size = 100
    self.num_features = 300
    self.num_workers = multiprocessing.cpu_count()
    self.sg = 0
    self.min_word_count = 5
    self.context_size = 7
    self.window = 5
    self.seed = 1
    self.iter = 2
    self.sorted_vocab = 1
    self.downsampling = 1e-3
    self.min_n = 3
    self.max_n = 6
```

Kode 5.9 Potongan Kode Inisialisasi Parameter *Training* Model *FastText*

Dalam membuat model *FastText*, langkah pertama yang dilakukan adalah memberikan *default parameter* untuk training yang akan dilakukan. Beberapa *parameter* pada kode 5.9 nantinya akan di-*override* ketika pemanggilan kelas berikutnya.

```
if __name__ == "__main__":

  fasttext = Fasttext()
  print("\nLoading Word Embedding RAW data ...")
```

```
5
6       file_read = ReadData('twitter.json')
7       sentences = []
8
9       for id, message in file_read:
10        if len(message) > 0 and message.isdigit() == False:
11          sentences.append(sentence_to_wordlist(message))
12
```

Kode 5.10 Potongan Kode *Training Model FastText*

Setelah menginisiasi kelas *FastText*, training model dilakukan. File data tweet yang akan di-training dimuat menjadi variabel *file_read* dengan memanggil objek *ReadData*, yang merupakan kelas pembersihan tweet. Kemudian untuk setiap baris *message* yang bukan terdiri dari 1 karakter dan bukan angka, *message* di-tokenize dengan memanggil *function sentence_to_wordlist()*. Kemudian *training* dilakukan dengan memanggil kelas *ft()*, dengan parameter yang sudah diubah.

5.4.1.1 Training Model *FastText 1*

```
1     print("Inisiasi FastText")
2       model_ft = ft(size = 300, sg = 0, min_count = 5,
3     window = 7, iter = 2, min_n = 3, max_n = 6)
4       model_ft.build_vocab(sentences)
5
6       print("\nFastText vocabulary length :
7     {}".format(len(model_ft.wv.vocab)))
8       token_count = sum([len(sentence) for sentence in
9     sentences])
10      print("\nCorpus contains {0:,}
11    tokens".format(token_count))
12
13      print("\nBuilding fasttext model...")
14      start = time.time()
15      model_ft.train(sentences, total_examples =
16    token_count, epochs=model_ft.iter)
17      end = time.time()
18
19      print("\nFinished!")
20      print("\nfastText training done in {}
21    seconds".format(end - start))
22
23      model_ft.save("modelV1_ft")
```

Kode 5.11 Training Model *FastText* Pertama

Kode 5.11 adalah kode untuk training model pertama yang dibuat. Setelah menginisiasi kelas *ft* dengan parameter yang baru, training dimulai dengan menjalankan fungsi *train()*. Setelah proses *training* selesai dilakukan, maka model disimpan dengan nama "modelV1_ft". Adapun detail parameter untuk *training* model adalah sebagai berikut.

Tabel 5.3 Parameter *Training* Model *FastText* Pertama

Parameter	Nilai
Dimension Size	300
Training Algorithm	CBOW
Minimum word freq.	5
Context window	7
Iterasi	2
Minimum character n-gram	3
Maximum character n-gram	6

5.4.1.2 Training Model *FastText 2*

```
print("Inisiasi FastText")
model_ft = ft(size = 300, sg = 1, min_count = 5, window
= 7, iter = 2, min_n = 3, max_n = 6)
  model_ft.build_vocab(sentences)

  print("\nFastText vocabulary length :
{}".format(len(model_ft.wv.vocab)))
  token_count = sum([len(sentence) for sentence in
sentences])
  print("\nCorpus contains {0:,}
tokens".format(token_count))

  print("\nBuilding fasttext model...")
  start = time.time()
  model_ft.train(sentences, total_examples =
token_count, epochs=model_ft.iter)
  end = time.time()

  print("\nFinished!")

```

```
    print("\nfastText training done in {}
seconds".format(end - start))

    model_ft.save("modelV2_ft")
```

Kode 5.12 Training Model *Fasttext* Kedua

Kode 5.12 adalah kode untuk training model kedua yang dibuat. Setelah menginisiasi kelas *ft* dengan parameter yang baru, training dimulai dengan menjalankan fungsi *train()*. Setelah proses *training* selesai dilakukan, maka model disimpan dengan nama "modelV2_ft". Adapun detail parameter untuk *training* model adalah sebagai berikut.

Tabel 5.4 Parameter *Training* Model *FastText* Kedua

Parameter	Nilai
Dimension Size	300
Training Algorithm	Skip-gram
Minimum word freq.	5
Context window	7
Iterasi	2
Minimum character n-gram	3
Maximum character n-gram	6

5.4.1.3 Training Model *FastText 3*

```
print("Inisiasi FastText")
model_ft = ft(size = 300, sg = 0, min_count = 5, window
= 5, iter = 2, min_n = 3, max_n = 6)
  model_ft.build_vocab(sentences)

  print("\nFastText vocabulary length :
{}".format(len(model_ft.wv.vocab)))
  token_count = sum([len(sentence) for sentence in
sentences])
  print("\nCorpus contains {0:,}
tokens".format(token_count))

  print("\nBuilding fasttext model...")
```

```
14    start = time.time()
15    model_ft.train(sentences, total_examples =
16  token_count, epochs=model_ft.iter)
17    end = time.time()
18
19    print("\nFinished!")
20    print("\nfastText training done in {}
21  seconds".format(end - start))
22
23    model_ft.save("modelV3_ft")
24
```

Kode 5.13 Training Model *FastText* Ketiga

Potongan kode di atas adalah kode untuk training model ketiga yang dibuat. Setelah menginisiasi kelas *ft* dengan parameter yang baru, training dimulai dengan menjalankan fungsi *train()*. Setelah proses *training* selesai dilakukan, maka model disimpan dengan nama "modelV3_ft". Adapun detail parameter untuk *training* model adalah sebagai berikut.

Tabel 5.5 Parameter *Training* Model *FastText* Ketiga

Parameter	Nilai
Dimension Size	300
Training Algorithm	CBOW
Minimum word freq.	5
Context window	5
Iterasi	2
Minimum character n-gram	3
Maximum character n-gram	6

5.4.1.4 Training Model *FastText 4*

```
1   print("Inisiasi FastText")
2   model_ft = ft(size = 300, sg = 1, min_count = 5, window
3   = 5, iter = 2, min_n = 3, max_n = 6)
4     model_ft.build_vocab(sentences)
5
6
```

```
    print("\nFastText vocabulary length :
{}".format(len(model_ft.wv.vocab)))
    token_count = sum([len(sentence) for sentence in
sentences])
    print("\nCorpus contains {0:,}
tokens".format(token_count))

    print("\nBuilding fasttext model...")
    start = time.time()
    model_ft.train(sentences, total_examples =
token_count, epochs=model_ft.iter)
    end = time.time()

    print("\nFinished!")
    print("\nfastText training done in {}
seconds".format(end - start))

    model_ft.save("modelV4_ft")
```

Kode 5.14 Training Model *FastText* Keempat

Potongan kode di atas adalah kode untuk training model keempat yang dibuat. Setelah menginisiasi kelas *ft* dengan parameter yang baru, training dimulai dengan menjalankan fungsi *train()*. Setelah proses *training* selesai dilakukan, maka model disimpan dengan nama "modelV4_ft". Adapun detail parameter untuk *training* model adalah sebagai berikut.

Tabel 5.6 Parameter *Training* Model *FastText* Keempat

Parameter	Nilai
Dimension Size	300
Training Algorithm	Skip-gram
Minimum word freq.	5
Context window	5
Iterasi	2
Minimum character n-gram	3
Maximum character n-gram	6

5.4.2 Pembuatan Model Word2Vec

```
class Word2vec():
  def __init__(self):
    self.size = 100
    self.num_workers = multiprocessing.cpu_count()
    self.sg = 0
    self.min_word_count = 5
    self.window = 5
    self.seed = 1
    self.iter = 5
    self.downsampling = 1e-3
```

Kode 5.15 Potongan Kode Inisialisasi Parameter *Training* Model *Word2Vec*

Dalam membuat model *Word2Vec*, langkah pertama yang dilakukan adalah memberikan *default parameter* untuk training yang akan dilakukan. Beberapa *parameter* di atas nantinya akan di-*override* ketika pemanggilan kelas berikutnya.

```
if __name__ == "__main__":

  word2vec = Word2vec()
  print("\nLoading Word Embedding RAW data ...")

  file_read = ReadData('twitter.json')
  sentences = []
  for id, message in file_read:
    if len(message) > 0:
      sentences.append(sentence_to_wordlist(message))
```

Kode 5.16 Potongan Kode *Training* Model *Word2Vec*

Setelah menginisiasi kelas *Word2vec*, training model dilakukan. File data tweet yang akan di-training dimuat menjadi variabel *file_read* dengan memanggil objek *ReadData*, yang merupakan kelas pembersihan tweet. Kemudian untuk setiap baris *message* yang bukan terdiri dari 1 karakter dan bukan angka, *message* di-tokenize dengan memanggil *function sentence_to_wordlist()*. Kemudian *training* dilakukan dengan memanggil kelas *Word2Vec()*, dengan parameter yang sudah diubah.

5.4.2.1 *Training* Model *Word2Vec 1*

```
print("Inisiasi Word2Vec")
model_w2v = Word2Vec(size= 300, sg = 1, min_count =
5, window = 7, iter = 2)
```

```
4         model_w2v.build_vocab(sentences)
5         print("\nWord2Vec vocabulary length :
6       {}".format(len(model_w2v.wv.vocab)))
7         token_count = sum([len(sentence) for sentence in
8       sentences])
9         print("\nCorpus contains {0:,}
10      tokens".format(token_count))
11        print("\nBuilding word2vec model...")
12        start = time.time()
13
14        model_w2v.train(sentences, total_examples =
15      token_count, epochs=model_w2v.iter)
16        end = time.time()
17
18        print("word2vec training done in {}
19      seconds".format(end - start))
20
21        model_w2v.save("modelV1_w2v")
```

Kode 5.17 Training Model *Word2Vec* Pertama

Kode 5.17 adalah kode untuk training model pertama yang dibuat. Setelah menginisiasi kelas *Word2vec* dengan parameter yang baru, training dimulai dengan menjalankan fungsi *train()*. Setelah proses *training* selesai dilakukan, maka model disimpan dengan nama "modelV1_w2v". Adapun detail parameter untuk *training* model adalah sebagai berikut.

Tabel 5.7 Parameter *Training* Model *Word2Vec* Pertama

Parameter	Nilai
Dimension Size	300
Training Algorithm	Skip-gram
Minimum word freq.	5
Context window	7
Iterasi	2

5.4.2.2 *Training* Model *Word2Vec 2*

```
1       print("Inisiasi Word2Vec")
2       model_w2v = Word2Vec(size= 300, sg = 1, min_count =
3       5, window = 5, iter = 2)
```

```
4         model_w2v.build_vocab(sentences)
5         print("\nWord2Vec vocabulary length :
6       {}".format(len(model_w2v.wv.vocab)))
7         token_count = sum([len(sentence) for sentence in
8       sentences])
9         print("\nCorpus contains {0:,}
10      tokens".format(token_count))
11        print("\nBuilding word2vec model...")
12        start = time.time()
13
14        model_w2v.train(sentences, total_examples =
15      token_count, epochs=model_w2v.iter)
16        end = time.time()
17
18        print("word2vec training done in {}
19      seconds".format(end - start))
20
21        model_w2v.save("modelV2_w2v")
```

Kode 5.18 Training Model *Word2Vec* Pertama

Kode 5.18 adalah kode untuk training model kedua yang dibuat. Setelah menginisiasi kelas *Word2vec* dengan parameter yang baru, training dimulai dengan menjalankan fungsi *train()*. Setelah proses *training* selesai dilakukan, maka model disimpan dengan nama "modelV2_w2v". Adapun detail parameter untuk *training* model adalah sebagai berikut.

Tabel 5.8 Parameter *Training* Model *Word2Vec* Pertama

Parameter	Nilai
Dimension Size	300
Training Algorithm	Skip-gram
Minimum word freq.	5
Context window	5
Iterasi	2

5.4.2.3 *Training* Model *Word2Vec 3*

```
1       print("Inisiasi Word2Vec")
2       model_w2v = Word2Vec(size= 300, sg = 0, min_count =
3       5, window = 5, iter = 2)
```

```
4         model_w2v.build_vocab(sentences)
5         print("\nWord2Vec vocabulary length :
6       {}".format(len(model_w2v.wv.vocab)))
7         token_count = sum([len(sentence) for sentence in
8       sentences])
9         print("\nCorpus contains {0:,}
10      tokens".format(token_count))
11        print("\nBuilding word2vec model...")
12        start = time.time()
13
14        model_w2v.train(sentences, total_examples =
15      token_count, epochs=model_w2v.iter)
16        end = time.time()
17
18        print("word2vec training done in {}
19      seconds".format(end - start))
20
21        model_w2v.save("modelV3_w2v")
```

Kode 5.18 Training Model *Word2Vec* Ketiga

Kode 5.18 adalah kode untuk training model ketiga yang dibuat. Setelah menginisiasi kelas *Word2vec* dengan parameter yang baru, training dimulai dengan menjalankan fungsi *train()*. Setelah proses *training* selesai dilakukan, maka model disimpan dengan nama "modelV3_w2v". Adapun detail parameter untuk *training* model adalah sebagai berikut.

Tabel 5.9 Parameter *Training* Model *Word2Vec* Ketiga

Parameter	Nilai
Dimension Size	300
Training Algorithm	CBOW
Minimum word freq.	5
Context window	5
Iterasi	2

5.4.3 Evaluasi Model

5.4.3.1 Pengambilan Kata Non-Kamus yang Paling Sering Muncul

Untuk mengevaluasi model yang telah dibuat, maka digunakan data uji berupa 100 kata non-kamus yang paling sering muncul. Untuk itu, dilakukan pengambilan kata-kata non-kamus yang paling sering muncul.

```
from gensim.models import FastText
import string

file =
open("D:/kuliah/ta/crawling/output_koreksi.txt","r")
new_model = FastText.load('modelV1_ft')
nonkamus = open("kata_non_kamus.txt","w")

d = []

for line in file:
  d.append(eval(line))

hit = 1

for asd in range(0,10000):
  kataKamus = False
  for word in d:
    if word["phrase"] == new_model.wv.index2word[asd] and
word["actual_phrase"] != None:
      kataKamus = True
    elif word["phrase"] == new_model.wv.index2word[asd]
and word["actual_phrase"] == None:
      kataKamus = True
    elif new_model.wv.index2word[asd].isdigit():
      kataKamus = True

  if kataKamus == False and
len(new_model.wv.index2word[asd]) != 1:

    n = print(hit, new_model.wv.index2word[asd],
new_model.wv.vocab[str(new_model.wv.index2word[asd])].cou
nt)
    nonkamus.write("\n"+ str(hit))
    nonkamus.write(", "+new_model.wv.index2word[asd])
    nonkamus.write(",
"+str(new_model.wv.vocab[str(new_model.wv.index2word[asd]
)].count))
    asd = asd + 1
    hit = hit + 1
```

Kode 5.19 Potongan kode untuk mengambil kata non kamus

Kode 5.19 menunjukkan proses pengambilan kata non kamus. Method *index2word* akan memunculkan kata berdasarkan indeks. Dalam model, kata-kata tersebut telah diurutkan mulai dari yang paling sering muncul ke yang paling jarang muncul. Dengan menggunakan perulangan dan method *index2word*, dilakukan pengecekan kata kamus. Apabila kata di kamus sama dengan kata yang dimunculkan *index2word*, maka kata tersebut akan ditulis ke dalam file kata_non_kamus.txt. begitu seterusnya hingga perulangan selesai. Kemudian, dibagi menjadi 2 file, 1 file berisi 100 baris teratas, dan file yang lain berisi 1000 baris teratas.

5.4.3.2 Pengujian 100 Kata Non-Kamus

Pengujian kemudian dimulai dengan memuat model, kamus, dan 100 kata non-kamus yang akan diujikan.

```
1     new_model = None
2       print("Loading model...")
3       _ = PROJECT_PATH.joinpath(
4           'data',
5           'model',
6           f'{FOLDER_MAPPING.get(pilihan[0])}',
7           f'{MODEL_MAPPING.get(pilihan[1])}'
8           )
9       if pilihan[0] == '1':
10        new_model = FastText.load(_.as_posix())
11      elif pilihan[0] == '2':
12        new_model = Word2Vec.load(_.as_posix())
```

Kode 5.20 Potongan Kode untuk Memuat Model

Potongan kode di atas menunjukkan proses pemuatan model. Setelah memilih model yang akan digunakan, maka model tersebut akan dimuat dengan menggunakan method *load()*, baik itu dari *FastText* maupun *Word2Vec*.

```
1     kamus = []
2       print("Loading kamus...")
3       _ = PROJECT_PATH.joinpath(
4           'data',
5           'kamus',
6           KAMUS_FILENAME
7           )
```

```
8       kamus_file = open(_, "r")
9       for line in kamus_file:
10        data = json.loads(line)
11        if '(cak)' not in data['arti']:
12          kamus.append(data)
13        else:
14          continue
```

Kode 5.21 Potongan Kode Memuat Kamus

Setelah model dimuat, berikutnya kamus dimuat. Setiap baris di file kamus dibaca dengan perulangan, lalu disimpan dalam variabel *kamus*, kecuali yang dalam arti katanya memiliki "(cak)" yang merupakan ragam cakapan.

```
1     print("Loading kata non kamus...")
2       daftar_cek = []
3       _ = PROJECT_PATH.joinpath(
4           'data',
5           'test',
6           CEK_KATA_MAPPING[cek_kata]
7           )
8       with open(_) as csv_file:
9         csv_reader = csv.reader(csv_file, delimiter=',')
10        for row in csv_reader:
11          daftar_cek.append(row[1])
```

Kode 5.22 Potongan Kode Memuat Kata Uji

Berikutnya adalah memuat kata uji. Dengan potongan kode diatas, file kata yang akan diujikan dimuat. Setiap baris di file kata uji akan dibaca lalu dimuat ke dalam variabel *daftar_cek*.

```
1      print("Mulai kalkulasi...")
2     with open(OUTPUT_FILE_PATH, mode="w") as
3   output_file:
4       csv_writer = csv.writer(output_file,
5   delimiter=',', quotechar='"',
6   quoting=csv.QUOTE_MINIMAL)
7       # Cek tiap kata
8       for word in daftar_cek:
9         prediksi_cocok = 0
10        # Generate prediksi
11        hasil_prediksi = new_model.most_similar(word,
12  topn = 500)
13        for prediksi in hasil_prediksi:
14          if prediksi_cocok == 10:
15            break
16          else:
```

```
17            for lemma in kamus:
18              if lemma["phrase"] == prediksi[0]:
19                posisi = prediksi_cocok
20                lv =
21  textdistance.levenshtein.normalized_similarity(word,
22  prediksi[0])
23                jw =
24  textdistance.jaro_winkler.normalized_similarity(word,
25  prediksi[0])
26                csv_writer.writerow([
27                  word,
28                  lemma["phrase"],
29                  posisi
30                  ])
31                prediksi_cocok += 1
32                print(word, lemma["phrase"])
33                break
34        print(f'Found: {prediksi_cocok}/10, for:
35  "{word}"')
36    print(f"Selesai Output {OUTPUT_FILE_PATH}")
37    return 0
38
39
40
```

Kode 5.23 Potongan Kode Pengujian 100 Kata Non-Kamus

Kode 5.23 menunjukkan proses pengujian 100 kata non-kamus yang paling sering muncul untuk model yang terpilih. Untuk setiap kata yang ada di variabel *daftar_cek* (kode 5.22 baris 2), model akan memunculkan 500 kata terdekat berdasarkan vektor dalam model. Apabila ditemukan kata yang termasuk di dalam kamus, maka akan dicetak dan dimasukkan ke dalam file csv beserta posisinya dari 10 kandidat yang akan dicetak. Jika kata yang cocok di kamus sudah mencapai 10, perulangan akan berhenti.

5.4.3.3 Pengujian 1000 Kata Non-Kamus yang Paling Sering Muncul

Untuk melakukan pengujian ini, digunakan kode yang hampir sama dengan pengujian 100 kata pada subbab sebelumnya. Perbedaanya adalah pada pengecekan kata. Pada pengujian kali ini, dtiambahkan variabel skor *Levenshtein's distance* dan *Jaro-Winkler distance*.

```
1     print("Mulai kalkulasi...")
```

```
2     with open(OUTPUT_FILE_PATH, mode="w") as
3   output_file:
4       csv_writer = csv.writer(output_file,
5   delimiter=',', quotechar='"',
6   quoting=csv.QUOTE_MINIMAL)
7       # Cek tiap kata
8       for word in daftar_cek:
9         prediksi_cocok = 0
10        # Generate prediksi
11        hasil_prediksi = new_model.most_similar(word,
12  topn = 500)
13        for prediksi in hasil_prediksi:
14          if prediksi_cocok == 10:
15            break
16          else:
17            for lemma in kamus:
18              if lemma["phrase"] == prediksi[0]:
19                posisi = prediksi_cocok
20                lv =
21  textdistance.levenshtein.normalized_similarity(word,
22  prediksi[0])
23                jw =
24  textdistance.jaro_winkler.normalized_similarity(word,
25  prediksi[0])
26                csv_writer.writerow([
27                  word,
28                  lemma["phrase"],
29                  lv,
30                  jw,
31                  posisi,
32                  (((1-(posisi/10))*0.5) + (lv*0.25) +
33  (jw*0.25))*100])
34                prediksi_cocok += 1
35                print(word, lemma["phrase"])
36                break
37        print(f'Found: {prediksi_cocok}/10, for:
38  "{word}"')
39    print(f"Selesai Output {OUTPUT_FILE_PATH}")
40    return 0
```

Kode 5.24 Potongan Kode Pengujian 1000 Kata Non-Kamus

Pada kode 5.24, selain dilakukan pengecekan untuk kata yang sama dengan kamus, kata yang diujikan juga diberikan skor berdasarkan posisi kemunculannya, skor *Levenshtein's Distance*, dan skor *Jaro-Winkler distance* (baris 26-33). Hal ini nantinya akan digunakan untuk menentukan *threshold* pengujian *tweet*.

5.4.3.4 Pembobotan Skor

Skor yang diberikan sebelumnya, yaitu skor posisi, skor *Levenshtein's dinstance, dan Jaro-Winkler distance*, akan diberi pembobotan untuk menilai hasil prediksi kata. Pembobotan yang diberikan adalah sebagai berikut.

```
Skor total = ((0.5 * skor posisi) + (0.25 * skor Lev)
+ (0.25 * skor Jw)) * 100
```

Kode 5.25 Pembobotan Skor

Skor posisi diberi bobot yang paling besar karena umumnya posisi kemunculan kata prediksi menggambarkan kedekatan kata tersebut terhadap kata yang diujikan. Sedangkan kedua skor lainnya diberi bobot yang sama karena keduanya merupakan skor pendukung yang memiliki kelebihannya masing-masing dalam menghitung kedekatan jarak antara dua *string*.

5.4.4 Pembuatan Sistem Normalisasi Teks

```
1           def _check_kamus(self, kata):
2               if kata in self.kamus:
3                   return {'found_in': 'kamus', 'result':
4       kata}
5               else:
6                   return None
7
8           def _check_kamus_mapping(self, kata):
9               x = self.kamus_mapping.get(kata)
10              if x is None:
11                  return None
12              else:
13                  return {'found_in': 'kamus_mapping',
14      'result' : x}
15
16          def _predict_from_model(self, kata, treshold,
17      gen_limit, match_limit):
18              hasil =  {'found_in':'model', 'result' : []}
19              generated_prediksi =
20      self.model.most_similar(kata, topn=gen_limit)
21              for prediksi in generated_prediksi:
22                  if len(hasil['result']) < match_limit:
23                      # Find match in loaded kamus
24                      for kata_kamus in self.kamus:
25                          if prediksi[0] == kata_kamus:
26                              posisi = len(hasil['result'])
27
```

```
28                              lv =
29      textdistance.levenshtein.normalized_similarity(kata,
30      prediksi[0])
31                              jw =
32      textdistance.jaro_winkler.normalized_similarity(kata,
33      prediksi[0])
34                              score = (((1 - ( posisi / 10
35      )) * 0.5 ) + ( lv * 0.25 ) + ( jw * 0.25 )) * 100
36
37                              # Check if satisfy treshold
38                              if score >= treshold:
39                                  hasil['result'].append({
40                                      'prediksi':
41      prediksi[0],
42                                      'posisi' : posisi,
43                                      'lv' : lv,
44                                      'jw' : jw,
45                                      'score' : score
46                                  })
47                              break
48                          else:
49                              continue
50                  else:
51                      break
52              if hasil['result']:
53                  hasil['result'] = sorted(hasil['result'],
54      key=itemgetter('score'), reverse=True)
55                  return hasil
56              else:
57                  return None
```

Kode 5.26 Potongan Kode *Function* Utama Normalisasi Teks

Kode 5.26 merupakan *function* dari 3 proses penting dari sistem normalisasi teks. *Function check_kamus()* (baris 1) akan mencetak kata masukan jika cocok dengan salah satu kata di kamus. *Function check_kamus_mapping()* (baris 8) akan mencetak kata pembenaran dari kamus mapping apabila kata input cocok dengan salah satu kata kunci di kamus mapping. Dan *function predict_from_model()* (baris 16) akan men-*generate* hasil prediksi dari model seperti sebelumnya. Jika ditemukan kata prediksi model yang cocok di kamus, maka akan dihitung skornya terlebih dahulu. Jika skor sama dengan atau melebihi *threshold*, dan skor merupakan skor tertinggi, maka kata tersebut akan dicetak.

```
        def _process_kata(self, kata, treshold, gen_limit,
    match_limit):
            kata = str(kata).lower()
            if len(kata) == 1:
                return {'found_in': 'satu_karakter',
    'result': kata}
            if kata.isdigit():
                return {'found_in': None, 'result': kata}
            res = self._check_kamus(kata)
            if res is None:
                res = self._check_kamus_mapping(kata)
                if res is None:
                    res = self._predict_from_model(kata,
    treshold, gen_limit, match_limit)

            if res is None:
                return {'found_in': None, 'result': kata}
            else:
                return res
```

Kode 5.27 Potongan Kode *Function* Pemrosesan Kata

Kode 5.27 merupakan *function* yang memproses kata masukan secara berurutan. Apabila kata tersebut hanya terdiri dari 1 karakter , maka kata masukan akan dicetak kembali. Kemudian jika kata masukan berupa angka, kata masukan akan dicetak kembali. Jika kata masukan bukan berupa angka atau 1 karakter, sistem akan memanggil *function* sebelumnya, yaitu mengecek kamus. Jika hasil tidak dtiemukan, maka sistem akan mengecek kamus mapping. Jika hasil tidak ditemukan, maka sistem akan memprediksi dari model. Jika hasil masih tidak ditemukan, maka sistem akan mencetak kembali kata masukan tersebut.

5.4.5 Pengujian Sistem Normalisasi Teks

Pengujian sistem dilakukan dengan menguji file *tweet* terhadap sistem.

```
    def true_main():
        print('Load model..')
        KataProcessor = kata_handler.KataProcessor()
        dict_kunci_jawaban = {}
        print('Load kunci..')
        row_counter = 1
        with open('kunci_jawaban2.csv', 'r', encoding='utf-
    8') as kunci_jawaban:

```

```
10          reader = csv.reader(kunci_jawaban,
11  delimiter=',')
12          dict_kunci_jawaban = {}
13          for row in reader:
14              dict_kunci_jawaban.update({
15                  row_counter : {
16                      "tweet_id": str(row[0]),
17                      "kata": str(row[1]),
18                      "kunci": str(row[2])
19                      }
20                  })
21              row_counter += 1
22
23      for treshold in TRESHOLD:
24          print(f'Process treshold {treshold} ...')
25          # Open CORE_ANALISIS_55
26          with open('core_analisis_55.csv', 'r',
27  encoding='utf-8') as file_kata:
28              file_kata_reader = csv.reader(file_kata,
29  delimiter=',')
30              # Open CORE_ANALISIS_TRESHOLD
31              with
32  open(f'out/core_analisis_{treshold}.csv', 'w',
33  encoding='utf-8', newline='') as savefile:
34                  savefile_writer = csv.writer(savefile,
35  delimiter=',')
36
37                  row_counter = 1
38                  for row in file_kata_reader:
39                      _ = handle_row(row, KataProcessor,
40  dict_kunci_jawaban, treshold, row_counter)
41                      print(f'TRESHOLDâ€“{treshold} ' + "
42  | ".join([str(x) for x in _]))
43                      savefile_writer.writerow(_)
44                      row_counter += 1
```

Kode 5.28 *Function true_main*

Kode 5.28 menunjukkan *function* utama yang akan memuat kunci jawaban, lalu memanggil *function handle_row* yang akan memproses file uji, membaca data uji, menyimpan hasil pengujian, dan mengulangi pengujian untuk setiap threshold yang ditentukan.

```
1   def handle_row(row, kata_processor, kunci_jawaban,
2   treshold, row_number):
3       if row[5].lower() in ['kamus_mapping',
4   'satu_karakter']:
5           return _handle_skip(row)
```

```
6         elif row[5].lower() in ['kamus']:
7             return _handle_kamus(row, kunci_jawaban,
8     row_number)
9         elif row[5].lower() in ['treshold_unsatisfied',
10    'model_kata_dasar', 'model']:
11            return _handle_reprocess(row, kata_processor,
12    kunci_jawaban, treshold, row_number)
```

Kode 5.29 *Function handle_row*

Kode 5.29 menunjukkan *function* dimana jika kata masukan ada di kamus mapping, atau terdiri dari 1 karakter, maka akan memanggil *function handle_skip()*. Jikda kata masukan ada di kamus, maka akan memanggil *function handle_kamus()*. Jika kata masukan perlu proses dari model, maka akan memanggil *function handle_reprocess()*.

```
1     def _handle_skip(row):
2         kata = row[1]
3         jawaban = row[2]
4         if row[5].lower() in ['satu_karakter']:
5             return [row[0], kata, jawaban, row[1], '-', '-
6     ', row[5], 7]
7         elif row[5].lower() in ['kamus_mapping']:
8             return [row[0], kata, jawaban, row[2], '-', '-
9     ', row[5], 7]
```

Kode 5.30 *Function handle_skip*

Kode 5.30 menunjukkan *function* dimana jika kata masukan merupakan satu karakter, maka akan mencetak nomor tweet, kata input, kata kamus, dan koreksi manualnya. Dan jika hasil koreksi berasal dari kamus mapping, maka akan mencetak nomor tweet, kata input, kata koreksi kamus mapping, dan koreksi manualnya.

```
1     def _handle_kamus(row, kunci_jawaban, row_number):
2         kata = row[1].lower()
3         kunci =
4     kunci_jawaban[row_number].get('kunci').lower()
5         jawaban = row[2]
6         if kata == kunci:
7             return [row[0], kata, jawaban, kunci, '-', '-',
8     row[5], 7]
9         else:
```

```
10          return [row[0], kata, jawaban, kunci, '-', '-',
11  row[5], 6]
```

Kode 5.31 *Function handle_kamus*

Kode 5.31 menunjukkan *function* dimana jika kata masukan ada di kamus, maka akan mencetak nomor tweet, kata input, dan koreksi manualnya. Jika kata kamus sesuai dengan koreksi manual, maka akan diberikan angka 7, namun jika tidak sesuai maka diberi angka 6. Hal ini digunakan untuk kategorisasi hasil pengujian nanti.

```
1   def _handle_reprocess(row, kata_processor,
2   kunci_jawaban, treshold, row_number):
3       stem = kata_processor.stemmer.stem
4       kata, kata_stemmed, kunci, kunci_stemmed =
5   row[1].lower(), stem(row[1]).lower(),
6   kunci_jawaban[row_number].get('kunci').lower(),
7   stem(kunci_jawaban[row_number].get('kunci')).lower()
8       result_model = kata_processor.process_kata(kata,
9   treshold=treshold, verbose=True)
10      if result_model['found_in'] == 'model':
11          for prediksi in result_model['result']:
12              kata_prediksi_stemmed =
13  stem(prediksi['prediksi'])
14              if kata_stemmed == kata_prediksi_stemmed:
15                  if kata_prediksi_stemmed in
16  kata_processor.kamus:
17                      if kata_prediksi_stemmed ==
18  kunci_stemmed:
19                          return [row[0], row[1],
20  kata_prediksi_stemmed, kunci, prediksi['posisi'],
21  prediksi['score'], 'model_kata_dasar', 1]
22                      else:
23                          return [row[0], row[1],
24  kata_prediksi_stemmed, kunci, prediksi['posisi'],
25  prediksi['score'], 'model_kata_dasar', 4]
26          top_data = result_model['result'][0]
27          if top_data['prediksi'] == kunci:
28              return [row[0], row[1],
29  top_data['prediksi'], kunci, top_data['posisi'],
30  top_data['score'], 'model', 2]
31          else:
32              return [row[0], row[1],
33  top_data['prediksi'], kunci, top_data['posisi'],
34  top_data['score'], 'model', 4]
35      elif result_model['found_in'] is None:
36          if result_model['result'] == kunci:
```

```
37                 return [row[0], row[1],
38     result_model['result'], kunci, '-', '-',
39     'treshold_unsatisfied', 3]
40             else:
41                 return [row[0], row[1],
42     result_model['result'], kunci, '-', '-',
43     'treshold_unsatisfied', 5]
```

Kode 5.32 *Function handle_reprocess*

Kode 5.32 menunjukkan *function* yang akan mengoreksi kata uji dengan prediksi dari model, lalu mencocokkannya dengan kata kunci.

5.4.6 Pembuatan Demo Sistem Normalisasi

Sistem demo normalisasi dibuat dengan inti sistem yang sama, hanya berbeda dalam menjalankan sistem dan masukannya.

```
1    def parse(inp):
2        i = inp.split()
3        if inp == 'VERBOSE':
4            STATE['VERBOSE'] = not STATE['VERBOSE']
5        elif i == []:
6            return 7
7        elif i[0] == 'SET' and len(i) == 3:
8            if i[1] == "TRESHOLD":
9                STATE[i[1]] = float(i[2])
10               return 0
11           else:
12               return 1
13       elif i[0] == 'DEMO':
14           c = DEMO()
15           if c == 0:
16               print("> DEMO END")
17               return "DEMO END"
18       elif i[0] == 'EXIT':
19           return 99
20       else:
21           return 1
```

Kode 5.33 *Function* Menu Utama

Pada menu utama, input dapat menerima 3 jenis masukan: *VERBOSE* untuk memunculkan atau menghilangkan detail koreksi; *SET THRESHOLD* untuk mengatur *threshold* yang akan digunakan ketika demo, dan *DEMO* untuk memulai demo.

```
def _parse_demo(strx):
    s = strx.split()
    if strx == 'VERBOSE':
        STATE['VERBOSE'] = not STATE['VERBOSE']
    elif s[0] == 'EXIT':
        return 0
    else:
        inp = TextPreprocessor.process(strx)
        print('> Processing...')
        if STATE['VERBOSE']:
            token_hasil = []
            verbose_to_print = []
            for kata in inp.split():
                token, row = _process_kata(kata)
                token_hasil.append(token)
                verbose_to_print.append(row)
            print("> ", " ".join(token_hasil))
            for row in verbose_to_print:
                print("> ", " | ".join([str(x) for x in
row]))

        else:
            print("> ", "
".join([str(KataProcessor.process_kata(kata,
treshold=STATE['TRESHOLD'], verbose=False)) for kata in
inp.split()]))
```

Kode 5.34 *Function* demo aplikasi

Setelah memulai demo, sistem akan menerima masukan dari pengguna berupa kalimat atau kata, dimana masukan tersebut kemudian akan diproses sama seperti ketika melakukan pengujian *tweet*, namun tidak memunculkan kata hasil koreksi manual.

BAB VI
HASIL DAN PEMBAHASAN

Pada bab ini dijelaskan hasil dan pembahasan berupa analisis dan pengujian aplikasi terhadap implementasi yang telah dilakukan.

6.1 Akuisisi Data

6.1.1 Hasil Akuisisi Data *Twitter*

Jumlah data *tweet* yang didapatkan dengan metode *crawling* mulai tanggal 20 Desember 2017 hingga 1 Mei 2017 adalah 62.873.706. Hasil ini nantinya akan berkurang seiring dengan adanya pra-proses data.

Tabel 6.1 Contoh *Tweet* Hasil *Crawling*

"@rlthingy Marah lah. Lw siapa menginvasi acc gw, yg sifatnya pribadi? Ga usah ngomongin dosa dulu, dari perspektif soal privasi aja udah salah."
"RT @ikhwan95: @CakKhum @SurYosodipuro_ @fadlizon Cebong ini kalau kalah debat nyerang pribadi dan teriak2 gak jelas mau ngomong apa!!"
"RT @AlfariRudi: Hello Saat ini Gerakan Hesteg #2019GantiPresiden Ada Saudara Kembarnya niich #17AprilPrabowoPresiden ,,,Yok Kita Gaungkan"
"RT @hadyan92: Siaann liatnya..menyerang pribadi saat diskusi menandakan kapal mulai tenggelam..ckckck..coba TKN evaluasi lah..maluuu"
"Ni yang paling gede yang kuliat tadi berantem-berantem ngalahin kepala geng berandal sekolahan. Astaga Kang Hyuk, rupanya kamu Kang Hoo. #LahGimana https://t.co/UWrvg0OaST"

6.2 Hasil Pra-proses Data

6.2.1 Hasil Penghapusan Data yang Duplikat

Data hasil *crawling* masih banyak menyisakan data duplikat. Walaupun tweet memiliki ID yang berbeda-beda, namun karena adanya fitur *retweet*, maka banyak terdapat tweet yang isinya sama, hanya ditambahi tanda "RT". *Crawler* tetap mendeteksi hal ini sebagai *tweet* dan mengambil data ini. Data yang duplikat dihilangkan dengan query yang telah disebutkan pada bab sebelumnya.

Tabel 6.2 Contoh *Tweet* Duplikat

ID Tweet	Message
1080876244240031744	"RT @XTaehyung_Kim: Enak ya punya kapel\nKalo bangun tidur ada kerjaan\nBuka dm isi spaman doi\nUcapan smalam pagi smbil cium sana sini"
1080875575013662720	"RT @XTaehyung_Kim: Enak ya punya kapel\nKalo bangun tidur ada kerjaan\nBuka dm isi spaman doi\nUcapan smalam pagi smbil cium sana sini"
1080875553924603905	"RT @XTaehyung_Kim: Enak ya punya kapel\nKalo bangun tidur ada kerjaan\nBuka dm isi spaman doi\nUcapan smalam pagi smbil cium sana sini"
1080866591238086656	"RT @_paay: Kadang suka kesel, kalo lagi ada waktu

	ketemu cuma dipake buat berantem aja. Udah gitu ketemunya cuma di dalam mimpi."
1080866036872667136	"RT @_paay: Kadang suka kesel, kalo lagi ada waktu ketemu cuma dipake buat berantem aja. Udah gitu ketemunya cuma di dalam mimpi."
1080865394615771136	"RT @_paay: Kadang suka kesel, kalo lagi ada waktu ketemu cuma dipake buat berantem aja. Udah gitu ketemunya cuma di dalam mimpi."

Setelah dilakukan penghapusan data duplikat, tersisa 16.296.013 *tweet*. Pengurangan data yang cukup signifikan.

6.2.2 Hasil Pra-proses Teks

Pra-proses data yang dilakukan antara lain *case folding*, *stripping* untuk menghilangkan spasi yang berlebih, membersihkan URL, tagar, *mention, RT, FAV,* emoji, dan *smiley*, mengubah tanda baca menjadi spasi, serta menghilangkan string “http” dan “https”, untuk meminimalisir kemungkinan bahwa kedua unsur tersebut masih muncul dalam teks, walaupun telah dilakukan penghapusan url.

Tabel 6.3 Contoh Hasil Pra-Proses Teks

***Tweet* asli**	RT @CERITA48: [C48] Kerecehan yang hakiki ikut ketawa terus wkwk https://t.co/mMnKbHfFOJ

Case Folding	rt @cerita48: [c48] kerecehan yang hakiki ikut ketawa terus wkwk https://t.co/mmnkbhffoj
Pembersihan dengan method clean_tweet()	[c48] kerecehan yang hakiki ikut ketawa terus wkwk
Penghilangan tanda baca	c48 kerecehan yang hakiki ikut ketawa terus wkwk
Stripping	c48 kerecehan yang hakiki ikut ketawa terus wkwk

6.3 Hasil Data Kamus

6.3.1 Kamus Utama

Data kamus utama berasal dari hasil *crawling* dari situs *Kateglo.* Data yang didapat berjumlah 72.254 baris, dimana masing-masing baris terdiri dari kata, kata aktual, dan arti. Data ini disimpan dalam sebuah file .txt berbentuk JSON

Tabel 6.4 Contoh Hasil Akuisisi Data *Kateglo*

{"phrase": "abuh", "actual_phrase": null, "arti": "(Jw) bengkak; riuh; sibuk;"}
{"phrase": "ilusi", "actual_phrase": null, "arti": "sesuatu yang hanya dalam angan-angan; khayalan; pengamatan yang tidak sesuai dengan pengindraan; tidak dapat dipercaya; palsu;"}
{"phrase": "inas", "actual_phrase": null, "arti": "bisul pada tengkuk;"}

6.3.2 Kamus Mapping

Data kamus mapping berasal dari penelitian sebelumnya[7], yang awalnya berjumlah 5.325 baris, ditambah dengan hasil koreksi pengujian model terhadap 1000 kata non-kamus yang paling sering muncul. Setelah ditambahkan, data koreksi yang

duplikat kemudian dihapus. Sehingga kini data berjumlah 5.736 kata non-kamus yang telah memiliki koreksi manualnya masing-masing.

6.3.3 Pembahasan Data Kamus

Jumlah data kamus yang dimiliki adalah 72.254 dari kamus utama ditambah dengan 5.736 data dari kamus mapping. Sehingga totalnya menjadi 77.990 baris kata. Dalam pemuatan kamus, terdapat beberapa kendala, seperti adanya ragam cakapan seperti “gue” dan “lo”. Sehingga ketika dilakukan pengujian, kata-kata tersebut dideteksi sebagai kata baku. Untuk meminimalisir hal ini, dilakukan penyaringan dengan tidak mengikutsertakan kata yang memiliki “(cak)” pada kolom arti.

6.4 Hasil Model

6.4.1 Kemunculan Kata Non-Kamus

Dari proses pengambilan 1000 kata non kamus, dapat dilihat jumlah kemunculan masing-masing kata. Sebagai contoh, berikut 10 kata non-kamus yang paling sering muncul.

Tabel 6.5 Sepuluh Kata Non-Kamus yang Paling Sering Muncul

No.	Kata	Jumlah
1	yg	3091902
2	rt	2406620
3	gak	1722845
4	ga	1406476
5	udah	944084
6	nya	836812
7	gimana	679805
8	gitu	515454
9	indonesia	408688
10	jokowi	396590

6.4.2 Pembahasan Kata Non-Kamus

Kata-kata non-kamus yang muncul didominasi oleh kata-kata slang, singkatan, ragam cakapan, merek, nama tempat, dan akronim. Hal ini disebabkan karena data diambil dari *platform Twitter*, yang mana merupakan media sosial dengan pengguna yang beragam. Selain itu, pada *platform* ini terdapat batas karakter untuk satu *tweet*. Sehingga wajar saja jika pengguna menggunakan kata-kata yang disingkat, untuk menghindari kelebihan batas dalam pengiriman *tweet.*

Kata-kata ini diambil untuk dua pengujian. Pengujian pertama menggunakan 100 kata non-kamus teratas, yang dilakukan untuk memilih model terbaik yang akan digunakan sebagai model untuk sistem normalisasi. Pengujian kedua menggunakan 1000 kata non-kamus teratas, yang dilakukan untuk menentukan *threshold* pengujian data sampel *tweet*. Berikut 100 kata yang digunakan untuk pengujian pertama.

Tabel 6.6 Seratus Kata Non-Kamus Teratas

yg	pake	jd	udh	karna
rt	wkwk	bgt	ngga	lg
gak	mikir	pengen	bener	hahaha
ga	tp	dgn	utk	sm
udah	tuh	nonton	wkwkwk	gt
nya	dm	gini	dr	jakarta
gimana	gw	tu	temen	rp
gitu	sampe	klo	tdk	yah
indonesia	nggak	org	jg	soalnya
jokowi	allah	yaa	amp	tuhan

haha	namanya	hmm	minggu	engga
hehe	hp	prabowo	blm	in
jgn	ntar	oppa	nanya	malem
dimana	ngomong	twitter	dg	dlm
gk	katanya	tks	pilkada	beneran
krn	hyung	2019gantipresiden	tetep	internet
trus	islam	apapun	the	yaudah
sebuah	gatau	km	april	dilakukan
disini	sdh	skrg	lakukan	kayaknya
dapet	sy	mulu	je	gabisa

6.4.3 Hasil Pengujian Model FastText

Model-model yang telah dibuat akan diujikan dengan 100 kata non-kamus yang telah disebutkan sebelumnya. Setiap kata tersebut akan dicari prediksi pembenarannya untuk masing-masing model. Model akan memunculkan 10 kandidat kata pembenaran yang sesuai dengan kamus. Untuk menghitung akurasi model, kata yang benar akan dipilih, dan dihitung skornya dengan mempertimbangkan posisi kemunculannya dari 10 kandidat kata yang dimunculkan. Rumus yang digunakan adalah sebagai berikut:

$$A = \frac{1}{100} * \sum_{i=1}^{100} \left(1 - \left(\frac{i}{10}\right)\right)$$

Dimana akurasi (A) adalah rata-rata jumlah skor posisi ditemukannya kata kandidat yang benar, yang ditandai dengan (i).

6.4.3.1 Pengujian modelV1_ft

Pengujian 100 kata terhadap modelV1_ft mendapatkan akurasi sebesar 36,30%. Model ini memiliki konfigurasi seperti pada tabel 6.7.

Tabel 6.7 Konfigurasi Parameter *Training* modelV1_ft

Parameter	Nilai
Dimension Size	300
Training Algorithm	CBOW
Minimum word freq.	5
Context window	7
Iterasi	2
Minimum character n-gram	3
Maximum character n-gram	6

6.4.3.2 Pengujian modelV2_ft

Pengujian 100 kata terhadap modelV2_ft mendapatkan akurasi sebesar 32,80%. Model ini memiliki konfigurasi seperti pada tabel 6.8.

Tabel 6.8 Konfigurasi Parameter *Training* modelV2_ft

Parameter	Nilai
Dimension Size	300
Training Algorithm	Skip-gram
Minimum word freq.	5
Context window	7

Iterasi	2
Minimum character n-gram	3
Maximum character n-gram	6

6.4.3.3 Pengujian modelV3_ft

Pengujian 100 kata terhadap modelV3_ft mendapatkan akurasi sebesar 45,60%. Model ini memiliki konfigurasi seperti pada tabel 6.9.

Tabel 6.9 Konfigurasi Parameter *Training* modelV3_ft

Parameter	Nilai
Dimension Size	300
Training Algorithm	CBOW
Minimum word freq.	5
Context window	5
Iterasi	2
Minimum character n-gram	3
Maximum character n-gram	6

6.4.3.4 Pengujian modelV4_ft

Pengujian 100 kata terhadap modelV4_ft mendapatkan akurasi sebesar 41,70%. Model ini memiliki konfigurasi seperti pada tabel 6.10.

Tabel 6.10 Konfigurasi Parameter *Training* modelV4_ft

Parameter	Nilai
Dimension Size	300
Training Algorithm	Skip-gram

Minimum word freq.	5
Context window	5
Iterasi	2
Minimum character n-gram	3
Maximum character n-gram	6

6.4.4 Hasil Pengujian Model Word2Vec

Akurasi model *word2vec* dihitung dengan cara yang sama dengan penghitungan akurasi model *fasttext*.

6.4.4.1 Pengujian modelV1_w2v

Pengujian 100 kata terhadap modelV1_w2v mendapatkan akurasi sebesar 31,00%. Model ini memiliki konfigurasi seperti pada tabel 6.11.

Tabel 6.11 Konfigurasi Parameter *Training* modelV1_w2v

Parameter	Nilai
Dimension Size	300
Training Algorithm	Skip-gram
Minimum word freq.	5
Context window	7
Iterasi	2

6.4.4.2 Pengujian modelV2_w2v

Pengujian 100 kata terhadap modelV2_w2v mendapatkan akurasi sebesar 42,50%. Model ini memiliki konfigurasi seperti pada tabel 6.12.

Tabel 6.12 Konfigurasi Parameter *Training* modelV2_w2v

Parameter	Nilai

Dimension Size	300
Training Algorithm	Skip-gram
Minimum word freq.	5
Context window	5
Iterasi	2

6.4.4.3 Pengujian modelV3_w2v

Pengujian 100 kata terhadap modelV3_w2v mendapatkan akurasi sebesar 44,60%. Model ini memiliki konfigurasi seperti pada tabel 6.13.

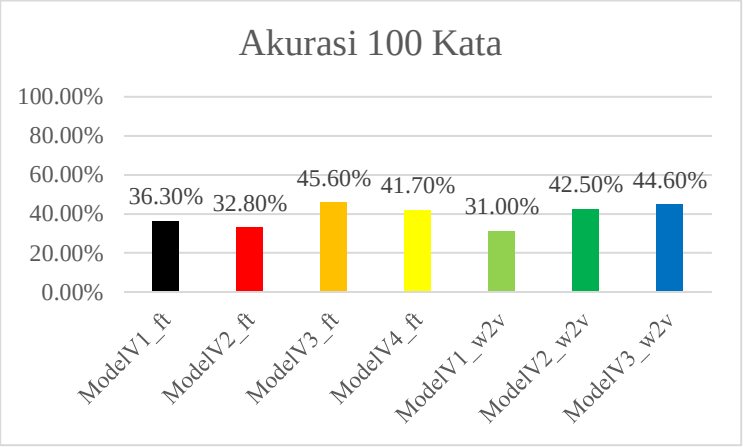
Tabel 6.13 Konfigurasi Parameter *Training* modelV3_w2v

Parameter	Nilai
Dimension Size	300
Training Algorithm	CBOW
Minimum word freq.	5
Context window	5
Iterasi	2

6.4.5 Pembahasan Hasil Pengujian Model

6.4.5.1 Model Terbaik

Dari 7 percobaan yang dilakukan, akurasi setiap model dapat dilihat pada gambar 6.1.



Gambar 6.1 Akurasi Model untuk Pengujian Pertama

Berdasarkan gambar 6.1, model dengan akurasi terbaik adalah modelV3_ft.

6.4.5.2 Parameter Penting *Training* Model

Parameter yang memiliki pengaruh paling besar pada 7 percobaan model adalah *context window*, dimana model yang menggunakan parameter *context window=5* memiliki akurasi sekitar 9% lebih baik daripada yang menggunakan parameter *context window=7*. Sedangkan perbedaan algoritma *training* antara CBOW dan *Skip-gram* menunjukkan perbedaan sekitar 4% lebih tinggi untuk algoritma CBOW.

6.5 Pengujian Sistem Normalisasi Teks

6.5.1 Hasil Pengujian Seribu Kata Non-Kamus

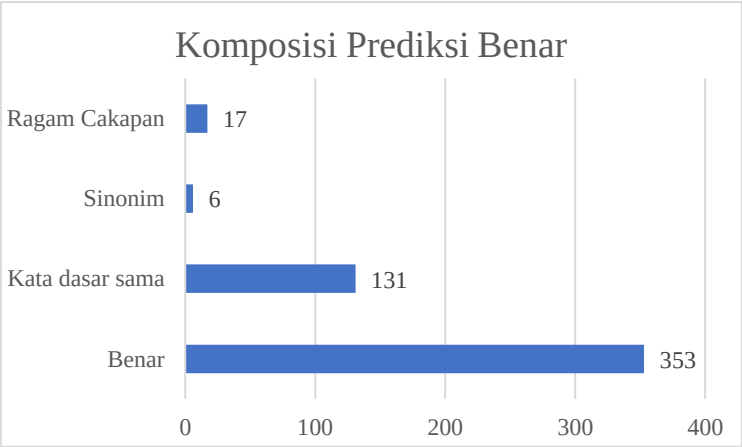
Model yang terpilih akan diujikan kembali dengan 1000 kata non-kamus yang paling sering muncul. Pengujian dilakukan dengan memunculkan paling banyak 10 kandidat kata beserta skor masing-masing kandidat kata sesuai dengan yang telah dijelaskan pada bab sebelumnya. Total prediksi yang dilakukan adalah sebanyak 5889 kali, karena tidak semua kata yang dikoreksi berhasil memunculkan 10 kandidat yang sesuai kamus. Dalam prediksi tersebut, model berhasil mengoreksi

507 kata dengan benar, dan 5381 sisanya merupakan prediksi yang salah. Hasil prediksi dalam pengujian dibagi menjadi 5 kategori, dan diberi angka 0-4. Kategori hasil pengujian adalah sebagai berikut.

Tabel 6.14 Kategori Hasil Pengujian

Kategori	**Keterangan**
Salah	0
Koreksi benar	1
Mengandung kata dasar yang sama	2
Merupakan sinonim	3
Ragam cakapan	4

Komposisi prediksi yang benar dapat dilihat pada gambar 6.2.



Gambar 6.2 Komposisi Hasil Prediksi Benar

kata	prediksi	skor_lv	skor_jw	skor_p	skor_total	keterang
denger	dengar	0.833333333	0.933333333	1	94.16666667	1
pinter	pintar	0.833333333	0.933333333	1	94.16666667	1
manggil	panggil	0.857142857	0.904761905	1	94.04761905	2
kaga	kagak	0.8	0.96	1	94	4
idol	idola	0.8	0.96	1	94	1
tiba2	tiba	0.8	0.96	1	94	1
masi	masih	0.8	0.96	1	94	1
anak2	anak	0.8	0.96	1	94	1

Gambar 6.3 Contoh Hasil Pengujian 1000 Kata Non-Kamus

Gambar 6.3 menunjukkan contoh hasil pengujian. Terdapat 7 kolom dalam gambar tersebut, kolom *kata* adalah kata non-kamus yang diujikan. Kolom *prediksi* adalah kata hasil prediksi model. Kolom *skor_lv* adalah skor *Levensthein's distance* antara *kata* dengan *prediksi*. Kolom *skor_jw* adalah skor *Jaro-Winkler distance* antara *kata* dengan *prediksi*. Kolom *skor_pos* adalah skor posisi ditemukannya kandidat kata sesuai kamus. Kolom *skor_total* adalah total skor setelah dihitung dengan bobot masing-masing. Dan kolom *keterangan* adalah kategori koreksi hasil uji.

6.5.2 Pembahasan Hasil Pengujian 1000 Kata Non-Kamus

6.5.2.1 Penentuan Nilai *Threshold*

Dari hasil pengujian, untuk setiap kategori koreksi yang benar akan dapat dilihat skor tertinggi dan terendah. Berikut datanya.

Tabel 6.15 Nilai Tertinggi dan Terendah Hasil Uji

kategori benar	1	2	3	4	rata-rata
MAX	96.67	94.54	87.32	94	93.13
MIN	25	42.78	40	54.44	40.56

Rata-rata skor terendah akan dijadikan *threshold* untuk pengujian data sampel *tweet*. Nilai *threshold* ini digunakan untuk menentukan batasan nilai hingga sebuah kata dapat dikatakan berhasil dikoreksi. Namun nilai *threshold* 40% dirasa terlalu rendah untuk pengujian. Dikhawatirkan dengan nilai tersebut akan terlalu banyak kata yang dikoreksi benar oleh model, namun pada kenyataannya koreksi tersebut salah. Untuk

mengurangi kemungkinan terjadinya hal tersebut, maka nilai *threshold* akan dinaikkan sesuai dengan rata-rata terendah untuk kategori 1 dan 4, dengan syarat nilai koreksi yang diambil hanyalah yang memiliki skor posisi 1,00. Dengan kata lain, hasil koreksi yang menempati urutan pertama dalam kemunculan kandidat kata. Tabel 6.16 menunjukkan nilai terendah dengan filter tersebut.

Tabel 6.16 Tabel Nilai Minimum Setelah *Filter*

kategori benar	1	4	rata-rata
MIN	50	69.44	59.72

Nilai rata-rata 60% jika dibulatkan dari 59,72% dirasa cukup untuk dijadikan patokan *threshold* awal. Dan untuk menambah reliabilitas hasil percobaan, maka akan dilakukan beberapa percobaan dengan nilai *threshold* lain, yaitu 55%, 60%, 65%, 70%, 80%, dan 90%. Diharapkan dengan adanya percobaan-percobaan ini, akan dapat melihat perbandingan *threshold* yang lebih cocok untuk digunakan dalam sistem normalisasi teks.

6.5.2.2 Penambahan Kata dalam Kamus Mapping

Telah dijelaskan sebelumnya bahwa kamus mapping adalah kamus untuk kata non-kamus yang telah diberi koreksi kata yang benar. Dan salah satu sumber untuk kamus mapping adalah percobaan ini. Dari data hasil koreksi percobaan ini, kata dan koreksi yang benar akan dimasukkan ke dalam kamus mapping.

6.5.3 Hasil Pengujian pada Data Sampel *Tweet*

Dari Tabel 6.17 dapat disimpulkan bahwa nilai *threshold* dengan akurasi terbaik adalah 55%, 60%, dan 65%. Terdapat 4 nilai threshold yang memiliki hasil koreksi terbanyak (328 kata) dengan kata dasar yang sama dengan koreksi manual, yaitu 55%, 60%, 65%, dan 70%, namun yang memiliki koreksi terbanyak (143 kata) yang sama persis dengan koreksi manual adalah pada *threshold* 55%, 60% dan 65%. Meskipun demikian, nilai *threshold* 55% memiliki jumlah kata terbanyak untuk kategori pengoreksian yang salah walaupun memenuhi *threshold*, dengan jumlah 1125 kata.

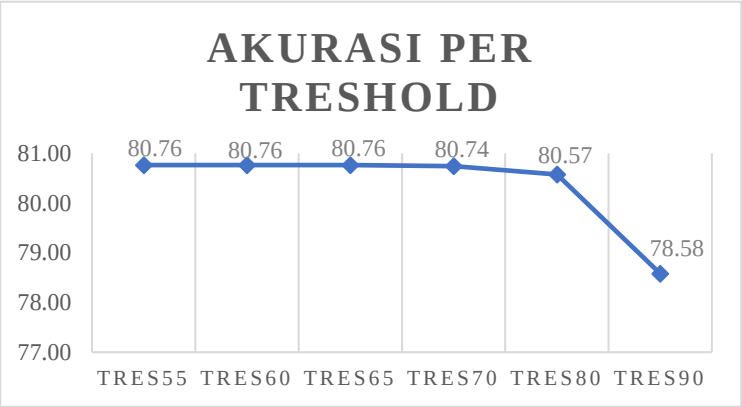
Di sisi lain, kegagalan yang paling besar ada pada nilai *threshold* 90% dengan akurasi 78,58% . Dimana Hasil prediksi yang tidak memenuhi *threshold* namun cocok dengan koreksi manual berjumlah 633 kata, dan yang tidak memenuhi *threshold* dan tidak cocok dengan koreksi manual berjumlah 446 kata.

Rincian dari hal ini akan dijelaskan pada sub bab berikutnya.

Tabel 6.17 Hasil Pengujian Data Sampel *Tweet*

Berhasil dikoreksi:	**T55**	**T60**	**T65**	**T70**	**T80**	**T90**
Dikoreksi ada pada Kamus/Kamus mapping	6712	6712	6712	6712	6712	6712
Kata berhasil dikoreksi model dan memiliki kata dasar yang sama dengan koreksi manual	328	328	328	328	323	188
Dikoreksi model dan sesuai dengan koreksi manual	143	143	143	141	131	89
Total	**7183**	**7183**	**7183**	**7181**	**7166**	**6989**
Gagal dikoreksi:						
Tidak memenuhi *threshold* tapi cocok dengan koreksi manual	168	175	188	251	348	633
Berhasil dikoreksi, tapi tidak cocok dengan koreksi manual	1125	1109	1091	1020	855	462
Tidak memenuhi *threshold* dan tidak sesuai koreksi manual	54	63	68	78	161	446
Ada di kamus, namun tidak sesuai koreksi manual	364	364	364	364	364	364
Total	**1365**	**1365**	**1368**	**1370**	**1385**	**1553**
Rata-rata	**80.76**	**80.76**	**80.76**	**80.74**	**80.57**	**78.58**

6.5.4 Pembahasan Hasil Pengujian



Gambar 6.4 Grafik Perbandingan Akurasi Hasil Uji Per *Threshold*

Dari Gambar 6.4, dapat dilihat bahwa akurasi setiap threshold tidak berbeda terlalu jauh dari tres55-tres70, sedangkan pada tres80 mulai terlihat perbedaan, dan pada tres90 akurasi mengalami penurunan yang cukup jauh.

Hal ini disebabkan oleh kegagalan model dalam memprediksi kata jika threshold terlalu tinggi. Namun hal tersebut dapat mengurangi hasil prediksi model yang salah. Di sisi lain, jika threshold terlalu rendah, maka semakin besar kemungkinan model untuk dapat memprediksi suatu kata, namun semakin besar pula kemungkinan prediksi kata tersebut adalah prediksi yang salah.

Tabel 6.18 Contoh Prediksi Salah

Token	dinolovers
Koreksi Manual	dinolovers
Tres 55	kers diprediksi, salah
Tres 60	dinolovers tidak ada di kamus
Tres 65	dinolovers tidak ada di kamus
Tres 70	dinolovers tidak ada di kamus
Tres 80	dinolovers tidak ada di kamus
Tres 90	dinolovers tidak ada di kamus

Sebagai contoh adalah pada Tabel 6.18. Karena *threshold* 55% terlalu rendah, maka sistem dapat menghasilkan kata prediksi, namun koreksi yang dihasilkan salah. Dan untuk nilai *threshold* 60-90% kata tersebut tidak dapat diprediksi karena tidak memenuhi threshold dan tidak ada di kamus.

Tabel 6.19 Contoh Kata Dasar Sama

Token	ditandai
Koreksi Manual	ditandai
Tres 55	tanda Kata dasar sama
Tres 60	Tanda Kata dasar sama
Tres 65	Tanda Kata dasar sama
Tres 70	Tanda Kata dasar sama
Tres 80	Dituai Diprediksi, salah
Tres 90	Ditandai tidak ada di kamus

Pada contoh di atas, nilai *threshold* 55-70% dapat memprediksi kata dan memiliki kata dasar yang sama dengan koreksi manual. Namun pada tres80 prediksi model salah. Artinya posisi prediksi tertinggi untuk kata "ditandai" adalah "dituai", dan pada tres90 kata tersebut dicetak kembali karena tidak ada hasil prediksi model yang muncul dengan *threshold* ≥ 90%.

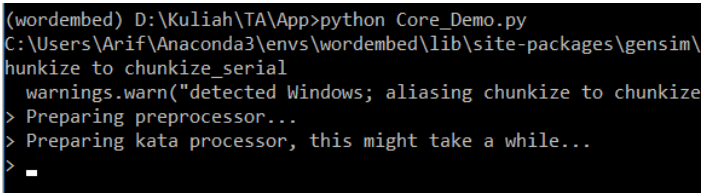
Tabel 6.20 Contoh Prediksi Sama

Token	syg
Koreksi Manual	sayang
Tres 55	sayang Prediksi sama
Tres 60	sayang Prediksi sama
Tres 65	sayang Prediksi sama
Tres 70	sayang Prediksi sama
Tres 80	syg tidak ada di kamus
Tres 90	syg tidak ada di kamus

Untuk contoh di atas, token *syg* dapat diprediksi dengan benar pada percobaan dengan nilai *threshold* 55-70%, namun untuk *threshold* 80-90% tidak berhasil diprediksi, karena skor prediksi kata *sayang* tidak lebih dari atau sama dengan 80%.

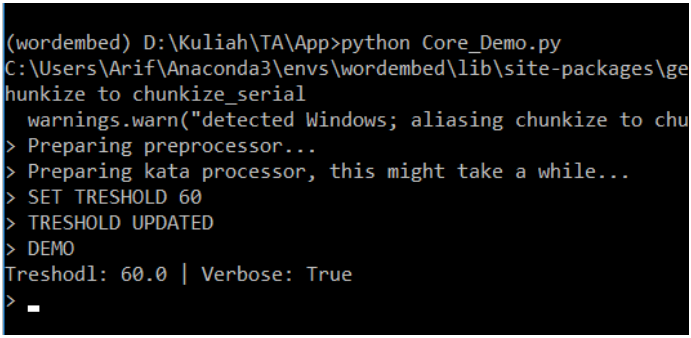
6.5.5 Hasil Aplikasi

Aplikasi yang dihasilkan berbasis *command line*. Pada aplikasi ini, sistem menerima masukan berupa kata/kalimat, yang kemudian akan dinormalisasi sesuai dengan sistem normalisasi yang digunakan.



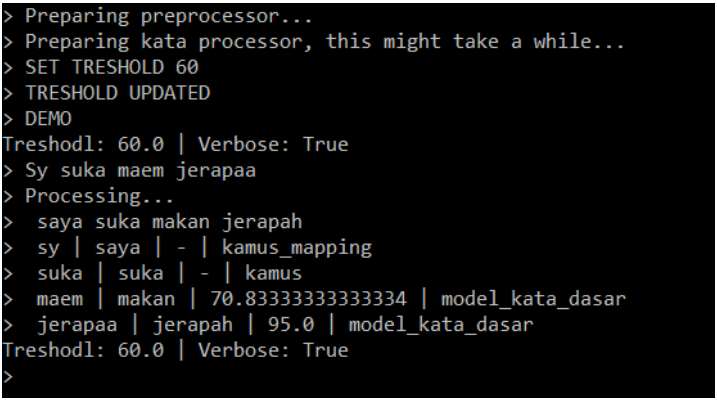
Gambar 6.5 Menu Utama Aplikasi

Pada saat aplikasi dijalankan, maka *preprocessor* dan *processor* kata akan dimuat, ini termasuk model yang digunakan, yaitu modelV3_ft. Pada tampilan awal, aplikasi ini menerima 3 perintah, “SET THRESHOLD ##” untuk mengatur threshold, “DEMO” untuk memulai demo, dan “EXIT” untuk keluar.



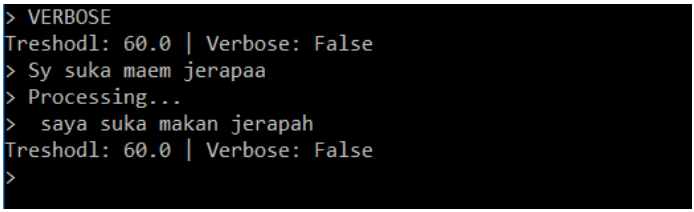
Gambar 6.6 Menu Demo

Setelah menu demo dijalankan, maka terlihat threshold yang digunakan dan status *verbose*. Jika tatus *verbose* adalah *true*, maka aplikasi akan menampilkan rincian hasil normalisasi. Pada tampilan ini, aplikasi menerima 3 masukan. “VERBOSE” untuk mengganti status *verbose*, “EXIT” untuk keluar, dan masukan berupa kata/kalimat yang akan dinormalisasi.



Gambar 6.7 Hasil Normalisasi dengan Status *Verbose True*

Gambar 6.7 menunjukkan hasil normalisasi kalimat "Sy suka maem jerapaa" secara rinci. Baris pertama menunjukkan hasil normalisasinya. Kemudian baris-baris setelahnya menunjukkan kata masukan, hasil koreksi kata, skor, dan asal koreksi kata. Jika status *verbose* adalah *false*, maka hasilnya akan jadi seperti di bawah ini.



Gambar 6.8 Hasil Normalisasi dengan Status *Verbose False*

Halaman ini sengaja dikosongkan

BAB VII
KESIMPULAN DAN SARAN

Pada bab ini dipaparkan kesimpulan dari pengerjaan tugas akhir yang telah dilakukan dan saran yang dapat diberikan sebagai pengembangan penelitian yang telah dilakukan.

7.1 Kesimpulan

Berdasarkan pengerjaan tugas akhir ini, dapat disimpulkan beberapa hal sebagai berikut:

1. Salah satu tahapan yang sangat penting pada pembuatan sistem ini adalah pra-proses data di awal. Data yang akan di-*training* perlu dipastikan benar-benar bersih. Ketepatan dalam melakukan pra-proses data akan sangat menentukan hasil dari pembuatan sistem.
2. Data kamus yang dihasilkan berjumlah 77.990 kata, yang terdiri dari 72.254 kata kamus utama, ditambah kamus mapping berjumlah 5.736 kata. Banyaknya data kamus akan sangat membantu dalam proses pengecekan kata.
3. Dari berbagai macam percobaan pembuatan model, ditemukan konfigurasi parameter *training* terbaik yang digunakan, yaitu modelV3_ft berbasis *FastText* dengan parameter *size*=300, *learning algorithm* CBOW, iterasi=2, *minimum word frequency*=5, *context window*=5, *min_n*=3, dan *max_n*=6. Akurasi yang didapatkan apabila diuji dengan 100 kata non-kamus adalah 45,6%.
4. Parameter penting yang perlu diperhatikan adalah *context window* dan *learning algorithm*. Dimana *learning algorithm* CBOW cocok untuk tujuan model ini yaitu memprediksi kata menurut konteks. dan *tweaking* parameter *context window* yang tepat tentu akan dapat meningkatkan akurasi model.

5. Dalam penelitian ini dapat disimpulkan bahwa model *FastText* sedikit lebih unggul dibandingkan model *Word2Vec*, dengan selisih akurasi pengujian 100 kata hanya sekitar 2~5% dengan parameter yang serupa.
6. Hasil normalisasi dengan pengecekan kamus, pembobotan model, *levenshtein's distance,* dan *jaro-winkler distance* dinilai dapat melakukan pengoreksian dengan cukup baik, karena hasil akhir pengujian menunjukkan akurasi yang cukup baik yaitu 80,76%

7.2 Saran

Berdasarkan hasil pengerjaan tugas akhir yang telah dilakukan, beberapa saran yang dapat dipertimbangkan untuk penelitian selanjutnya antara lain:

1. Melakukan pembersihan karakter dengan tepat pada tahap pra-proses data. Karakter tertentu akan lebih baik jika dihilangkan, bukan diganti dengan spasi. Begitu pula sebaliknya.
2. Memperkaya data kamus mapping, karena semakin besar kosa kata yang sudah terpetakan dengan benar, maka hasil akhir sistem tentu akan semakin baik.
3. Penambahan koreksi untuk kata majemuk. Hasil koreksi dengan model akan menghasilkan 1 kata koreksi untuk 1 kata masukan. Sehingga koreksi untuk kata-kata seperti "trims" perlu ditambahkan.
4. Penambahan *tag* untuk mendeteksi nama objek, merek, orang, atau nama tempat agar tidak diikutkan dalam proses pengoreksian.

DAFTAR PUSTAKA

[1] Asosiasi Penyelenggara Jasa Internet Indonesia - APJII, "Penetrasi & Perilaku Pengguna Internet Indonesia - Survey 2016," p. 34, 2016.

[2] Djuwita utami, "Karakteristik penggunaan bahasa pada status facebook," *Skripsi*, 2010.

[3] J. Pustejovsky and A. Stubbs, *Natural Language Annotation for Machine Learning*. .

[4] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," pp. 1–12, 2013.

[5] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Distributed Representations of Words and Phrases and Their Compositionality," pp. 1–9.

[6] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of Tricks for Efficient Text Classification," 2016.

[7] S. Priansya, "NORMALISASI TEKS MEDIA SOSIAL MENGGUNAKAN WORD2VEC, LEVENSHTEIN DISTANCE, DAN JARO-WINKLER DISTANCE," Institut Teknologi Sepuluh Nopember Surabaya, 2017.

[8] J. Turian, L. Ratinov, Y. Bengio, and J. Turian, "Word Representations: A Simple and General Method for Semi-supervised Learning," *Proc. 48th Annu. Meet. Assoc. Comput. Linguist.*, no. July, pp. 384–394, 2010.

[9] D. Jurafsky, "Minimum Edit Distance," *Standford Univ.*, 2014.

[10] A. Kurniawati, "Implementasi Algoritma Jaro-Winkler Distance untuk Membandingkan Kesamaan Dokumen Berbahasa Indonesia," *Proceeding, Semin. Ilm. Nas. Komput. dan Sist. Intelijen KOMMIT 2008, Depok, Indones.*, 2010.

[11] Herman and D. A. Mononimbar, “Indonesia Fifth-Largest Country in Terms of Twitter Users,” 2017.

[12] I. M. P. Doyle, G. Dale, H. Choi, and B. City, “United States Patent: Method of Web Crawling Utilizing Crawl Numbers,” 2012.

BIODATA PENULIS

Penulis lahir di Sangatta pada tanggal 29 Oktober 1997. Merupakan anak kedua dari 3 bersaudara. Penulis telah menempuh beberapa pendidikan formal yaitu; SD Buin Batu Sumbawa, SMP Ar-Rohmah Malang, SMP Negeri 2 Mataram, dan SMA Negeri 1 Mataram.

Pada tahun 2014 pasca kelulusan SMA, penulis melanjutkan pendidikan dengan jalur SBMPTN (tulis) di Departemen Sistem Informasi FTIK, Institut Teknologi Sepuluh Nopember (ITS) Surabaya, dan terdaftar sebagai mahasiswa dengan nomor pokok (NRP) 05211440000118. Selama masa perkuliahan, penulis mengikuti beberapa kegiatan kemahasiswaan, seperti Latihan Ketrampilan Manajemen Mahasiswa hingga tingkat pra-dasar. Penulis juga mengikuti beberapa kepanitiaan, dan pernah menjabat sebagai staf ahli Kajian Islam Sistem Informasi di departemen kaderisasi.

Pada tahun keempat, penulis mengambil fokus pada Laboratorium Akuisisi Data dan Diseminasi Informasi (ADDI). Penulis dapat dihubungi melalui *email* di *arif.samudro2@gmail.com*