TUGAS AKHIR – KS141501

RANCANG BANGUN APLIKASI BERBASIS MICROSERVICE UNTUK KLASIFIKASI SENTIMEN.

STUDI KASUS: PT. YESBOSS GROUP INDONESIA (KATA.AI)

DEVELOPMENT OF MICROSERVICE BASED APPLICATION FOR SENTIMENT CLASSIFICATION.

CASE STUDY: PT. YESBOSS GROUP INDONESIA (KATA.AI)

RAMA RAHMANDA

NRP 5214 100 119

Dosen Pembimbing:

Dr. Ir. Aris Tjahyanto, M.Kom

DEPARTEMEN SISTEM INFORMASI
Fakultas Teknologi Informasi dan Komunikasi

Institut Teknologi Sepuluh Nopember

Surabaya 2018

TUGAS AKHIR – KS141501

RANCANG BANGUN APLIKASI BERBASIS MICROSERVICE UNTUK KLASIFIKASI SENTIMEN.

STUDI KASUS: PT. YESBOSS GROUP INDONESIA (KATA.AI)

RAMA RAHMANDA

NRP 5214 100 119

Dosen Pembimbing:

Dr. Ir. Aris Tjahyanto, M.Kom

DEPARTEMEN SISTEM INFORMASI
Fakultas Teknologi Informasi dan Komunikasi

Institut Teknologi Sepuluh Nopember

Surabaya 2018

FINAL PROJECT – KS 141501

DEVELOPMENT OF MICROSERVICE BASED APPLICATION FOR SENTIMENT CLASSIFICATION.

CASE STUDY: PT. YESBOSS GROUP INDONESIA (KATA.AI)

RAMA RAHMANDA
NRP 5214 100 119

Supervisor:
Dr. Ir. Aris Tjahyanto, M.Kom

DEPARTMENT OF INFORMATION SYSTEMS
Faculty of Information Technology and Communication
Institut Teknologi Sepuluh Nopember
Surabaya 2018

LEMBAR PENGESAHAN

RANCANG BANGUN APLIKASI BERBASIS MICROSERVICE UNTUK KLASIFIKASI SENTIMEN. STUDI KASUS: PT. YESBOSS GROUP INDONESIA (KATA.AI)

TUGAS AKHIR

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

Oleh:

RAMA RAHMANDA
NRP 5214 100 119

Surabaya, Januari 2018

KEPALA
DEPARTEMEN SISTEM INFORMASI

Dr. Ir. Aris Tjahyanto, M.Kom
NIP. 196503101991021001

Halaman ini sengaja dikosongkan

LEMBAR PERSETUJUAN

RANCANG BANGUN APLIKASI BERBASIS MICROSERVICE UNTUK KLASIFIKASI SENTIMEN. STUDI KASUS: PT. YESBOSS GROUP INDONESIA (KATA.AI)

TUGAS AKHIR

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

Oleh:

RAMA RAHMANDA
NRP 5214 100 119

Disetujui Tim Penguji: Tanggal Ujian: 05 Januari 2018
Periode Wisuda: Maret 2018

ix

Dr. Ir. Aris Tjahyanto, M.Kom (Pembimbing 1)

Nisfu Asrul Sani, S.Kom., M.Sc (Penguji 1)

Irmasari Hafidz, S.Kom, M.Sc (Penguji 2)

Halaman ini sengaja dikosongkan

RANCANG BANGUN APLIKASI BERBASIS MICROSERVICE UNTUK KLASIFIKASI SENTIMEN.

STUDI KASUS: PT. YESBOSS GROUP INDONESIA (KATA.AI)

Nama Mahasiswa	**:**	**Rama Rahmanda**
NRP	**:**	**5214 100 119**
Departemen	**:**	**Sistem Informasi**
Pembimbing 1	**:**	**Dr. Ir. Aris Tjahyanto, M.Kom**

ABSTRAK

Perkembangan AI di Indonesia menyebabkan munculnya peluang bisnis baru bagi perusahaan di Indonesia. Salah satu perusahaan di bidang AI yaitu Kata.ai telah membuat chatbot yang memiliki 1.7 pengguna dan 200 juta pesan selama setahun. Dengan tingkat penggunaan yang tinggi bot dituntut untuk merespon pengguna dengan baik dan cepat, analisa sentimen perlu dilakukan sehingga bot dapat merespon sesuai dengan sentimen pengguna sedangkan pengembangan dengan microservice memudahkan layanan untuk disebar dibeberapa instance sehingga dapat meningkatkan respon dari layanan.

Pada beberapa penelitian sebelumnya telah dibuat banyak model klasifikasi sentimen yang membagi kelasnya menjadi 3 yaitu positif, negatif dan netral. Namun, kelas tersebut masih luas sehingga dibutuhkan pengelompokan ke kelas yang lebih spesifik agar bot dapat merespon dengan lebih baik. Studi ini menghasilkan sebuah aplikasi web service untuk mengklasifikasikan sentimen pengguna ke dalam 9 kelas sentimen.

Perancangan model klasifikasi terbaik dilakukan dengan membuat beberapa skenario pelatihan dengan parameter dan algoritma yang berbeda. Skenario tersebut dipisah menjadi 2 tipe model yaitu datar dan hirarki. Hasil dari perancangan model klasifikasi sentimen dengan model hirarki memberikan performa yang paling baik dengan presisi 62% dan akurasi 76.26% namun dalam implementasinya waktu respon yang dibutuhkan untuk mengakses model hirarki lebih besar.

Kata Kunci: Sentimen Analisis, Microservice, Klasifikasi

DEVELOPMENT OF MICROSERVICE BASED APPLICATION FOR SENTIMENT CLASSIFICATION. CASE STUDY: PT. YESBOSS GROUP INDONESIA (KATA.AI)

Nama Mahasiswa	**:**	**Rama Rahmanda**
NRP	**:**	**5214 100 119**
Departemen	**:**	**Sistem Informasi**
Pembimbing 1	**:**	**Dr. Ir. Aris Tjahyanto, M.Kom**

ABSTRACT

The development of AI in Indonesia led to the emergence of new business opportunities for companies in Indonesia. One company in the field of AI that is Kata.ai has made a chatbot that has 1.7 users and 200 million messages over the year. With high usage levels bots are required to respond to users well and quickly, sentiment analysis needs to be done so bots can respond according to user sentiments while development with microservice makes it easy for services to be deployed in multiple instances in order to improve the response of the service.

In some previous studies have made many models of classification sentiments that divide the class into 3 that is positive, negative and neutral. However, the class is still large so it needs grouping into a more specific class so the bots can respond better. This study produced a web service application to classify user sentiments into 9 sentiment classes.

The design of the best classification model is done by creating several training scenarios with different parameters and

algorithms. The scenario is split into 2 types of models, namely flat and hierarchy. The result of design of sentiment classification model with hierarchical model gives the best performance with 62% accuracy and 76.26% accuracy but in the implementation of response time needed to access hierarchy model is longer.

Keywords: Sentiment Analysis, Microservice, Classification

KATA PENGANTAR

Puji dan syukur penulis tuturkan ke hadirat Allah SWT, Tuhan Semesta Alam yang telah memberikan kekuatan dan hidayah-Nya kepada penulis sehingga penulis mendapatkan kelancaran dalam menyelesaikan tugas akhir ini yang merupakan salah satu syarat kelulusan pada Departemen Sistem Informasi, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember Surabaya.

Terima kasih penulis sampaikan kepada pihak-pihak yang telah mendukung, memberikan saran, motivasi, semangat, dan bantuan baik berupa materiil maupun moril demi tercapainya tujuan pembuatan tugas akhir ini. Tugas akhir ini tidak akan pernah terwujud tanpa bantuan dan dukungan dari berbagai pihak yang sudah melauangkan waktu, tenaga dan pikirannya. Secara khusus penulis akan menyampaikan ucapan terima kasih yang sebanyak-banyaknya kepada:

1. Segenap keluarga besar terutama kedua orang tua dan kakak penulis, Bapak Budi Harman, Ibu Elly Mawarsari, dan Sekar Tanjung yang senantiasa mendoakan, memberikan motivasi dan semangat, sehingga penulis mampu menyelesaikan pendidikan S1 ini dengan baik.
2. Bapak Dr. Ir. Aris Tjahyanto, M.Kom., selaku Kepala Departemen Sistem Informasi ITS, Bapak Nisfu Asrul Sani, S.Kom, M.Sc selaku KaProdi S1 Sistem Informasi ITS serta seluruh dosen pengajar beserta staf dan karyawan di Departemen Sistem Informasi, FTIF ITS Surabaya selama penulis menjalani kuliah.
3. Bapak Dr. Ir. Aris Tjahyanto, M.Kom. selaku dosen pembimbing yang telah banyak meluangkan waktu untuk membimbing, mengarahkan, dan mendukung dengan memberikan ilmu, petunjuk, dan motivasi dalam penyelesaian Tugas Akhir.

4. Ibu Mahendrawati ER, ST, M.Sc, Ph.D sebagai dosen wali penulis selama menempuh pendidikan di Departemen Sistem Informasi.
5. Bapak, Nisfu Asrul Sani, S.Kom, M.Sc serta Ibu Irmasari Hafidz, S.Kom, M.Sc selaku dosen penguji yang telah memberikan kritik, saran, dan masukan yang dapat menyempurnakan Tugas Akhir ini.
6. Fariz Ikhwantri selaku teman sekaligus pengembang Machine Learning Kata.ai yang telah memberikan bantuan dalam bentuk teori dan praktik dalam pengerjaan tugas akhir ini.
7. Teman-teman Sistem Informasi angkatan 2014 (OSIRIS) yang senantiasa menemani dan memberikan motivasi bagi penulis selama perkuliahan hingga dapat menyelesaikan tugas akhir.
8. Rekan-rekan serumah gading kuning yang telah memberikan banyak kenangan manis dan pahit semasa kuliah
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.

Penyusunan laporan ini masih jauh dari kata sempurna sehingga penulis menerima adanya kritik maupun saran yang membangun untuk perbaikan di masa yang akan datang. Semoga buku tugas akhir ini dapat memberikan manfaat bagi pembaca.

Surabaya, 05 Januari 2018o

Penulis,

Rama Rahmanda

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR KODE

DAFTAR TABEL

1. BAB I
PENDAHULUAN

Pada bab pendahuluan akan diuraikan proses identifikasi masalah penelitian yang meliputi latar belakang masalah, perumusan masalah, batasan masalah, tujuan tugas akhir, manfaat kegiatan tugas akhir dan relevansi terhadap pengerjaan tugas akhir. Berdasarkan uraian pada bab ini, harapannya gambaran umum permasalahan dan pemecahan masalah pada tugas akhir dapat dipahami.

1.1 Latar Belakang Masalah

Perkembangan AI di Indonesia menyebabkan munculnya peluang bisnis baru bagi perusahaan Indonesia. Kata.ai (PT. YesBoss Group Indonesia) adalah salah satu perusahaan *Business-to-Business* yang mengembangkan sebuah teknologi bernama *Bot Studio Platform*, sebuah aplikasi untuk mengembangkan chatbot yang dilengkapi dengan teknologi *Natural Language Processing (NLP*) Bahasa Indonesia[1]. Menurut Engkun W. Juganda, "Indonesia sekarang merasa bisnis mereka sangat akan terpengaruhi dengan kehadiran teknologi AI, bahkan kedepannya AI ini akan menjadi juru bicara meski bukan secara harfiah tapi ini adalah gabungan bagaimana customer akan berkomunikasi dengan perusahaan tersebut. Di Indonesia AI akan mulai populer."[1]. Penting bagi perusahaan untuk mengetahui sentimen pelanggannya terhadap kualitas layanan perusahaan, sehingga perusahaan dapat mengukur tingkat kepuasan dari pelanggan[1]. Melihat hal tersebut Kata.ai ingin teknologi NLP-nya dapat digunakan untuk mengklasifikasikan, memahami dan mengevaluasi sentimen dan opini yang diungkapkan oleh pengguna chatbotnya.

Jemma merupakan salah satu chatbot hasil kerjasama Kata.ai dengan Unilever telah berhasil berteman dengan lebih dari 1.7 juta pengguna Line[2]. Dalam waktu satu tahun Kata.ai telah

total chatbot Kata.ai melampaui 26 juta orang dan lebih dari 200 juta pesan[2]. Untuk memproses pesan secara cepat Kata.ai memecah beberapa layanannya menjadi layanan-layanan kecil dengan *multi-instance*. Pemecahan sebuah aplikasi menjadi layanan kecil tersebut dinamakan *microservice architecture*[3]. *microservice* yang memberikan fungsionalitas dengan menyusun sejumlah layanan kecil dengan fungsionalitas yang spesifik untuk menyelesaikan tugas yang lebih besar[3]. Oleh karena itu, pengembangan dengan *microservice* memudahkan pengembang untuk meningkatkan skalabilitas sistem.

Kata.ai memiliki sebuah layanan untuk membantu percakapan dengan bot yaitu *verstand service. Verstand service* sendiri merupakan layanan yang dapat melakukan klasifikasi dan *tagging* terhadap sebuah teks dengan melakukan pelatihan pada data menggunakan beberapa algoritma pembelajaran mesin yang tersedia. Hasil dari klasifikasi atau *tagging* tersebut digunakan untuk memprediksi atau mengelompokkan sebuah teks. Namum pemanfaatan *Verstand* sendiri masih terbatas karena hanya dapat diakses melalui merapi proxy buatan Kata.ai. *Web service* mengakses sebuah aplikasi lewat protocol http/https sehingga dapat diakses oleh banyak pengguna[4]. Oleh karena itu, *verstand service* dapat digunakan untuk mengembangkan fitur klasifikasi dan sebuah web service dapat digunakan untuk mengakses hasil klasifikasi tersebut.

Analysis Sentiment merupakan sebuah metode untuk menganalisa sentimen seseorang dan salah satu dari implementasinya adalah melakukan pengelompokan sebuah teks ke dalam kelas sentimen[5]. Opini merupakan pusat dari hampir semua aktivitas manusia yang melambangkan kebiasaan manusia tersebut[5]. Pada penelitian sebelumnya telah dilakukan penelitian *Analysis Sentiment* terhadap pada akun resmi pemerintah kota Surabaya dengan mengelompokan sebuah kalimat ke dalam 3 kelas sentimen yaitu positif, negatif

dan netral menggunakan algoritma pembelajaran mesin seperti *Naive Bayes* dan *Support Vector Machine* (SVM)[6]. Namun, sentimen ataupun opini yang dimiliki oleh manusia lebih dari sekedar positif dan negatif[6]. Oleh karena itu, diperlukan pengembangan klasifikasi sentimen yang dapat mengetahui opini pengguna lebih jauh, sehingga sistem akan bisa memberikan jawaban yang tepat untuk kebutuhan pengguna.

Dengan adanya tugas akhir ini diharapkan dapat mengembangkan sebuah *web service* berbasis *microservice architecture* untuk mengklasifikasikan sentimen pengguna. Web service yang akan dikembangkan merupakan layanan independen yang menyimpan data hasil klasifikasi dari *verstand service*.

1.2 Perumusan Masalah

Rumusan Masalah dari penelitian ini adalah:

1. Bagaimana mengembangkan model klasifikasi sentimen menggunakan *verstand service*?
2. Bagaimana mengembangkan aplikasi *web service* yang terhubung dengan *verstand service*?

1.3 Batasan Masalah

1. Penelitian ini menggunakan data dan layanan yang telah disediakan oleh perusahaan Kata.ai (PT. YesBoss Group Indonesia).
2. Pembelajaran mesin akan dilakukan menggunakan *verstand Service* Kata.ai untuk mengembangkan model klasifikasi.
3. Klasifikasi teks berdasarkan 9 kelas sentimen yaitu *upset, mocking, sarcasm, denial, spam, closing, neutral, pleasant,* dan *compliment*
4. Klasifikasi sentimen menggunakan bahasa indonesia dan dua buah algoritma pembelajaran mesin yaitu *Fasttext* dan *Logistic Regression*.

1.4 Tujuan Penelitian

Tujuan dari penelitian ini adalah:

1. Membangun model klasifikasi sentimen
2. Mengimplementasikan aplikasi web service yang terintegrasi dengan layanan Verstand

1.5 Manfaat Penelitian

Bagi masyarakat, sebagai bentuk awal yang memungkinkan untuk pemanfaatan layanan model klasifikasi lainnya dari berbagai provider agar dapat digunakan dalam satu platform yang terintegrasi.

1.6 Relevansi

Pada tugas akhir ini akan diimplementasikan sebuah aplikasi klasifikasi sentimen berbasis *microservice architecture*. Sehingga relevansi tugas akhir ini terhadap laboratorium Infrastruktur Sistem dan Keamanan Teknologi Informasi (IKTI) ialah berkaitan dengan penerapan mata kuliah bidang keilmuan laboratorium IKTI yaitu Teknologi Open-Source dan Terbaru dengan implementasi *microservice* dan klasifikasi sentimen pada mata kuliah Sistem Cerdas.

BAB II
TINJAUAN PUSTAKA

Pada bab ini akan membahas mengenai penelitian sebelumnya yang berhubungan dengan tugas akhir dan teori - teori yang berkaitan dengan permasalahan tugas akhir

2.1 Studi Sebelumnya

Tabel 2.2.1 menampilkan daftar penelitian sebelumnya yang mendasari tugas akhir ini

Tabel 2.2.1 Studi Sebelumnya

1. Eksperimen Sistem Klasifikasi Analisa Sentimen Twitter Pada Akun Resmi Pemerintah Kota Surabaya Berbasis Pembelajaran Mesin
Penulis/Tahun/Sumber: F, Nuke, K, Renny, dan H, Irmasari; 2016[6]
Metode: - *Naive Bayes,* - *Support Vector Machine (SVM)* - *Stemming* - *Stop words removal*
Kesimpulan: Penelitian ini berhasil menemukan perbandingan akurasi untuk algoritma *Naive Bayes* dan SVM. Tingkat akurasi menggunakan SVM lebih besar sekisar 1-2% daripada algoritma *Naive Bayes*. Hasil akurasi dengan algoritma *Naive Bayes* dengan menggunakan *stopword removal* meningkat namun tidak terlalu signifikan dibandingkan dengan tidak menggunakan metode tersebut.
2. Analisis Sentimen Calon Gubernur DKI Jakarta 2017 Di Twitter

Penulis/Tahun/Sumber: Ghulam Asrofi Buntoro; 2017[7]
Metode: - *Naive Bayes,* - *Support Vector Machine (SVM)* - Tokenisasi - *Part of Speech Tagger*
Kesimpulan: Pada penelitian ini penlusin dapat membuktikan bahwa Analisa Sentimen dapat dilakukan untuk mengetahui sentimen masyarakat Indonesia khususnya pada media Twitter terhadap calon Gubernur DKI Jakarta 2017 Nilai akurasi tertinggi didapatkan dengan algoritma *Naive Bayes*
3. Perbandingan Statistik Bahasa N-Gram Bahasa Indonesia Dalam Cryptanalysis Simple Substitution Cipher Berbasis Algoritma Genetik
Penulis/Tahun/Sumber: Markus dan Nico Saputro; 2006[8]
Metode: - *Monogram* - *Bigram* - *Trigram*

Kesimpulan:
Pada penelitian ini didapatkan bahwa penggunaan n-gram dengan nilai 3 atau biasa disebut trigram memiliki hasil statistik yang lebih baik dibandingkan dengan monogram dan bigram. Namun trigram memiliki waktu perhitungan fitness yang lebih lama dibandingkan monogram dan bigram.

4. Prediction of Sentiment from Textual Data Using Logistic Regression Based on Stop Word Filteration and Volume of Data

Penulis/Tahun/Sumber:
Rajni Mohana dan Sukhnandan Kaur; 2016[9]

Metode:
- *Logistic Regression*
- *Support Vector Machine (SVM)*
- *Naive Bayes*
- *Stemming*
- *Stop Word Removal*

Kesimpulan:
Pada penelitian ini peneliti membandingkan 3 algoritma pembelajran mesin dengan metode *stop word* dan tanpa menggunakan *stop word*. Dari hasil penelitian tersebut didapati algoritma *Logical Regression* memiliki nilai akurasi tertinggi dan penggunaan *stop word* meningkatkan akurasi secara signifikan.

5. Lsislif: CRF and Logistic Regression for Opinion Target Extraction and Sentiment Polarity Analysis

Penulis/Tahun/Sumber:
Frederic Bechet, Patrice Bellot, dan Hussam Hamdan; 2015 [10]

Metode:
- *N-gram*
- *Logistic Regression*
- *Conditional Random Field (CRF)*
- *Sentiment Lexicon-based*
- *Negation*
- *Z-score*

Kesimpulan:
Pada penelitian ini, peneliti membahas mengenai *Opinion Target Extraction* (OTE) yang dilatih dengan algoritma CRF. Klasifikasi sentimen dilakukan menggunakan *Logistic Regression* dengan pembobotan ditiap domain.

6. Bag of Tricks for Efficient Text Classification

Penulis/Tahun/Sumber:
Armand Joulin Edouard Grave Piotr Bojanowski Tomas Mikolov ; 2016 [11]

Metode:
- *N-gram*
- *Fasttext*
- *SVM*
- *CNN*

Kesimpulan:
Penelitian ini melakukan komparasi algoritma *fasttext* dengan beberapa algoritma lainnya seperti SVM dan CNN dengan permasalahan analisa sentimen. Hasilnya algoritma *fasttext* memiliki akurasi yang sebanding dengan algoritma lainnya, namun dalam proses komputasi *fasttext* lebih cepat dalam mengklasifikasikan sentimen.

7. Sentiment Analysis: A Study On Product Features
Penulis/Tahun/Sumber: Yanyan Meng; 2012[12]
Metode: - *Sentiment Lexicon* - *Part of Speech* - *Tokenization* - *N-grams*
Kesimpulan: Pada penelitian ini, peneliti melakukan analisa untuk fitur mengurangi dimensi dan pemilihan fitur pada dokumen atau kalimat. Fitur yang dilakukan pada pembelaran *supervised* dan *unsupervised*. Hasil penggunaan fitur seperti *Bag-of-Words* memiliki ketergantungan terhadap besarnya corpus. Untuk metode pemilihan fitur *Information Gain* memliki performa yang lebih menangani analisa dokumen, sedangkan *Mutual Information* memiliki performa yang lebih baik pada analisa kalimat.

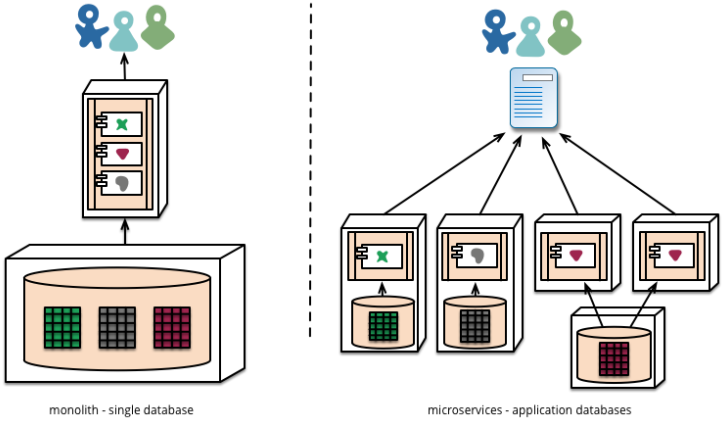
2.2 Dasar Teori

Pada bab ini akan dijelaskan mengenai dasar teori untuk mengerjakan tugas akhir ini. Dasar teori digunakan sebagai justifikasi penggunaan sebuah metode atau cara pengerjaan tugas akhir.

2.2.1 *Microservice*

Microservice adalah arsitektur perangkat lunak dan paradigma pengiriman yang menyusun sebuah fungsionalitas menjadi sejumlah service yang berdiri secara independen. *Service* yang dibuat dengan menggunakan arsitektur ini akan memiliki fungsi yang diskrit dan spesifik sehingga dapat diintegrasikan dengan *service* lainnya untuk menyediakan fungsionalitas bisnis secara

keseluruhan[3]. Dengan membagi fungsionalitas menjadi sejumlah service, *microservice* memberikan akses secara cepat tanpa terlalu membebani server[3].



Gambar 2.1 Perbedaan monolith dan microservice.

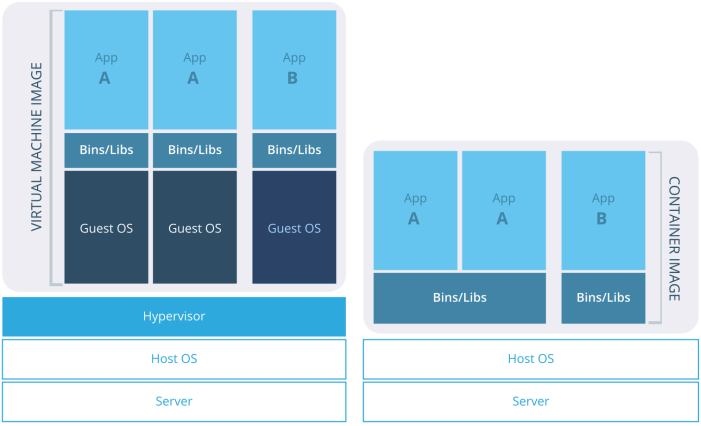
Sumber:
https://martinfowler.com/articles/microservices/images/micro-deployment.png

Dengan demikian, apabila Kata.ai ingin menambahkan atau membuat sebuah fitur baru maka Kata.ai dapat berfokus untuk merubah sebuah *service* tanpa menganggu *service* lainnya. Kata.ai memiliki banyak *service* yang mendukung *Bot Studio Platform* salah satunya adalah *verstand service*. Oleh karena itu, peneliti dapat mengembangkan klasifikasi sentimen pada *verstand service* tanpa menganggu *service* lainnya.

2.2.2 *Docker*

Docker mengimplementasikan konsep kontainer dengan menyediakan cara mudah bagi pengembang untuk mengemas aplikasi dan dependensinya dalam *image* kontainer yang dapat dijalankan di Linux modern manapun, dan segera Windows, server[13]. Kontainer bekerja mirip dengan *Virtual Machine (VM)* pada umumnya, yang menjadikan kontainer berbeda dengan VM adalah tiap kontainer membagi sumber daya OS yang sama. Kontainer mengisolasi, lingkungan portabel di

mana dapat menjalankan aplikasi beserta semua *library* dan dependensi yang dibutuhkan[13].



Gambar 2.2 Kontainer vs VM.

Sumber: [13]

2.2.3 Merapi

Merapi merupakan sebuah *library* buatan Kata.ai yang memiliki fungsi utama yaitu sebagai *dependency injector*[14]. Merapi sendiri dapat berjalan diatas NodeJS dan memiliki beberapa *plugin* yang memudahkan pengembang dalam melakukan injeksi terhadap komponen sistem[14]. Selain itu merapi memiliki sebuah *proxy service* sendiri yang dapat diaktifkan menggunakan *merapi-proxy*.

2.2.4 *Pre-processing*

Dalam pengambilan data, isi dari data tersebut tergantung dari tempat data tersebut diambil sehingga data tidak terstruktur. Hal tersebut mengarahkan pada data *noise*[5]. Oleh karena perlu dilakukan banyak *pre-processing* atau pengolahan pada data sebelum melakukan analisa.

2.2.4.1 *N-grams tokenization*

Teks perlu diolah atau dipisahkan sebagai token atau *strings* sebelum menerapkan model pembelajaran mesin. Proses ini

disebut tokenizing[12]. N-grams sendiri merupakan tipe probabilistik model untuk mengkombinasikan kata yang berdekatan pada sebuah kalimat. Fungsi probabilistiknya adalah $P(x_i \mid x_{i-(n-1)} ..., x_{i-1})$[12]. Ukuran n-gram biasanya terdiri dari 1 sampai 3 dengan nama Unigram, Bigram dan Trigram. Menurut penelitian[12], ukuran n-gram yang memiliki performa paling baik untuk bahasa Indonesia adalah 3. Biasanya hasil tokenisasi tersebut akan diberikan pembobotan. Menurut Hans Peter Luhn (1957), Bobot istilah yang terjadi dalam dokumen hanya sebanding dengan frekuensi rata-rata. Oleh karena itu untuk pembobotan akan menggunakan algoritma Term Frequency Inversed Document Frequency (TF-IDF). TF-IDF adalah sebuah Statistik numerik yang dimaksudkan untuk mencerminkan betapa pentingnya sebuah kata adalah sebuah dokumen dalam kumpulan atau korpus (*Rajaraman, A.; Ullman, J. D. 2011*).

2.2.5 *Feature Engineering*

Ketika membicarakan tentang *Natural Language*, fitur yang biasa digunakan dalam pengembangan model ialah kata, *part of speech tags*, atau informasi linguistik lainnya[15]. Secara garis besar terdapat dua buah cara merepresntasikan fitur yaitu *Dense Vectors* dan *One-Hot Representations*[15]. *Dense vectors* merepresentasikan fitur dalam sebuah vektor sedangkan *One-hot* merepresentasikan fiturnya pada tiap dimensi[15]. Representasi dengan *dense vector* memiliki tingkat komputasi yang lebih tinggi dibandingkan *one-hot*[15]. Oleh karena *dense vector* merepresentasikan fitur dalam sebuah vektor, fitur yang merupakan bilangan biner akan mudah direpresentasikan dengan cepat menggunakan *dense vector*.

2.2.5.1 *Negations*

Negations biasanya digunakan baik dalam *supervised* ataupun *unsupervised learning*. Pada *unsupervised learning*, *negations* digunakan untuk menentukan sifat berlawanan pada opini[12]. *Negations* dapat berupa frasa maupun kata. tetapi, Pott (2011) mengaplikasikan metode *negations tagging* yang dianjurkan oleh Pang et al (2002) dan menghasilkan peningkatan akurasi

0.886 menjadi 0.895. Pada tahun 2008, Ikeda et al mengajukan model dengan pergeseran polaritas untuk menangkap apakah polaritas sebuah kata telah bergeser atau tidak. Hal tersebut diperkuat oleh Nakagawa et al. (2010) yang menunjukan bahwa interaksi antar kata dalam sebuah kalimat perlu diperhitungkan dalam *Analysis Sentiment*[12]. *Negations* menggunakan daftar kata / *bag of words* tidak dapat menangkap interaksi antar kata dengan baik. Oleh karena itu Syntactic dependency tree patterns digunakan untuk menangkap interaksi tersebut. Setelah model dilatih dengan CRF dan variabel *hidden,* Nakagawa et, al. mendapatkan hasil performa yang jauh lebih baik dari sebelumnya[12].

2.2.5.2 *Part of Speech (POS)*

Part of Speech (POS) telah banyak diterapkan dalam metode NLP dan klasifikasi teks. POS *tagging* menggunakan tags spesifik untuk mengetahui arti sebuah kata dalam kalimat seperti adjective, adverb, verb, none, conjunction dan lain-lain[12]. Biasanya pada *supervised learning* POS digunakan sebagai fitur. Mejova et al. (2011) melakukan percobaan untuk melihat efektifitas dalam *feature tagging* yang berbeda dan menghasilkan performa yang jauh lebih baik dibandingkan sebuah individual diberlakukan sebagai adjectives, verbs, dan nouns[12]. POS tagger adalah sebuah proses untuk memberikan arti pada sebuah kata dalam kalimat. Berdasarkan aturan linguistik pada kata diperoleh sentimen sementara.

proses POS *tagging* terbagi ke dalam tiga proses yaitu pemisahan setiap token dalam dokumen dengan pengecekan[12] setiap kata dalam dokumen, mengidentifikasi setiap kata dalam dokumen dengan pemberian jenis kata, pengecekan kata yang belum teridentifikasi terhadap bentuk imbuhan dan akhiran sehingga diperoleh[12] kata dasar. Penentuan sentimen dilakukan dengan melihat adanya kata yang mengandung opini baik yang memiliki *polarity* positif maupun negatif dari tweet yang sudah dilabeli kelas katanya. Kelas kata yang dipilih adalah kata sifat (adjective), kata

keterangan (adverb), kata benda (noun) dan kata kerja (verb), sesuai dengan penelitian (Azhar, 2013).

2.2.6 *Scikit-learn*

Scikit-learn adalah sebuah modul Python yang mengintegrasikan berbagai jenis algoritma *Machine Learning* untuk menyelesaikan permasalahan pembelajaran seperti *Supervised Learning* dan *Unsupervised Learning*[16]. Beberapa algoritma *Machine Learning* yang dapat diimplementasikan melalui *Scikit-learn* adalah *Naive Bayes*, *Logistic Regression,* dan *Conditional Random Field*. Selain mengintegrasikan algoritma *Machine Learning, Scikit-learn* memudahkan perbandingan antar algoritma. Karena *Scikit-learn* berjalan dalam lingkungan Python, penggunaan modul ini dapat dengan mudah diintegrasikan ke dalam aplikasi di luar analisa statistika[16].

2.2.6.1 *Logistic Regression*

Logistic Regression (LR) merupakan sebuah pendekatan pembelajaran dengan fungsi $P(Y|X)$. Dimana fungsi ini berkebalikan dengan *Naive Bayes* yang memiliki asumsi *conditional independence.* Pada fungsi LR Y adalah nilai diskrit dan dipengaruhi oleh himpunan X_n yang berisikan diskrit ataupun kontinyu[17]. Setiap himpunan X memiliki bobot sendiri yang akan mempengaruhi Y. Fungsi pembobotan pada variabel menyebabkan algoritma ini disebut sebagai algoritma yang diskriminatif. Ketika Y merupakan sebuah *boolean* maka fungsi yang didapatkan yaitu:

$$P(Y=1 \mid X)=\frac{1}{(16)\ 1+exp(w0+\sum ni=1\ wiXi)}$$

dan

$$P(Y=0 \mid X)=\frac{exp(w0+\sum ni=1 wiXi)}{(17)\ 1+exp(w0+\sum ni=1\ wiXi)}$$

Persamaan LR diatas mengimplikasikan asumsi pada *Gaussian Naive Bayes Classifier* dengan fungsi yang sama yaitu $P(Y|X)$ [17]. Pada penelitian sebelumnya telah dilakukan perbandingan *Naive Bayes, SVM* dan *Logistic Regression* pada klasifikasi biner. Hasil yang didapatkan *Logistic Regression* melebihi algoritma lainnya[9]. Persamaan pada *Logistic Regression* dapat dengan mudah dibentuk ke dalam klasifikasi *multinomial* dengan menambahkan fungsi $\sum(sum)$ pada rumus *Binary(boolean) Logistic Regression*, yaitu[17]:

$$P(Y_k \mid X) = \frac{exp(w0 + \sum ni=1 wiXi)}{1 + \sum(k-1), j=1\ exp(w0 + \sum ni=1\ wiXi)}$$

Ketika $Y = Y_k$, maka menjadi

$$P(Y = Y_k \mid X) = \frac{1}{1 + \sum(k-1), j=1\ exp(w0 + \sum ni=1\ wiXi)}$$

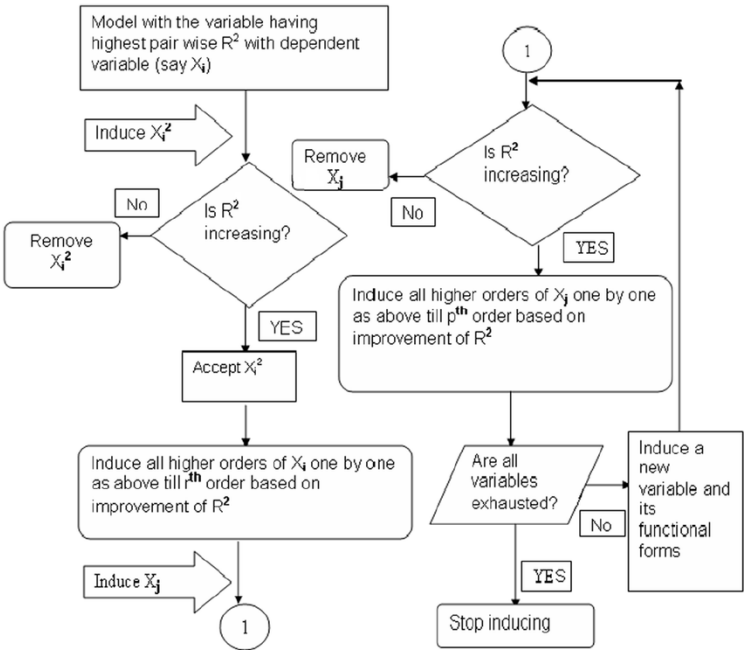
Logistic Regression memiliki sifat diskriminatif, hal tersebut memiliki kekurangan yaitu apabila sebuah variabel/fitur memiliki pembobotan yang tinggi akan menyebabkan terjadinya *overfitting.* Salah satu pendekatan untuk mengurangi terjadinya *overfitting* adalah regularisasi[17]. Pada proses regularisasi akan dilakukan pengurangan atau pemberian penalti terhadap bobot sebuah variabel. Terdapat dua pendekatan untuk melakukan regularisasi yaitu L2 dan L1[17]. Pada Regularisasi L2, penalti akan dihitung dengan mengkuadratkan normalisasi dari bobot. Sedangkan L1 akan menghitung berdasarkan nilai absolut dari bobot[18].

$$\text{L2} = \alpha \sum_{i=1}^{N} w^2$$

sedangkan

$$\text{L1} = \alpha \sum_{i=1}^{N} |w|$$

Regularisasi L1 akan mengabaikan *outliers data* atau data yang berada jauh dari data lainnya. Sedangkan L2 akan tetap menghitung *outliers data*[18].



Gambar 2.3 Flow pembelajaran dengan Logistic Regression

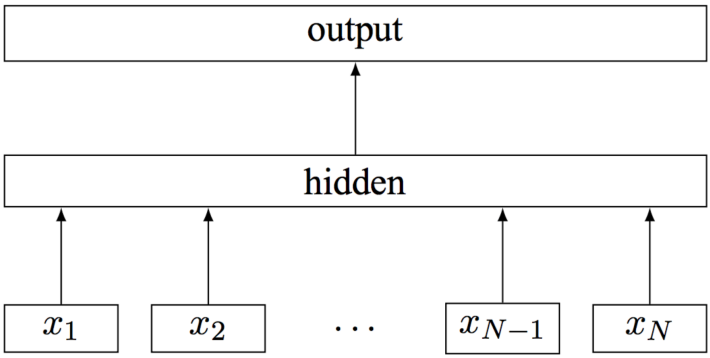
Sumber: [19]

Gambar diatas menjelaskan bagaimana pembelajaran mesin dengan algoritma *Logistic Regression* melatih data *training* sebelum akhirnya terbentuk sebuah model sehingga dapat dimanfaatkan untuk proses klasfikasi. Pada penelitian sebelumnya telah dilakukan perbandingan *Naive Bayes, SVM* dan *Logistic Regression* pada klasifikasi biner. Hasil yang didapatkan *Logistic Regression* melebihi algoritma lainnya[10]. Dalam beberapa penelitian sebelumnya juga telah membuktikan bahwa *Logistic Regression* memiliki hasil yang baik pada klasifikasi teks[10].

2.2.6.2 *Fasttext*

Fasttext merupakan sebuah library untuk pembelajaran mesin secara efisien pada klasifikasi kalimat dan representasi kata[20]. Fasttext menggabungkan beberapa konsep paling

sukses yang diperkenalkan oleh pemrosesan bahasa alami dan komunitas pembelajaran mesin dalam beberapa dekade terakhir[21]. Algoritma ini memiliki struktur hirarki serta merepresentaiskan fiturnya dengan *dense vector*, hal ini mengurangi kompleksitas dalam pelatihan[21]. Selain itu, fasttext menekankan bahwa kelas pada klasifikasi tidak seimbang.



Gambar 2.4 Arsitektur model fasttext untuk klasifikasi kalimat

sumber: [11]

Gambar 2.4 Arsitektur model fasttext untuk klasifikasi kalimat menunjukan linear model sederhana dengan batasan peringkat dan representasi kata tiap fitur akan membentuk representasi teks, fitur tersebut akan disatukan dan dirata-ratakan membentuk sebuah *hidden* variabel[11]. Oleh karena itu, fasttext dengan fitur n-grams baik digunakan untuk melatih klasifikasi teks yang memiliki kelas yang tidak merata.

2.2.7 Pengukuran performa

Melihat sebuah performa dari model machine learning dapat dihitung menggunakan beberapa ukuran yaitu akurasi, presisi, *lift* dan *recall*[22]. Pada [6] melakukan perbandingan model klasifikasi dengan menggunakan perhitungan akurasi, presisi, recall dan f-measure untuk mendapatkan model terbaik dari beberapa pelatihan menggunakan algoritma yang berbeda.

Akurasi akan menghitung jumlah data faktual dibandingkan dengan data keseluruhan[23].

$$Akurasi = \frac{tp+tn}{tp+tn+fp+fn}$$[23]

tp = True positive fp = False positive
tn = True negative fn = false negative

Akurasi saja tidak cukup untuk mengukur performa dan akurasi sering tidak tepat[22], karena akurasi akan menghitung berdasarkan jumlah keseluruhan data sehingga apabila jumlah data tidak seimbang nilai akurasi menjadi tidak valid. Oleh karena itu akan digunakan sebuah pengukuran presisi untuk melihat tingkat keberhasilan pada kelas yang sama.

$$Presisi = \frac{tp}{tp+tn}$$[23]

Dengan menghitung presisi maka dapat terlihat performa sebuah model dalam mengklasifikasikan tiap kelasnya terhadap jumlah data dari kelas itu sendiri. Ketika mencari rata-rata presisi dari kelas, presisi memberikan bobot pada kelas yang sama walaupun memiliki jumlah data yang berbeda.

2.2.8 Pengujian unit dan integrasi

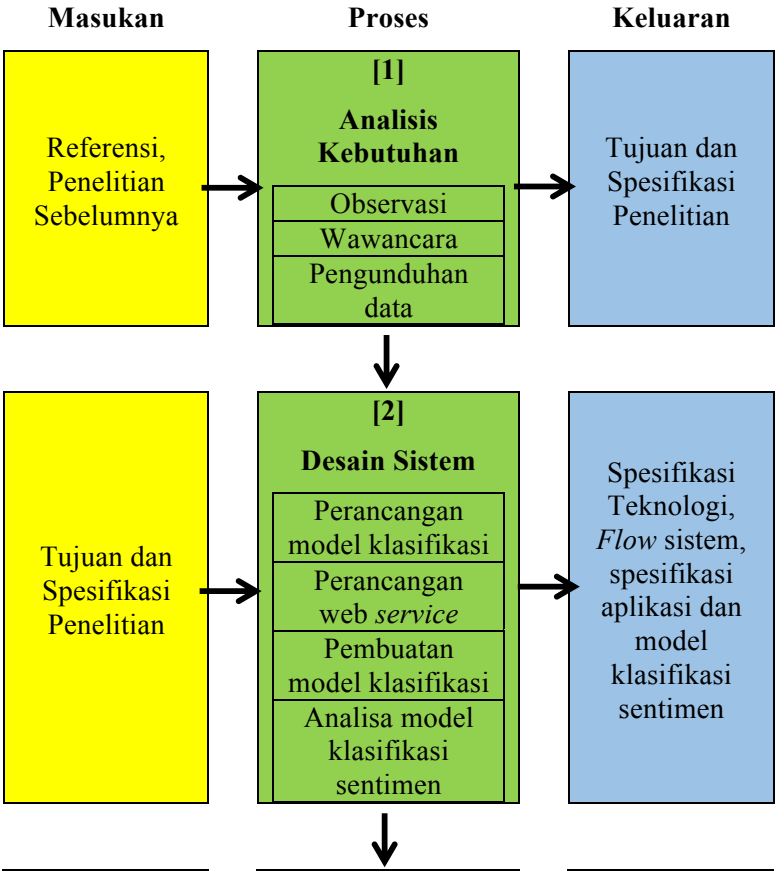
Sebuah unit merupakan bagian terkecil yang dapat diuji pada sebuah aplikasi yang berarti *unit testing* akan memastikan sebuah unit untuk memenuhi tujuannya[24]. Tanpa tingkat kepercayaan yang tinggi pada sebuah unit akan lebih sulit untuk memastikan keberhasilan sebuah *software* secara keseluruhan[24]. Keuntungan dari melakukan *unit testing* yaitu dapat mendeteksi bug saat tahap pengembangan dan pembenahan pada tahap ini lebih mudah dan murah ketimbang pembenahan pada tahap selanjutnya dalam pengembangan[24]. Rancangan pengujian dilakukan dengan menjabarkan fungsional menjadi activity diagram untuk menentukan test case yang diuji[25]. Sedangkan pengujian integrasi akan menguji sistem secara keseluruhan sesuai dengan fungsionalnya hal tersebut dilakukan untuk memastikan sistem dapat berjalan secara unit dan keseluruhan[26].

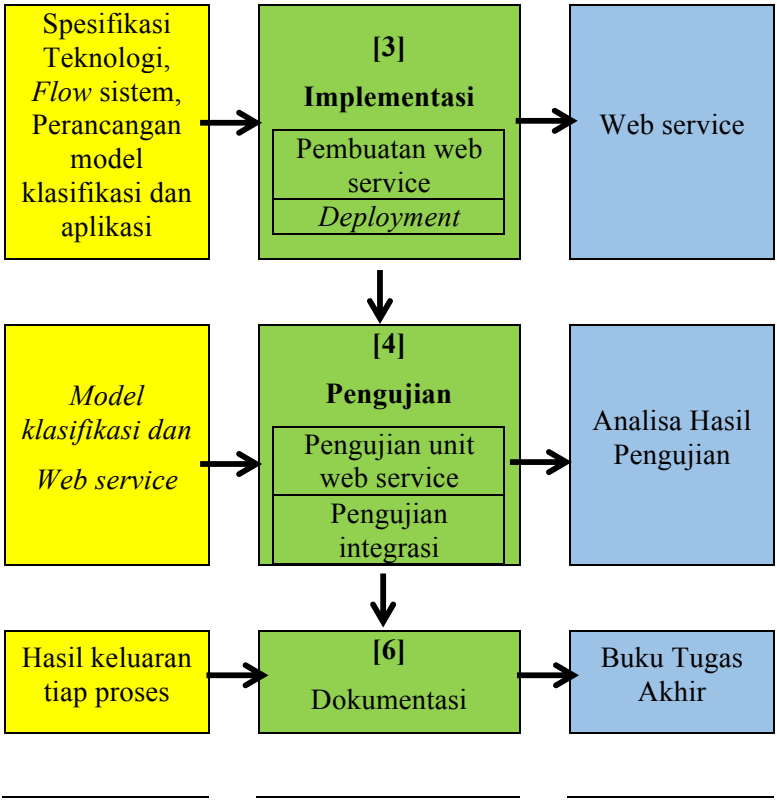
BAB III
METODOLOGI

Pada bab ini akan membahas mengenai tahapan dilakukannya penelitian menggunakan SDLC Waterfall dari studi literatur sampai dengan implementasi. Setiap tahapan terdiri atas masukan (*input*), proses, dan keluaran (*output*).

3.1 Tahapan Pelaksanaan Tugas Akhir

Table 3.1 Metodologi Penelitian





3.1.1 Analisis kebutuhan

Tahapan ini merupakan fase pertama dalam pengerjaan Tugas Akhir ini. Pada tahap studi literatur dilakukan penggalian kebutuhan pengetahuan terkait dengan studi kasus yang akan diambil. Paper yang dijadikan referensi berasal dari luar dan dalam negeri yang menggunakan metode pendukung penelitian. Paper-paper yang diambil adalah penelitian seputar klasifikasi sentimen, *microservice* beserta metode untuk melakukan *preprocessing* data. Pada tahap ini, akan dilakukan penggalian informasi mengenai arsitektur dan kebutuhan Kata.ai dalam mengembangkan klasifikasi sentimen pada *verstand service.*

3.1.1.1 Observasi

Pada tahapan ini akan dilakukan observasi terhadap beberapa media Kata.ai untuk mengetahui teknologi yang digunakan dan teknologi yang akan digunakan untuk membantu proses pembuatan aplikasi.

3.1.1.2 Wawancara

Pada tahapan ini akan dilakukan wawancara terhadap karyawan atau pihak Kata.ai untuk menggali kebutuhan pengguna. Dari hasil kebutuhan akan terbagi menjadi kebutuhan fungsional dan non-fungsional yang akan digunakan sebagai bahan perancangan pengujian.

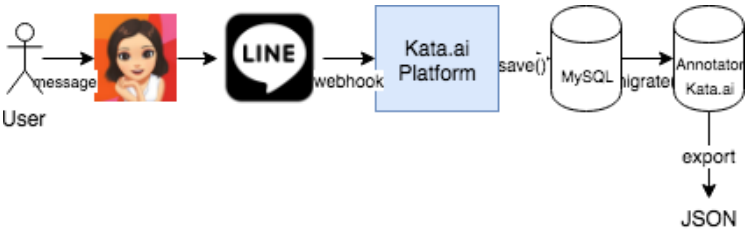
3.1.1.3 Pengunduhan data

Pada fase ini dilakukan pengunduhan data dari Kata.ai melalui sebuah website *annotator.kata.ai*. Data berupa teks frasa, kalimat dan paragraf yang telah dilabeli berdasarkan tag, topik dan *traits*. Sumber dari dataset ini ialah teks data percakapan antara bot dengan pengguna pada tahun 2016 sampai dengan 2017.

Spesifikasi dataset:

1. Sumber:
- Percakapan bot LINE
- Percakapan Virtual Asistant
2. Tahun: 2016 – 2017
3. Total data: 9806

Data tersebut diambil oleh Kata.ai pada tahun 2016 sampai dengan 2017 pada sebuah akun *official* LINE “Jemma”.



Gambar 3.1 Sumber data

Klasifikasi berdasarkan sentimen terdiri atas 9 kelas sentimen. Hal tersebut didasari oleh keinginan Kata.ai untuk memberikan kecerdasan pada bot untuk dapat mengenali perasaan/sentimen pengguna secara spesifik. Sehingga ketika mengetahui sentimen, bot dapat bertindak sesuai dengan kebutuhan pengguna. Lima dari sembilan klasifikasi memiliki konteks negatif dikarenakan, fungsi bot lebih diprioritaskan untuk mengetahui pelanggan yang bermasalah. Berikut adalah penjelasan untuk tiap klasifikasi berdasarkan Kata.ai dan KBBI:

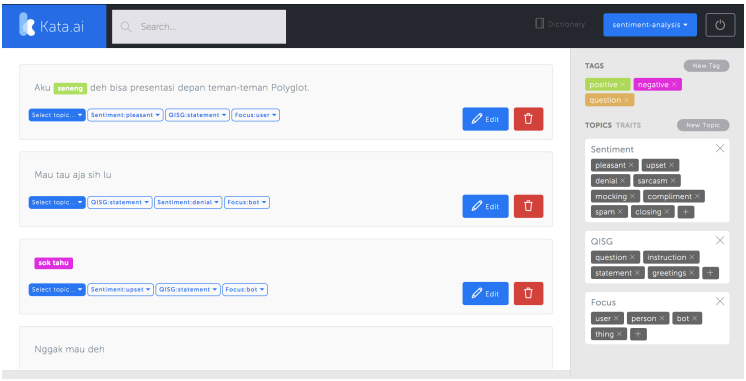
Table 3.2 Justifikasi klasifikasi

Klasifikasi	Definisi	
	Kata.ai	KBBI
Upset (Kecewa)	terdiri dari perasaan yang menganggu, kesal, tidak suka, dan kesedihan.	kecil hati, tidak puas (karena tidak terkabul keinginannya, harapannya, dan sebagainya), tidak senang
Denial (Penolakan)	ketika pengguna menolak memberikan informasi atau untuk terus berinteraksi dengan bot	proses, cara, perbuatan menolak
Sarcasm (Sarkasme)	digunakan untuk menjelaskan ejekan dalam bentuk pujian	kata-kata pedas untuk menyakiti hati orang lain, cemoohan atau ejekan kasar

Mocking (Mengejek)	digunakan untuk menentukan kemarahan dan penggunaan nama panggilan, kata-kata kutukan, dan cara menghina lainnya.	mengolok-olok (menertawakan, menyindir) untuk menghinakan, mempermainkan dengan tingkah laku, mencemooh
Compliment (Pujian)	digunakan untuk membiarkan bot tahu bahwa itu pengguna sedang memuji sebuah objek	pernyataan memuji
Spam	digunakan untuk mengkategorikan salah ejaan, kata-kata kotor, dan kata-kata asing.	surat yang dikirim tanpa diminta melalui internet, biasanya berisi iklan
Closing (Mengakhiri)	digunakan untuk mengkategorikan ketika pengguna ingin menyudahi sebuah objek pembicaraan.	proses, cara, perbuatan mengakhiri, penyudahan
Pleasant (Nyaman)	digunakan ketika pengguna merasa nyaman dan puas akan sesuatu.	Sedap, sejuk, enak

Neutral	digunakan ketika teks tidak mengandung sentimen	tidak dalam kelompok

Data yang diberikan melalui website *annotator.kata.ai* tidak perlu melalui proses *cleansing* symbol, normalisasi teks, penghapusan duplikasi sehingga peneliti hanya akan mendefinisikan fitur yang digunakan untuk melatih model.



Gambar 3.2 Data dari kata.ai

Dibawah ini merupakan sampel data dari dataset Kata.ai

Table 3.3 Sampel data

Teks	**Klasifikasi**
Aku seneng deh bisa presentasi depan teman-teman Polyglot.	pleasant
Mau tau aja sih lu	statement
sok tahu	upset
Nggak mau deh	denial
Rahasia	denial
Aduhhh gk perlu	pleasant

Nomor tefon kamu berapa	neutral
Nanya mulu	upset
Kepo	denial
BODOK AMAT	upset
Aku	neutral
Hatiku sakit	upset

3.1.2 Desain sistem

Pada tahapan ini akan dilakukan persiapan untuk rancang bangun aplikasi yang digunakan untuk mengklasifikasikan sentimen pengguna. Pelatihan model klasifikasi sentimen akan menggunakan layanan dari Kata.ai yaitu *verstand service* dan media untuk mengakses model tersebut akan dibuat sebagai web service.

3.1.2.1 Perancangan model klasifikasi

Mendefinisikan *flow* dan skenario dari beberapa gabungan pre-process, ekstrasi fitur dan perbedaan algoritma untuk melatih model klasifikasi pada *verstand service*. Hasil dari tiap skenario dipilih yang memiliki tingkat akurasi tertinggi.

3.1.2.2 Pembuatan model klasifikasi sentimen

Setelah rancangan model klasifikasi telah dibuat maka data yang akan digunakan untuk melatih model klasifikasi akan melalui tahap *pre-process* sebelum masuk ke dalam proses pelatihan.

3.1.2.2.1 Menghapus data non-sentimen

Dari data yang telah diunduh sebelumnya akan dilakukan penyaringan sehingga data yang tidak memiliki *subtopics* Sentimen akan dihilangkan.

3.1.2.2.2 Pemisahan data

Dataset yang telah disaring akan dipisah menjadi dua kategori data yaitu *train* dan *test.* Data *train* akan digunakan untuk proses selanjutnya sedangkan data *test* akan digunakan untuk melakukan validasi terhadap model.

3.1.2.3 Ekstrasi fitur

Setelah melalui beberapa pembersihan data. Beberapa ciri dari data yang dapat digunakan untuk merepresentasikan data akan digunakan sebagai masukan dalam pelatihan.

3.1.2.3.1 N-gram tokenization

Pada tahapan ini, setiap data akan dipecah menjadi beberapa kata atau frasa dengan menggunakan metode n-gram khususnya trigram. Di mana kata yang dipecah akan berjumlah maksimal 3 kata dalam satu frasa. Hasil dari proses tokenisasi ini akan dijadikan sebagai fitur untuk melatih model. Di bawah ini merupakan contoh tokenisasi trigram. Dari hasil tokenisasi tersebut akan dilakukan pembobotan berdasarkan frekuensi munculnya kata dalam kalimat terhadap keselurahan data. Pembobotan menggunakan algoritma Term Frequency Inverse Document Frequency (TF-IDF).

Table 3.4 Proses tokenizing

Sebelum *Pre-processing*	Setelah *Pre-processing*
Aku seneng deh bisa presentasi depan teman-teman Polyglot.	{ Aku seneng deh: 1, seneng deh bisa: 1, deh bisa presentasi: 1, bisa presentasi depan: 1, presentasi depan teman-teman: 1, depan teman-teman Polyglot: 1. }
Mau tau aja sih lu	{ Mau tau aja: 2, tau aja sih: 1, aja sih lu:1 }

3.1.2.3.2 Kata positif dan negatif

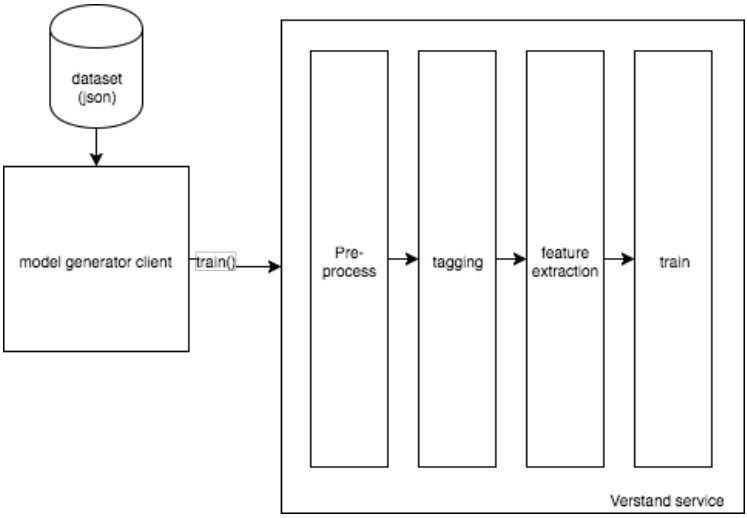
Pada tahapan ini setiap kata pada hasil tokenisasi akan dilakukan pengecekan apakah terdapat kata-kata yang memiliki yang terdapat pada corpus positif atau negatif. Fitur ini akan menghitung kata positif dan negatif yang terdapat pada teks, dimana nantinya kata yang merupakan positif atau negatif akan diganti dengan corpus itu sendiri.

3.1.2.3.3 *Part of speech tagging*

Pada tahapan ini akan dilakukan proses *tagging* pada kata atau frasa yang telah ditokenisasi. Proses tagging akan menggunakan POS *Tagger* yang telah dikembangkan oleh Kata.ai menggunakan algoritma *Recurrence Neural Network* (RNN). Kata pada hasil tokenisasi yang memiliki *tag* tertentu akan dihapus dan diganti dengan sebuah nilai dari *tag* tersebut.

3.1.2.3.4 Pelatihan model

Data yang telah melewati *pre-processing* akan dibuat model menggunakan algoritma *Logistic Regression* dan *Fasttext* sesuai dengan skenario yang telah dibuat pada tahap perancangan model. Tujuan mengubah parameter dari proses *training* untuk mengetahui fitur terbaik mana yang akan digunakan sebagai bahan membuat model klasifikasi sentimen. Dibawah ini merupakan *flow* dari dataset sampai dengan model klasifikasi sentimen.



Gambar 3.3 Pembuatan model klasifikasi sentimen dengan *verstand service*

Pada proses ini, dari dataset akan dilakukan akan dibuat sebuah program *client* untuk memudahkan proses pelatihan data menggunakan API dari *verstand service*. Dari client tersebut

data akan diteruskan menuju *verstand service* dengan proses pertama yaitu *pre-processing* setelah itu kata pada teks akan diberikan sebuah tag menggunakan *part of speech tagger*. Setelah proses tagging selesai fitur akan diekstrasi dengan mengganti kata yang teridentifikasi sebagai *part of speech* menjadi label dari *part of speech* itu sendiri. Setelah selesai maka model akan dilatih menggunakan algoritma pembelajaran mesin seperti *Logistic Regression* dan *Fasttext*. Hasil dari model tersebut akan disimpan ke dalam *folder* sehingga model dapat digunakan secara berulang dan bersifat *persistent*.

3.1.2.4 Analisa model klasifikasi sentimen

Berdasarkan tahapan sebelumnya telah dibuat skenario dan pemisahan data *training & test* dengan skala 2:1. Skala tersebut berdasarkan peneltian [27] yang telah membuktikan bahwa untuk algoritma klasifikasi nilai yang standar dan hampir mencapai optimal adalah 2:1. Pada penelitian [27] menggunakan metode *cross-validation* oleh karena itu akurasi dari model akan diuji menggunakan *cross-validation*. Metode ini akan melihat akurasi dan presisi dari model dengan membagi dataset menjadi *train & test,* setelah model dibuat maka akan dilakukan validasi menggunakan data *train* itu sendiri dan data *test* pada masing-masing skenario. Setelah itu model akan dibandingkan untuk dipilih agar dapat diimplementasikan pada *web service.*

3.1.2.5 Perancangan *web service*

Pada tahap ini akan digambarkan secara rinci arsitektur, teknologi dan arus aplikasi sehingga dapat mengakses model klasifikasi sentimen.

3.1.3 Implementasi

Setelah proses pembuatan klasifikasi sentimen selesai dibuat dan dipilih model yang akan digunakan, *web service* akan mulai dilakukan. Model klasifikasi sentimen yang dibuat menggunakan layanan verstand dari Kata.ai akan diakses melalui *web service* oleh pengguna akhir. Oleh karena itu web service akan berguna sebagai media model klasifikasi antar

pengguna dan *verstand service* yang menyediakan model klasifikasi sentimen.

3.1.3.1 Pembuatan web service

Web service yang telah dirancang pada tahap sebelumnya akan mulai dibangun sehingga nantinya *web service* akan dapat mengakses model klasifikasi sentimen yang telah dibuat sebelumnya.

3.1.3.2 Deployment

Pada tahap ini aplikasi *web service* dan layanan verstand akan diubah ke dalam bentuk *image* untuk memudahkan proses *deployment* menggunakan Docker. Aplikasi yang telah diubah menjadi *image* akan dijalankan ke dalam lingkungannya masing-masing sehingga dibutuhkan beberapa pengaturan pada tiap *image* aplikasi.

3.1.4 Pengujian

Setelah model klasifikasi dan *web service* selesai dibangun, maka akan dilakukan pengujian terhadap dua keluaran tersebut.

3.1.4.1 Pengujian unit web service

Jenis pengujian ini dinamakan *unit testing* yaitu, pengujian fungsionalitas yang diuji pada lingkungan tertutup dan tidak terintegrasi dengan layanan lainnya. Hasil dari pengujian ini bertujuan untuk menentukan apakah program yang telah dibuat telah sesuai dan berjalan secara fungsionalitas.

3.1.4.2 Pengujian integrasi

Pengujian pada tahapan ini akan menggunakan dasar dari analisa kebutuhan atau kebutuhan fungsional. Kebutuhan fungsional ini nantinya akan dijadikan skenario untuk melakukan pengujian..

3.1.5 Dokumentasi

Pada tahapan terakhir ini akan dilakukan pembuatan laporan dalam bentuk buku tugas akhir yang disusun sesuai format yang telah ditentukan. Buku ini berisi dokumentasi langkah-langkah pengerjaan tugas akhir secara rinci. Buku ini diharapkan dapat

bermanfaat sebagai referensi untuk pengerjaan penelitian lain, serta sebagai acuan untuk pengembangan lebih lanjut terhadap topik pengembangan yang serupa.

BAB IV
PERANCANGAN

Pada bab ini akan membahas mengenai perancangan dari luaran tugas akhir ini.

4.1 Analisis kebutuhan

Tahap awal dari perancangan dimulai dari mengumpulkan informasi mengenai kondisi dan teknologi yang digunakan oleh perusahaan Kata.ai. Pengambilan informasi dilakukan dengan menggunakan dua buah metode yaitu observasi dan wawancara.

4.1.1 Observasi

Metode ini dilakukan dengan mengamati dokumentasi mengenai penggunaan dari *verstand service* melalui website kata.quip.com, Bitbucket.org. Dokumentasi tersebut meliputi kode aplikasi perusahaan dan penjelasan dan tata cara instalasi aplikasi. Dari hasil observasi tersebut terdapat tumpukan teknologi yang digunakan oleh *verstand service* yaitu,

Table 4.1 Arsitektur layanan verstand

No	Tumpukan Teknologi	Teknologi yang digunakan
1	Bahasa Pemrograman	Python
2	*Framework*	Flask
3	*Runtime Environment*	Python2.7
4	Basis data	MongoDB
5	Daftar *library*	- Flask:0.11.1 - gevent:1.1.2 - pymongo:3.3.0 - celery:4.0.2 - scikit-learn:0.18.1 - keras:2.0.5 - gensim:1.0.1 - hickle:2.1.0

		- dill:0.2.6 - pystruct:0.2.4 - cvxopt:1.1.8 - python-parallel-collections:2.0.0 - tqdm:4.14.0 - pandas:latest - numpy:latest - h5py:latest - nltk:latest - plac:latest

4.1.2 Wawancara

Wawancara akan dilakukan kepada salah satu karyawan Kata.ai yang memiliki peran sebagai *Machine Learning Engineer* yaitu Fariz Ikhwantri. Dalam wawancara ini hal yang ditanyakan adalah bahasan seputar kekurangan dari *verstand service* dalam membangun model pembelajaran mesin. Dari hasil wawancara tersebut terdapat beberapa kekurangan dari *verstand service* sehingga dari sana didapat beberapa fungsional dan non-fungsional yang dibutuhkan yaitu:

1. Fungsional

Table 4.2 Kebutuhan fungsional

Kode	Deskripsi
F1	Aplikasi dapat digunakan untuk mengklasifikasikan kata, kalimat atau paragraf ke dalam sebuah kelas sentimen.
F2	Aplikasi dapat mengklasifikasikan teks ke dalam sentimen menggunakan model hirarki
F3	Aplikasi dapat menggabungkan beberapa model menjadi klasifikasi hirarkial

2. Non-fungsional
    - Aplikasi dapat diakses melalui *web service*

- Model klasifikasi sentimen memiliki tingkat akurasi yang baik.

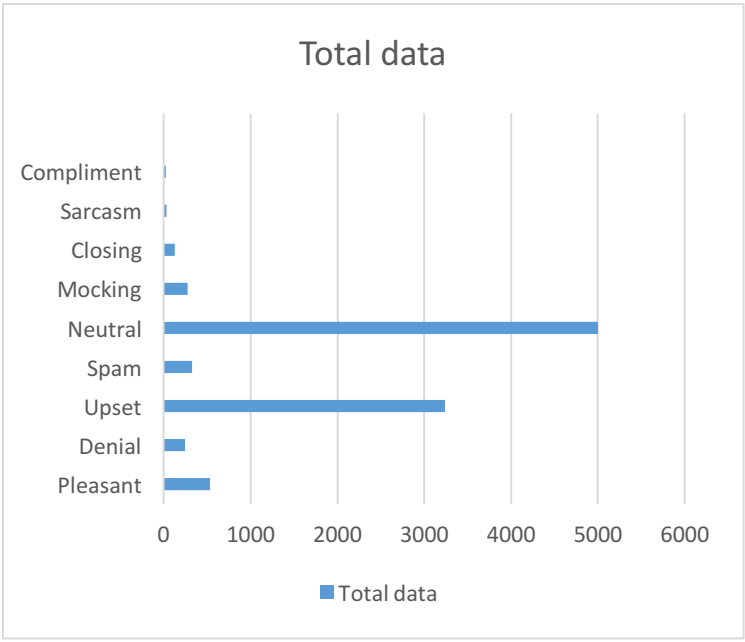
4.1.3 Pengunduhan data

Data yang digunakan untuk melakukan pelatihan terhadap model klasifikasi sentimen telah disediakan oleh Kata.ai. Data tersebut dapat diunduh melalui website annotator.kata.ai. Spesifikasi data tersebut ialah:

- Nama file: message.json
- Ukuran: 5.8 MB
- Ekstensi: json
- Total data: 9806 teks

Table 4.3 Total kelas sentimen

Sentimen	**Total**
Pleasant	536
Denial	244
Upset	3238
Spam	327
Neutral	5004
Mocking	278
Closing	128
Sarcasm	28
Compliment	23
	9806

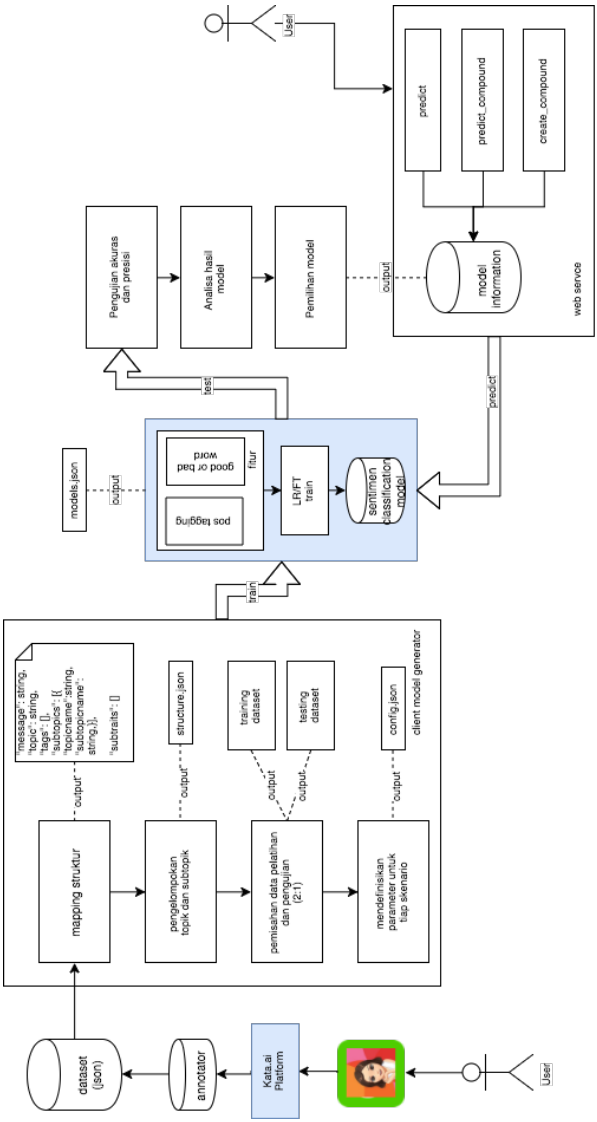


Grafik 4.1 Jumlah data tiap kelas

Dari Grafik 4.1 Jumlah data tiap kelas dan Table 4.3 Total kelas sentimen terlihat bahwa persebaran dari kelas tidak seimbang dengan jumlah kelas tertinggi sebanyak 5004 data pada kelas *neutral* sedangkan sebanyak 23 data pada kelas *compliment.*

4.2 Desain sistem

Pada tahap sebelumnya telah diketahui kebutuhan fungsional dan non-fungsional. Sehingga dalam tahap ini akan dirancang pengembangan aplikasi klasifikasi sentimen sesuai dengan kebutuhan. Secara garis besar perancangan akan meliputi pembuatan skenario pembuatan model klasifikasi sentimen dan aplikasi *web service* sebagai media untuk mengakses model klasifikasi sentimen.



Gambar 4.1 Alur aplikasi

Gambar 4.1 menjelaskan bagaimana aplikasi akan dibuat dan dapat dipergunakan oleh pengguna untuk melakukan klasifikasi sentimen terhadap teks. Dataset diambil dari hasil percakapan

antara pengguna dengan chatbot Jemma, setelah itu teks tersebut disimpan dan dikelompokan sesuai dengan sentimennya oleh Kata.ai. Data yang telah dikelompokan tersebut adalah data yang digunakan untuk melatih model klasifikasi sentimen. Sebelum dataset dilatih pada *verstand service* dibuat sebuah *script client model generator* untuk melakukan pre-processing dan pengaturan skenario pelatihan. Setelah semua model klasifikasi sentimen dibuat maka model akan disimpan dalam *verstand service.* Setelah itu model akan divalidasi dengan menghitung akurasi dan presisi pada data pelatihan dan pengujian. Model klasifikasi yang paling optimal akan diimplementasikan pada *web service* dengan memasukan informasi model ke dalam basis data *web service*. Pengguna akhir dapat mengakses *web service* dengan menuliskan url dan prefix predict untuk melakukan klasifikasi teks. *Web service* akan memanggil model klasifikasi yang ada pada *verstand service* untuk mengklasifikasikan teks. Hasil dari klasifikasi tersebut akan diteruskan *web service* ke pengguna akhir.

4.2.1 Perancangan model klasifikasi

Untuk memberikan performa yang baik pada sebuah model klasifikasi skenario untuk membandingkan performa perlu dilakukan. Sesuai dengan data yang telah diunduh, klasifikasi akan dibagi ke dalam 9 kelas yaitu *upset, mocking, sarcasm, denial, closing, neutral, pleasant,* dan *compliment*. Model klasifikasi yang akan dibuat akan memiliki beberapa parameter yang berbeda saat melakukan proses pelatihan. Skenario tersebut ialah sebagai berikut.

Table 4.4 Skenario

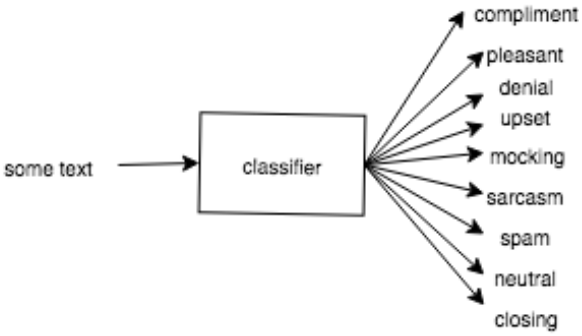
Skenario	Logistic Regression	Fasttext	Kata positif dan negatif	POS tag
1	x			
2	x		x	
3	x			x

4	x		x	x
5		x		
6		x	x	
7		x		x
8		x	x	x

Skenario pada Table 4.4 Skenario akan digunakan aplikasikan pada pembuatan model datar dan model hirarki. Dari hasil skenario tersebut akan dianalisa performa dan dipilih yang terbaik.

4.2.1.1 Model datar

Model datar adalah sebuah metode pengklasifikasian menggunakan satu model yang akan melakukan 9 klasifikasi sekaligus.



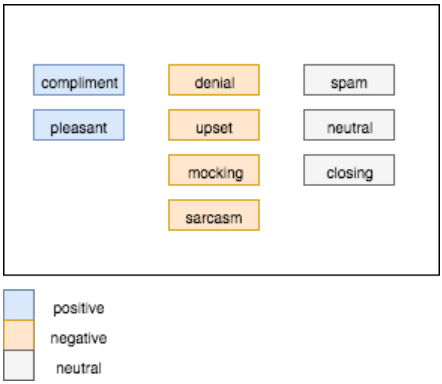
Gambar 4.2 Alur klasifikasi model datar

Pada pembuatan model ini prediksi teks hanya akan dilakukan oleh satu model klasifikasi.

4.2.1.2 Model hirarki

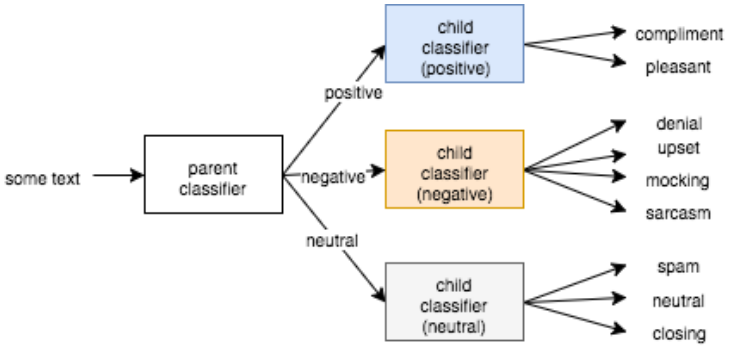
Semakin banyaknya kelas yang ingin diklasifikasikan maka semakin tinggi kemungkinan model untuk melakukan kesalahan klasifikasi. Oleh karena itu, skenario ini akan

mengelompokkan 9 kelas sentimen ke dalam kelas yang lebih tinggi.



Gambar 4.3 Pengelompokan kelas sentimen

Pembuatan model ini akan dilakukan sebanyak dua kali yaitu untuk pembuatan model *parent* dan *child classifier*. *Parent classifier* akan mengelompokkan data menjadi 3 kelas yaitu positif, negatif dan netral. Setelah itu teks akan diteruskan untuk diprediksi menggunakan *child classifier* yang sesuai dengan hasil klasifikasi pada *parent classifier*.



Gambar 4.4 Alur klasifikasi model hirarki

4.2.2 Perancangan *Pre-processing* Data

Sebelum diproses, data harus terlebih dahulu mengalami proses persiapan data. Data yang telah diunduh pada tahap sebelumnya

perlu dilakukan persiapan sebelum dapat diproses. Berikut ini merupakan tahapan yang dilakukan pada pra-pemrosesan data berikut.

4.2.2.1 Perancangan penyaringan kelas sentimen

Pada data yang telah diunduh data masih memiliki banyak label tiap teksnya. Maka pada tahap ini dilakukan penyaringan terhadap teks yang memiliki sentimen diantara 9 kelas yang telah disebutkan diatas. Kelas akan disaring sehingga keluaran memiliki struktur seperti Kode 4.1.

Kode 4.1 Struktur dataset

```
{
   "topic": string,
   "message": string
   "subtopics": [{
      "topicname": "positive" | "negative" | "neutral"
      "subtopicname": "pleasant" | "denial" | "upset" | "spam"
| "neutral" | "mocking" | "closing" | "sarcasm" | "compliment"
   }],
   "subtraits": Array<T>,
   "tags": Array<T>
}   }
```

4.2.2.2 Perancangan pemisahan data

Setelah melakukan penyaringan dan memiliki keluaran seperti Kode 4.1 Struktur dataset maka dataset perlu dibagi menjadi data pelatihan dan data pengujian. Data pelatihan nantinya akan digunakan sebagai masukkan untuk melatih model klasifikasi. Data akan dipisah dengan proporsi data pelatihan:data pengujian sebanyak 2:1 dari total data pada kelas tersebut. Data pengujian dipilih secara acak dengan jarak sebanyak total data tiap kelas. Sehingga pemisahan data akan tampil seperti

Table 4.5 Pembagian data pelatihan dan pengujian

Kelas	Jumlah data		
	Pelatihan	**Pengujian**	**Total**
Pleasant	357	179	536
Denial	162	82	244
Upset	2158	1080	3238
Spam	218	109	327
Neutral	3336	1668	5004
Mocking	185	93	278
Closing	85	43	128
Sarcasm	18	10	28
Compliment	15	8	23
	6534	**3272**	**9806**

4.2.2.3 Perancangan kata positif dan negatif

Pada tahap ini, akan digunakan *corpus* dari sumber terbuka pada github yang dimiliki oleh Devid Haryalesmana. Data terdiri dari dua buah *corpus* yaitu *opinion words*. *Opinion words* akan digunakan pada model *logistic regression* dan *fasttext*. Corpus *opinion words* terdiri dari opini positif dan negatif. Bila kata dalam teks merupakan opini positif atau negatif pada *corpus* maka kata tersebut akan diganti dengan nilai *positive* dan *negative.*

Table 4.6 Skema kata positif dan negatif

Masukan	**Kata dalam *corpus* positif**	**Kata dalam *corpus* negatif**	**Keluaran**
Kamu tampan tapi nakal	[“tampan”]	[“nakal”]	Kamu positive tapi negative

Asdasdsa nggak	[]	[]	Asdasdsa nggak

4.2.3 Perancangan ekstrasi fitur

Pada tahap ini fitur yang digunakan sebagai bahan masukan untuk proses pembelajaran mesin akan didefinisikan. Fitur akan direpresentasikan menggunakan *dense vector* sehingga fitur hanya berupa satu dimensi namun dimensi tersebut merupakan gabungan dari beberapa fitur yang telah dipilih.

4.2.3.1 Perancangan tokenisasi

Setelah semua *pre-processing* data telah selesai, maka selanjutnya adalah proses tokenisasi. Proses ini bertujuan untuk menjadikan satu data teks menjadi token-token yang digunakan pada pembelajaran *logistic regression* agar dapat diproses lebih lanjut menggunakan *Term Frequency Inverse Document Frequency*. Tokenisasi ini dilakukan dengan cara menggunakan class TfidfVectorized yang telah tersedia pada library sklearn. Tfidf akan merubah string menjadi bentuk vektor angka dimana beberapa parameter yang diperlukan untuk memanggil class ini ialah *ngram_range*, *max_features*, dan *min_word*. Ngram_range berarti penggunaan tuple n-gram, sedangkan max_features akan membatasi fitur yang akan dihasilkan pada saat proses tokenisasi. Min_word akan membatasi minimal kata yang masuk ke dalam semua dokumen. Pada pembelajaran fasttext parameter yang digunakan hanya ngram_range. Parameter yang akan digunakan yaitu,

- "ngram_range": [1, 3], // 1 – 3 gram
- "min_word": 2,
- "max_features": 10000

4.2.3.2 Perancangan POS *Tagging*

Pada proses ini akan dirancang sebuah *tagger* yang digunakan untuk memberikan kelas pada sebuah kata. Proses ini dilakukan dengan cara *parsing,* kemudian ditentukan kelas yang berdasarkan Kamus Besar Bahasa Indonesia (KBBI). Pembuatan POS Tagging ini menggunakan algoritma

Recurrence Neural Network buatan Kata.ai. Terdapat 22 kelas yang digunakan untuk memberikan *tag* pada kata yaitu [“SYM”, “NN”, ”NOUN”, “PRONOUN”, “WP”, “VBT”, “VBI”, “MD”, “JJ”, “NEG”, “IN”, “CC”, “SC”, “UH”, “DT”, “DT”, “RP”, “FW”, “CDI”, “NUM”, “RB”, “X”]. Namun untuk menggunakan POS Tagging sebagai salah satu fitur pada klasifikasi sentimen kelas POS Tag yang akan digunakan hanya sebagian yaitu

- NEG (negasi)
- NOUN (kata benda)
- PRONOUN (kata ganti)
- CC (*coordinate conjunction*)
- WP (WH *Pronouns*)

Setiap kata yang memiliki satu diantar 5 kelas akan diganti dengan kelas tersebut.

Table 4.7 Skema POS Tagging

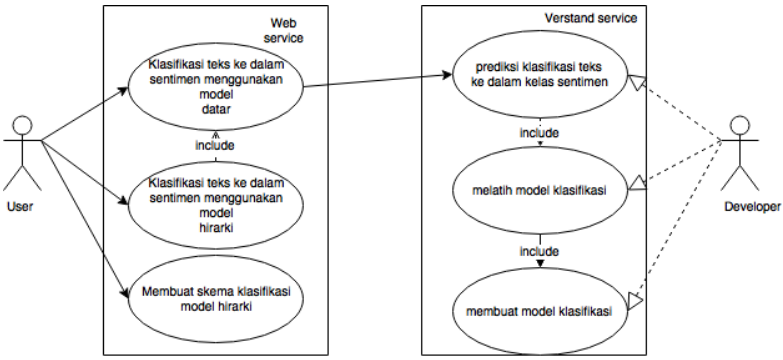
Masukan	***Tagging***	**Keluaran**
Saya tidak mau makan	[[['Saya', 'PRONOUN'], ['tidak', 'NEG'], ['mau', 'VBT'], ['makan', 'VBT']]]	PRONOUN NEG mau makan
Namaku jemma	[[['Namaku', 'WP'], ['jemma', 'NOUN']]]	WP NOUN

Dapat dilihat bahawa penggantian dilakukan per kata sehingga menemukan kelas yang terdapat diantara 4 kelas *POS Tagging*. Penggantian kata tersebut dinamakan proses *masking*. Hal

tersebut bertujuan untuk memberikan ekstrasi fitur dalam bentuk vektor.

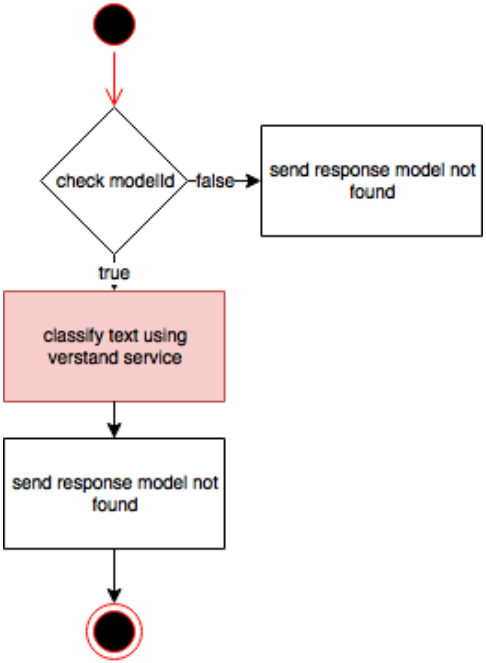
4.2.4 Perancangan *web service*

Pada bagian ini akan dijelaskan mengenai perancangan aplikasi *web service* dari teknologi sampai dengan alur aplikasi yang akan digunakan. *web service* akan menghubungkan antara pengguna dengan *verstand service*. Fungsionalitas pada *web service* akan dibuat sesuai dengan kebutuhan fungsional yang telah didapatkan pada bab sebelumnya, Gambar 4.5 Usecase diagram menunjukan fungsionalitas dari *web service* dari sudut pandang pengguna dan pengembang.



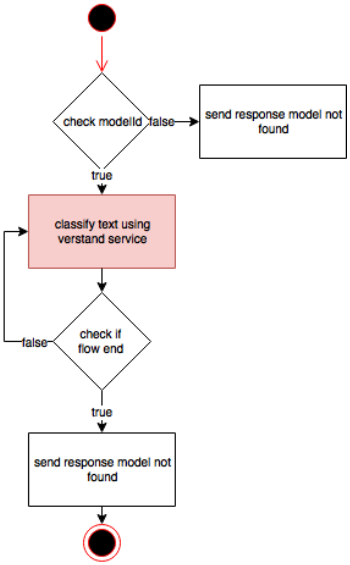
Gambar 4.5 Usecase diagram

Secara umum untuk mengakses *web service* ini memiliki 3 fungsi utama yang dapat dimanfaatkan oleh pengguna menggunakan protocol http.



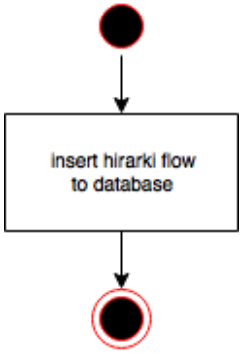
Gambar 4.6 f1 activity diagram

Gambar 4.6 menjelaskan alur aktifitas dari fungsional satu yaitu dengan melakukan pengecekan terhadap modelId yang diinputkan oleh pengguna, apabila modelId tidak ditemukan maka sistem akan mengembalikan respon model tidak tersedia. Setelah itu bila model ditemukan maka akan diteruskan pada *verstand service* untuk dilakukan klasifikasi, hasil dari klasifikasi akan diberikan kembali pada pengguna. Gambar 4.8



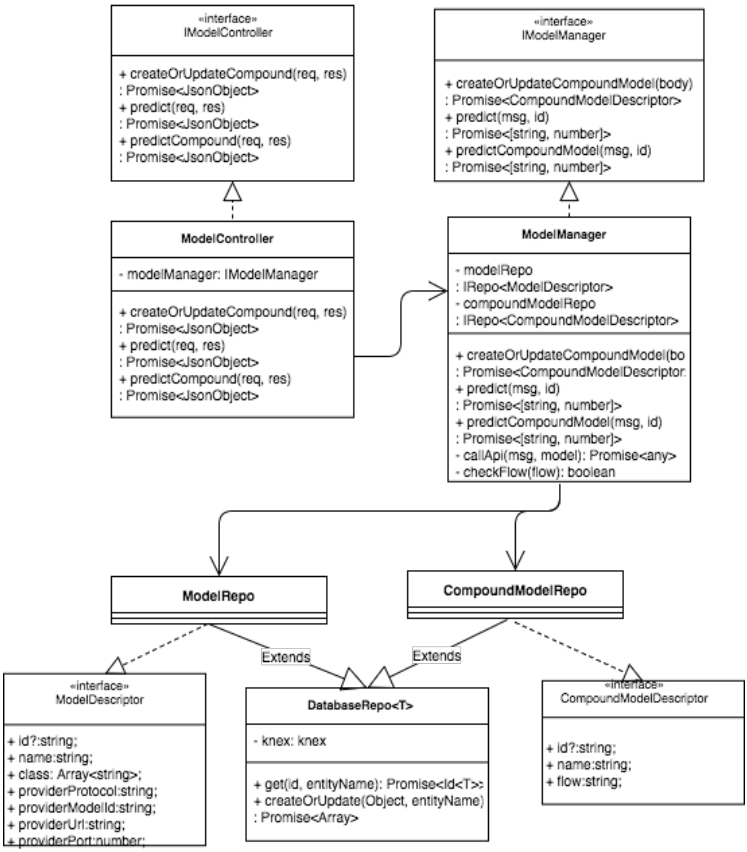
Gambar 4.7 f2 activity diagram

Gambar 4.7 menjelaskan alur aktifitas dari fungsional dua yaitu dengan melakukan pengecekan terhadap modelId yang diinputkan oleh pengguna, apabila modelId tidak ditemukan maka sistem akan mengembalikan respon model tidak tersedia. Setelah itu bila model ditemukan maka akan diteruskan pada *verstand service* untuk dilakukan klasifikasi, hasil dari klasifikasi akan diperiksa apakah alur klasifikasi hirarki telah berakhir, bila tidak maka sistem akan melakukan klasifikasi lagi menggunakan *verstand service* sampai alur hirarki terakhir. Setelah itu hasil klasifikasi terakhir akan diberikan pada pengguna.



Gambar 4.8 f3 activity diagram

Gambar 4.8 menggambarkan alur ketika pengguna menggunakan sistem untuk membuat alur klasifikasi secara hirarki menggunakan model-model yang tersedia pada *web service.* Sedangkan Gambar 4.9 menggambarkan secara umum kelas yang akan digunakan dalam mengembangkan aplikasi *web service* tersebut. Aplikasi terbagi menjadi 3 kelas besar yaitu modelController, modelManager dan modelRepo. Ketiga model tersebut memiliki kelas *interface* yang berbeda-beda. Sehingga penggunaan fungsi dari kelas tersebut juga memiliki perbedaan. Kelas interface didefinisikan untuk memudahkan pengembang dalam mengembangkan sebuah fitur baru kedepannya. Kelas modelRepo dan compoundModelRepo hanya memiliki *parent class* tanpa sebuah method, karena pada kali ini pemanfaatan kelas tersebut sudah dapat diakses melalui kelas *parent* yaitu databaseRepo.

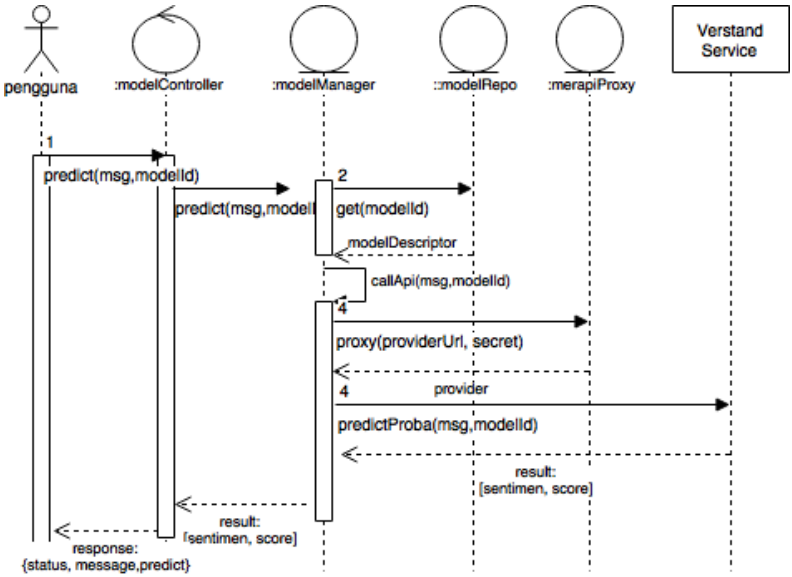


Gambar 4.9 Class diagram

Setelah Gambar 4.9 menjelaskan mengenai gambaran umum kelas. Maka selanjutnya merupakan spesifikasi dari 3 fungsional yang akan dibuatkan skema akses http dengan diikuti oleh alur *sequence diagram*. Berikut adalah fungsional utama dan alur hubungan antar kelas dan *verstand service* sehingga dapat digunakan sesuai dengan kebutuhan fungsional.

1. GET /predict/{modelId}
   Method ini digunakan untuk mengakses model klasifikasi non-hirarki

Parameter:
modelId: <string> *required
Query:
msg: <string> *required

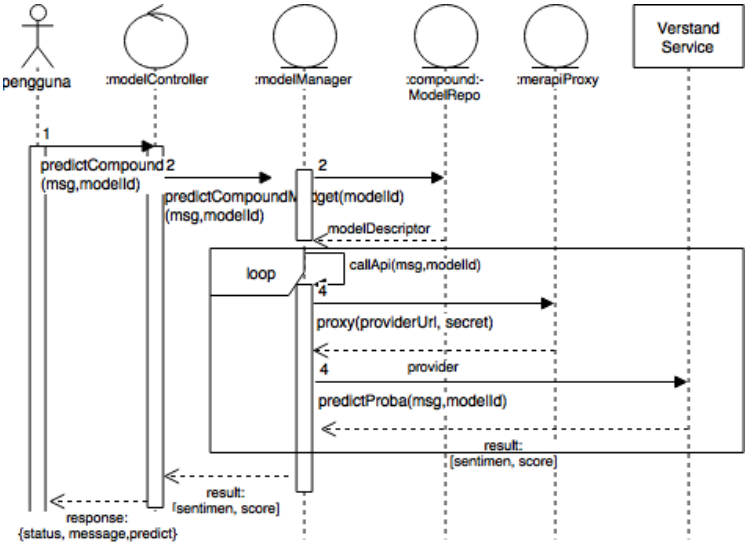


Gambar 4.10 Predict sequence diagram

Gambar 4.10 menjelaskan untuk mengakses *web service* pengguna harus memasukan uri dengan method *get* dan dua buah parameter yaitu modelId dan msg yang berisi teks yang ingin diklasifikasikan. Request akan diterima oleh kelas modelController untuk menangani input dari user setelah itu dipanggil method predict pada modelManager, dimana modelManager akan memastikan modelId tersedia dan memanggil model klasifikasi sentimen dengan method predictProba pada *verstand service* menggunakan merapi-proxy. Setelah itu hasil dari prediksi model akan diberikan sebagai respon pada pengguna.

2. GET /predict/compound/{modelId}

Method ini digunakan untuk mengakses model klasifikasi hirarki
Parameter:
modelId: <string> *required
Query:
msg: <string> *required



Gambar 4.11 Predict hirarki sequence diagram

Gambar 4.11 menjelaskan untuk mengakses *web service* pengguna harus memasukan uri dengan method *get* dan dua buah parameter yaitu modelId dan msg yang berisi teks yang ingin diklasifikasikan. Request akan diterima oleh kelas modelController untuk menangani input dari user setelah itu dipanggil method predict pada modelManager, dimana modelManager akan memastikan modelId tersedia. Setelah mendapatkan deskripsi dari alur klasifikasi hirarki maka teks akan diloop menggunakan beberapa model klasifikasi sesuai dengan alur yang telah didefinisikan. Tiap model klasifikasi akan ditembakan pada *verstand service.*

3. POST /compound/{compoundId}

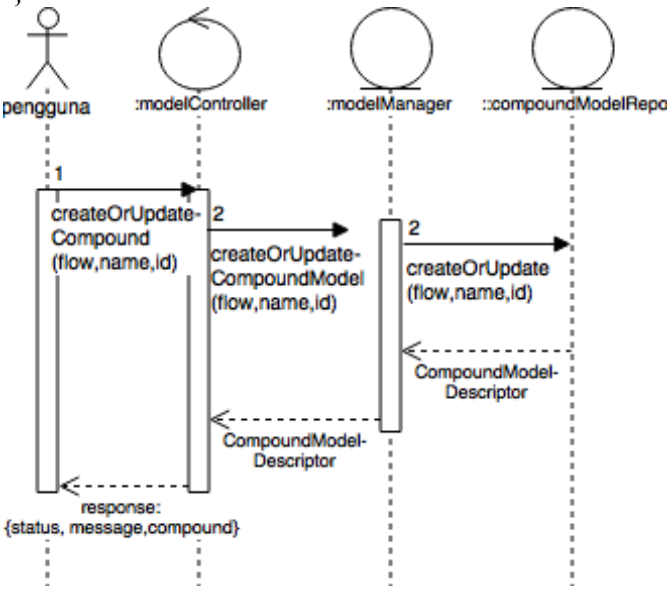
Method ini digunakan untuk membuat flow model klasifikasi hirarki

Parameter:

compoundId: <string>

body(json):

```
{
    "name": string,
    "flow":
    {
        "model_id": string,
        "class":{
            <classification_label>: string,
            <classification_label>: string,
            <classification_label>: string
        }
    }
}
```



Gambar 4.12 Alur membuat model hirarki

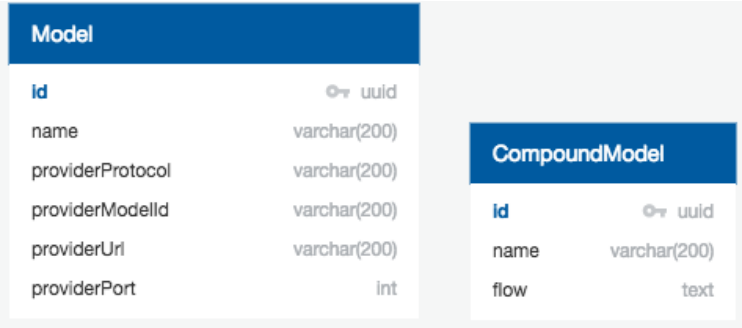
Pada Gambar 4.12 menjelaskan pengguna dapat membuat alur klasifikasi hirarki dengan mengakses prefix compound/ dengan method *post*. Beberapa parameter yang dibutuhkan adalah *flow, name* dan id. ModelController akan menangani request dari pengguna setelah memvalidasi request maka method createOrUpdateCompoundModel pada modelManager akan dipanggil. Data akan dimasukan ke dalam basis data dan memberikan respon seperti masukan dari pengguna itu sendiri. Table 4.8 menunjukan teknologi yang digunakan untuk membuat *web service* ini.

Table 4.8 Teknologi yang digunakan untuk web service

No	Tumpukan Teknologi	Teknologi yang digunakan
1	Bahasa Pemrograman	Javascript
2	*Framework*	merapi
3	*Runtime Environment*	NodeJS
4	Basis data	MySQL
5	Daftar *library*	- Merapi - Knex - MomentJS - MySQL - TypeScript - Yaml - Request - ExpressJS - mocha

4.2.4.1 Perancangan basis data

Pada bagian ini akan dijelaskan mengenai desain basis data yang memiliki tujuan merancang basis data yang terstruktur dengan membuat relasi antar tabel yang pada pengerjaannya memiliki berbagai tahapan agar kolom-kolom dalam tabel tersebut hanya berisikan data-data yang tidak redundan.



Gambar 4.13 Skema database

4.2.4.2 Perancangan skenario pengujian

Terdapat dua buah pengujian yang akan dilakukan yaitu pengujian unit dan integritas. Pada *activity diagram* dan *sequence diagram* yang dibuat sebelumnya dideskripsikan penggunaan alur aktifitas dan alur hubungan antar kelas dalam menyelesaikan kebutuhan fungsional. Dari hal tersebut terlihat kelas modelManager merupakan kelas utama yang menghubungkan antara *verstand service* dan aplikasi. Sehingga dari kelas tersebut akan dilakukan pengujian unit. Unit yang digunakan untuk pengujian ini adalah sebuah *method* yang terdapat pada kelas tersebut. Gambar 4.9 memperlihatkan kelas yang dapat diuji pada modelManager adalah predict, predictCompoundModel dan createOrUpdateCompoundModel. Berikut merupakan *mapping* dari pengujian unit yang dilakukan.

Table 4.9 Mapping skenario

Kode	Fungsional	Deskripsi alur	Tipe alur
C1	F1	Teks berhasil diklasifikasikan menggunakan model datar	*Basic*
C2	F1	Id model datar tidak ditemukan	*Alternate*

C3	F2	Teks berhasil diklasifikasikan menggunakan model hirarki	*Basic*
C4	F2	Id model hirarki tidak ditemukan	*Alternate*
C5	F3	Alur klasifikasi dimasukkan dalam basisdata	*Basic*

Setelah mengetahui skenario alur *basic* dan *alternate.* **Table 4.10** dilakukan mapping terhadap method yang tersedia pada kelas modelManager untuk mengetahui fungsional dieksekusi oleh method apa dalam kelas modelManager.

Table 4.10 Mapping method dan fungsional

Fungsional	**Method**	**Parameter**
F1	predict	msg, modelId
F2	predictCompoundModel	msg, modelId
F3	createOrUpdateCompoundModel	flow

Tiap method dalam kelas tersebut memiliki parameter, oleh karena itu pengujian unit prediksi menggunakan model datar pada method akan digunakan parameter modelId yang berbeda seperti didefinisikan pada Table 4.11.

Table 4.11 Skenario unit test f1

Case	**Parameter**		**Expected result**
	msg	**modelId**	
C1	Halo	topic	return sentiment=neutral
C2	Halo	-	return null

Sedangkan unit test untuk menguji fungsional dapat melakukan klasifikasi sentimen menggunakan model hirarki didefinisikan pada Table 4.12.

Table 4.12 Skenario unit test f2

Case	Parameter		Expected result
	msg	**modelId**	
C3	Halo	topic	return sentiment=neutral
C4	Halo	-	return null

Pengujian unit Table 4.13 merupakan alur *basic* dengan mendefinisikan masukan dan keluaran yang diharapkan pada pengujian dalam membuat alur model klasifikasi.

Table 4.13 Skenario unit case f3

Case	Flow	Expected result

C5	{
	id: "test",
	name: "test",
	flow: {
	model_id: "topic",
	class: {
	compliment: "subtopic"
	}
	}
	}
		return id=test

4.3 Pembuatan model klasifikasi sentimen

Untuk memudahkan proses pembuatan model maka dibuat sebuah script sederhana untuk menampung parameter dan dataset dalam bentuk config yang menggunakan bahasa

pemrograman JavaScript. Terdapat 8 skenario masing-masing akan memiliki parameter yang berbeda yaitu.

```
"logr": {
        "type": "tfidf",
        "solver": "logr",
        "normalize_word": true,
        "params": {
          "penalty": "l2",
          "solver_opt": "newton-cg",
          "use_punctuation": true,
          "multi_class": "multinomial",
          "ngram_range": [1, 3],
          "min_word": 2,
          "max_features": 10000,
          "class_weight": "balanced"}}
```

Kode 4.2 Parameter skenario 1

Kode 4.2 menjelaskan data akan dilatih menggunakan algoritma *logistic regression* yang akan dinormalisasi terlebih dahulu. Seperti halnya skenario yang telah dibuat sebelumnya pada skenario ini tidak menggunakan *tagging* dan penggantian kata baik dan kata yang buruk. Karena data memiliki lebih dari kelas biner maka parameter yang dimasukkan ialah *newton-cg* dan *multinomial*. Parameter ini merupakan parameter bawaan dari *library scikit-learn* yang diperuntukan klasifikasi *multiclass*.

```
"logr-repl": {
        "type": "tfidf",
        "solver": "logr",
        "normalize_word": true,
        "use_punctuation": true,
        "replace": {
          "positive": ["a+", "acungan jempol”,…,  "yihaa"],
          "negative": ["abnormal", "absurd”,…, "Yahudi"]
        },
        "params": {
```

```
            "penalty": "l2",
            "solver_opt": "newton-cg",
            "multi_class": "multinomial",
            "ngram_range": [1, 3],
            "min_word": 2,
            "max_features": 10000,
            "class_weight": "balanced"

        }
    }
```

Kode 4.3 Parameter skenario 2

Kode 4.3 menjelaskan bahwa data akan dilatih menggunakan algoritma *logistic regression*. Semua parameter yang dimasukkan sama dengan pada Kode 4.2 perbedaannya ialah pada salah satu kunci *replace* yang merupakan salah satu fitur yang digunakan pada algoritma ini untuk merubah kata yang diidentifikasi sebagai kata positif/negatif menjadi label positif/negatif itu sendiri.

```
"logr-repl-rnn": {
        "type": "tfidf",
        "solver": "logr",
        "normalize_word": true,
        "use_punctuation": true,
        "kind": "compound-classifier",
        "normalize_word": true,
        "tagger_model_id":      "7064187d-3e0e-429c-b7d1-
a28608f3d9c0",
        "mask": ["NEG", "NOUN", "PRONOUN", "WP",
"CC"],
        "params": {
            "penalty": "l2",
            "solver_opt": "newton-cg",
            "multi_class": "multinomial",
            "ngram_range": [1, 3],
            "min_word": 2,
            "max_features": 10000,
            "class_weight": "balanced"}}
```

Kode 4.4 Parameter skenario 3

Kode 4.4 menjelaskan bahwa data akan dilatih menggunakan algoritma *logistic regression*. Semua parameter yang dimasukkan sama dengan pada Kode 4.2 dengan tambahan fitur *tagging*. Sebuah *tagger* yang akan digunakan merupakan *tagger* milik Kata.ai yang dapat akan memberikan label pada kalimat sesuai dengan *Part of Speech*. Dari semua tag tersebut hanya beberapa tag yang akan digunakan sebagai fitur seperti yang didefinisikan pada kunci *mask*.

```
"logr-repl-rnn": {
        "type": "tfidf",
        "solver": "logr",
        "normalize_word": true,
            "kind": "compound-classifier",
        "normalize_word": true,
        "use_punctuation": true,
        "tagger_model_id":      "7064187d-3e0e-429c-b7d1-
a28608f3d9c0",
        "mask": ["NEG", "NOUN", "PRONOUN", "WP",
"CC"],
        "replace": {
           "positive": ["a+", "acungan jempol",…,  "yihaa"],
           "negative": ["abnormal", "absurd",…, "Yahudi"]
        },
        "params": {
           "penalty": "l2",
           "solver_opt": "newton-cg",
           "multi_class": "multinomial",
           "ngram_range": [1, 3],
           "min_word": 2,
           "max_features": 10000,
           "class_weight": "balanced"
        }
     }
```

Kode 4.5 Parameter skenario 4

Kode 4.5 menjelaskan bahwa pelatihan akan dilakukan menggunakan algoritma *logistic regression*. Kode disini merupakan gabungan antara Kode 4.4 dan Kode 4.3, fitur yang digunakan adalah *part of speech* dan *replace negative/positive words.*

```
"fasttext": {
        "type": "fasttext",
        "solver": "supervised",
        "name": "test",
        "params": {
           “dim”: 100,
           "ngram_range": 3,
           "epoch": 100
        },
        "use_punctuation": true,
        "normalize_word": true
    }
```

Kode 4.6 Parameter skenario 5

Kode 4.6 menjelaskan bahwa pelatihan model akan menggunakan algoritma fasttext. Karena data telah memiliki label maka *solver* yang digunakan adalah *supervised*. Pada pelatihan ini fitur yang digunakan ialah fitur standar dengan *epoch* 100 dan *trigram* yang merupakan parameter bawaan dari algoritma *fasttext.*

```
"fasttext-repl": {
        "type": "fasttext",
        "solver": "supervised",
        "name": "test",
        "replace": {
           "positive": ["a+", "acungan jempol”,…,  "yihaa"],
           "negative": ["abnormal", "absurd”,…, "Yahudi"]
        },
        "params": {
           “dim”: 100,
           "ngram_range": 3,
           "epoch": 100
```

```
        },
        "use_punctuation": true,
        "normalize_word": true
    }
```

Kode 4.7 Parameter skenario 6

Kode 4.7 menjelaskan pelatihan data menggunakan algoritma *fasttext* sama seperti pada Kode 4.6 namun dengan tambahan menggunakan fitur *replace positive/negative words* yang dapat dilihat pada kunci *replace*.

```
"fasttext-rnn": {
        "type": "fasttext",
        "solver": "supervised",
        "name": "test",
        "kind": "compound-classifier",
        "tagger_model_id": "7064187d-3e0e-429c-b7d1-
a28608f3d9c0",
        "mask": ["NEG", "NOUN", "PRONOUN", "WP",
"CC"],
        "params": {
          “dim”: 100,
          "ngram_range": 3,
          "epoch": 100
        },
        "use_punctuation": true,
        "normalize_word": true}
```

Kode 4.8 Parameter skenario 7

Kode 4.8 menjelaskan bahwa data akan dilatih menggunakan algoritma *fasttext*. Semua parameter yang dimasukkan sama dengan pada Kode 4.6 dengan tambahan fitur *tagging*. Sebuah *tagger* yang akan digunakan merupakan *tagger* milik Kata.ai yang dapat akan memberikan label pada kalimat sesuai dengan *Part of Speech*. Dari semua tag tersebut hanya beberapa tag yang akan digunakan sebagai fitur seperti yang didefinisikan pada kunci *mask*.

```
"fasttext-repl-rnn": {
        "type": "fasttext",
```

```
        "solver": "supervised",
        "name": "test",
            "kind": "compound-classifier",
        "tagger_model_id":      "7064187d-3e0e-429c-b7d1-
a28608f3d9c0",
        "mask": ["NEG", "NOUN", "PRONOUN", "WP",
"CC"],
        "replace": {
           "positive": ["a+", "acungan jempol",…, "yihaa"],
           "negative": ["abnormal", "absurd",…, "Yahudi"]
        },
        "params": {
           "dim": 100,
           "ngram_range": 3,
           "epoch": 100
        },
        "use_punctuation": true,
        "normalize_word": true
    }
```

Kode 4.9 Parameter skenario 8

Kode 4.9 akan menggabungkan fitur pada Kode 4.7 dengan fitur pada Kode 4.8. Kedua fitur tersebut adalah *pos tag* dan *positive/negative words*. Pengaturan tersebut akan digunakan pada dua buah proyek yaitu pembuatan model klasifikasi datar dan model klasifikasi hirarki seperti pada Kode 4.10. Masing-masing dari folder tersebut akan memiliki dataset yang berbeda untuk dilatih. Pada proyek *noncompound* data akan terbagi menjadi satu kelas yaitu topik, sedangkan pada *compound* data terbagi menjadi beberapa kelas yaitu topik, positif, negatif dan netral.

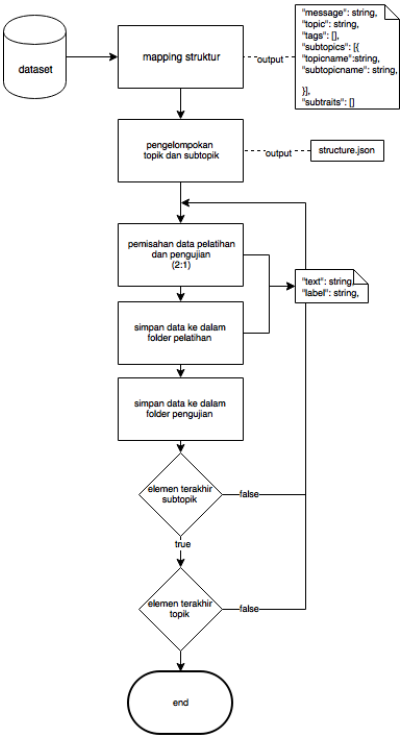
```
"projects": {
    "sentiment_analysis_compound": {
       "data": "",
       "test": ""
    },
    "sentiment_analysis_noncompound": {
       "data": "",
```

```
        "test": ""
    }
}
```

Kode 4.10 Pengaturan model datar dan hirarki

4.3.1 *Pre-processing data*

Pada Gambar 4.14 menjelaskan alur dari *pre-processing* sebelum data dilatih menggunakan *verstand service.*



Gambar 4.14 Alur preprocess data

Mapping struktur data dilakukan dengan menggunakan Kode 4.1. Sehingga keluaran yang dihasilkan sesuai dengan alur pada Gambar 4.14.

```
with open('messages.json') as json_data:
    data = json.load(json_data)
```

```
    output = []
    for i in data:
        sentiment = "neutral"
        try:
            for topic in i['subtopics']:
                if topic['topicname'] == "Sentiment":
                    sentiment = topic['subtopicname']
        except:
            print "%s -- dont have a subtopics" % (i['message'])
        topicname = "Sentiment"
        positiveArray = ["pleasant", "compliment"]
        neutralArray = ["neutral", "spam", "closing"]
        negativeArray = ["upset", "sarcasm", "mocking",
"denial"]
        if sentiment in negativeArray:
            topicname = "negative"
        elif sentiment in neutralArray:
            topicname = "neutral"
        elif sentiment in positiveArray:
            topicname = "positive"

        item = {
            "message": i['message'],
            "topic": i['topic'],
            "tags": [],
            "subtopics": [{
                "topicname": topicname,
                "subtopicname": sentiment
            }],
            "subtraits": []
        }
        output.append(item)

    with open('outfile.json', 'wb') as fp:
        json.dump(output, fp)
```

Kode 4.11 Membuat struktur dataset model hirarki

Pada Kode 4.11 data yang telah diunduh akan dirubah ke dalam struktur yang telah ditentukan untuk itu dibuat *script* menggunakan bahasa pemrograman *Python* untuk membersihkan data, karena pada dataset sebelumnya terdapat beberapa kelas yang tidak memiliki sentiment maka teks tersebut dianggap sebagai *neutral*. Kode tersebut akan membaca data dari file json yang kemudian tiap barisnya akan dilakukan *looping,* tiap putaran data akan ubah menjadi object dan dipindahkan ke dalam array baru. Setelah itu data akan ditulis kembali menjadi *outfile.json* dan dipindahkan menuju folder sentiment_analysis_compound. Perbedaan antara *script* untuk struktur model non-hirarki adalah pada bagian *topicname* seperti Kode 4.12.

```
item = {
            "message": i['message'],
            "topic": i['topic'],
            "tags": [],
            "subtopics": [{
               "topicname": sentiment
            }],
            "subtraits": []
         }
```

Kode 4.12 Struktur item dataset model non-hirarki

4.3.2 Pemisahan data

Setelah dataset siap pada masing-masing folder maka data perlu dipisahkan sesuai pembagian data pelatihan dan data pengujian. Kedua data tersebut juga akan pisah sesuai dengan klasifikasinya masing-masing. Sehingga pada model non-hirarki memiliki data pelatihan sebanyak satu set, sedangkan model hirarki akan melatih datanya tiap topik yang dimiliki yaitu positif, negatif dan netral.

```
let annotatorData;
   try {
      annotatorData = require(filename);
   } catch (e) {
      return null;
```

```
    }
    let data = {};
    let topics = {};
    annotatorData.forEach((item, i) => {
        if (item.topic && !data[item.topic]) {
            data[item.topic] = {}
            topics[item.topic] = []
        }
        if (item.subtopics.length > 0) {
            item.subtopics.forEach((sub_item, i) => {
                if (!data[sub_item.topicname])
                    data[sub_item.topicname] = {};
                if (!topics[sub_item.topicname])
                    topics[sub_item.topicname] = [];
                if
(!data[sub_item.topicname][sub_item.subtopicname]) {

data[sub_item.topicname][sub_item.subtopicname]          =
[item.message];
                } else {

data[sub_item.topicname][sub_item.subtopicname].push(ite
m.message);
                }
                if
(!(topics[sub_item.topicname].indexOf(sub_item.subtopicn
ame) > -1)) {
                    if (sub_item.subtopicname)

topics[sub_item.topicname].push(sub_item.subtopicname);
                }
            });
        }
    });
    return {
        data,
        topics
    }
```

Kode 4.13 Pemisahan topik dan subtopik

Kode 4.13 membaca data dari file dataset yang telah dibuat sebelumnya, apabila data memiliki sub-topik seperti halnya pada dataset model hirarki maka *script* tersebut akan mengembalikan relasi antara topik dan subtopik dalam bentuk json selain itu *script* akan mengembalikan tiap sesuai dengan topik atau subtopiknya.

```
fs.writeFileSync(`./data/${project}/structure.json`,
JSON.stringify(topics, null, 4));
```

Kode 4.14 Membuat relasi kelas

Setelah Kode 4.13 memberikan relasi antar kelas maka untuk memudahkan membaca struktur dari kelas, pada tiap proyek akan dibuat relasi kelas dengan menggunakan salah satu fungsi javascript untuk membuat dan menuliskan ke dalam file pada Kode 4.14.

```
for (let topic in data) {
      trainData[topic] = {};
      testData[topic] = {};
      for (let subtopic in data[topic]) {
         if (subtopic.length > 0) {
            const {
               train,
               test
            } = splitData(data[topic][subtopic], n);
            trainData[topic][subtopic] = train;
            testData[topic][subtopic] = test;
         }
      }
   }
```

Kode 4.15 Looping tiap data dan subtopik

Setelah data dipisah sesuai dengan topik atau subtopiknya pada Kode 4.15. Maka tiap topik dan subtopik akan dipisah untuk menjadi data pelatihan dan data pengujian.

```
const n = Math.floor(array.length * testRange);
```

```
   const shuffled = array.sort(() => .5 - Math.random()); //
shuffle
   let train = shuffled.slice(0, n),
      test = shuffled.slice(n, array.length);

   return {
      train,
      test};
```

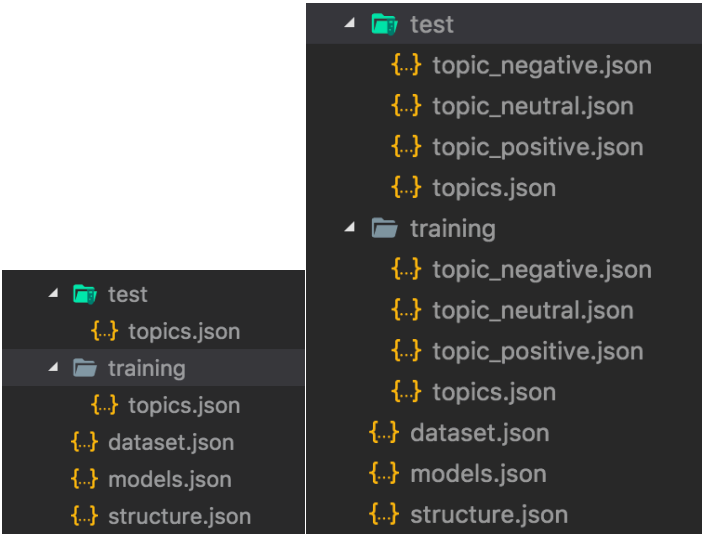
Kode 4.16 Pemisahan data pelatihan dan pengujian

Kode 4.16 akan memisah data dengan perbandingan 2/3 dari total data. Data akan diurutkan secara random sehingga data yang digunakan untuk pelatihan dan pengujian senantiasa acak. Data pelatihan akan mengambil 2/3 dari data yang telah diacak menggunakan fungsi *slice* dimulai dari index 0 sedangkan data pengujian akan mengambil potongan 1/3 pada data dimulai dari index n (2/3 * panjangdata). Setelah itu data pelatihan dan pengujian akan dimasukkan kedalam dua buah folder yang berbeda yaitu *train* dan *test* pada tiap proyek.

```
let topicData = Object.keys(data).reduce((array, label) => {
      if (data[label].length) {
         return array.concat(data[label].map(text => ({
            text,
            label
         })));
      } else {
         return
array.concat(Object.keys(data[label]).reduce((array,      sub)
=> {
            return  array.concat(data[label][sub].map(text  =>
({
               text,
               label
            })));
         }, []));
      }
   }, []);
```

Kode 4.17 Mapping data tiap topik dan subtopik

Kode 4.17 akan merubah struktur data tiap subtopik dan topik menjadi *text* dan *label*. Setelah semua data selesai *mapping* maka akan dihasilkan sebuah file baru yang merupakan data pelatihan dan pengujian tiap kelas. Seperti gambar dibawah ini.



Gambar 4.15 Hasil akhir pemisahan data untuk model non-hirarki (kiri) dan hirarki (kanan)

4.3.3 Pelatihan model

Tahapan selanjutnya adalah proses pelatihan dataset untuk masing-masing skenario yang telah dibuat sehingga menghasilkan model kaslifikasi sentimen. Untuk mengakses *verstand service* maka beberapa library perlu digunakan. Pengaturan yang digunakan sesuai dengan perancangan yang telah dibuat sebelumnya.

```
const fs = require("fs");
const config = require("./config");
const proxy = require("@yesboss/merapi-console-proxy");
const sleep = require("sleep");
const verstand = proxy(config.verstand);
const worker = config["worker"] || false
```

Kode 4.18 library untuk melatih menggunakan verstand

Setiap proyek akan dilatih sesuai dengan pengaturan yang telah dibuat sebelumnya.

```
for (let project in config.projects) {
   let topics = require(`./data/${project}/structure`);
   let models = {};
   try {
      models = require(`./data/${project}/models`);
   } catch (e) {}
   for (let i in config.classifiers) {
      let topicsModelName = i + "_topics";
      models[topicsModelName] =
train(models[topicsModelName], config.classifiers[i],
createLabels(Object.keys(topics)),
require(`./data/${project}/training/topics`));

      for (let topic in topics) {
         if (topics[topic].length && topics[topic].length > 0) {
            let modelName = i + "_" + topic;
            models[modelName] =
train(models[modelName], config.classifiers[i],
createLabels(topics[topic]),
require(`./data/${project}/training/topic_${topic}`));
         }
      }
   }

   fs.writeFileSync(`./data/${project}/models.json`,
JSON.stringify(models, null, 4));
}
```

Kode 4.19 Script pelatihan model

Kode 4.19 akan melatih data dari tiap proyek yang telah dibuat sebelumnya. Apabila file models.json telah dibuat sebelumnya maka pelatihan akan menggunakan models.json tersebut untuk memperbarui model. Untuk setiap topic akan subtopik akan dilakukan pelatihan secara berulang sampai dengan semua topik

dan subtopik selesai dilatih menggunakan *verstand service.* Setelah itu hasil dari pelatihan tersebut akan disimpan ke dalam file models.json.

```
let model;
    if (id && verstand.findModel({
            id
        })) {
        model = verstand.updateModel(id, {
            problem_type: options.type,
            solver_algo: options.solver,
            vars: options.params,
            start_end: options.start_end || false,
            normalize_word: options.normalize_word || false,
            replace: options.replace || {},
            use_punctuation: options.use_punctuation || false
        });
        console.log(model.id, "updated");
    } else {
        model = verstand.createModel({
            id,
            model_type: options.kind || 'classifier',
            problem_type: options.type,
            solver_algo: options.solver,
            vars: options.params,
            name: options.name,
            mask: options.mask,
            replace: options.replace || {},
            normalize_word: options.normalize_word || false,
            use_punctuation: options.use_punctuation || false,
            tagger_model_id:      options.tagger_model_id      ||
"854eb669-2463-4ab8-9dfe-fe8d15bf707d"
        });
        console.log(model.id, "created model");
    }
```

Kode 4.20 Pengecekan dan mapping parameter verstand

Kode 4.20 akan melihat apakah modelId yang didapat dari models.json sebelumnya sudah terdaftar pada *verstand service* melalui fungsi findModel(modelId). Apabila model ditemukan maka *mapping* untuk mengisi parameter verstand service akan dilakukan. *Mapping* diambil dari pengaturan pada config.js yang telah dibuat sebelumnya dan memperbarui pengaturan melalui verstand.updateModel() sedangkan ketika id tidak ditemukan maka verstand.createModel() akan dijalankan untuk membuat model baru. Pada tahap ini model yang dibuat hanya berupa pengaturan atau konfigurasi yang disimpan ke dalam basis data verstand. Setelah itu pada Kode 4.21 akan dibuat daftar label pada model tersebut dari tiap topik yang ada pada file structure.json. Hasil dari *mapping* label tersebut akan disimpan ke dalam basis data *verstand* menggunakan Kode 4.22 dimana *labels* merupakan hasil dari Kode 4.21.

```
return array.reduce((obj, key, idx) => {
      obj[key] = idx;
      return obj;
   }, {});
```

Kode 4.21 Mapping label

```
verstand.createEntity(model.id, {
      labels
   });
```

Kode 4.22 Membuat entitas pada verstand

Untuk memasukkan dataset ke dalam basis data verstand maka kita perlu memanggil verstand.addTrainingData(). Kode 4.23 akan memilah data dengan kelipatan 200. Setiap 200 data akan dimasukkan untuk ke dalam basis data verstand sebagai bahan untuk pelatihan model.

```
for (let i = 0; i < numIteration; i++) {

      verstand.addTrainingData(model.id, data.slice(i * 200,
Math.min((i + 1) * 200, data.length)));

   }
```

Kode 4.23 Memasukkan dataset ke database verstand

Setelah pengaturan model, entitas dan data pelatihan telah masuk ke dalam *verstand* maka data dapat dilatih menggunakan pemanggilan API seperti Kode 4.24 dimana model.id merupakan id dari model yang terdapat pada pengaturan model atau models.json.

```
verstand.syncTrainModel(model.id)
```

Kode 4.24 Mulai melatih model

Setelah model selesai dibuat maka hasilnya melalui Kode 4.25 akan dituliskan dalam models.json, baik itu pembaruan model ataupun pembuatan model.

```
fs.writeFileSync(`./data/${project}/models.json`,
JSON.stringify(models, null, 4));
```

Kode 4.25 Daftar model dan model id

4.3.4 Validasi model klasifikasi

Validasi model klasifikasi sentimen akan menggunakan data yang digunakan untuk melatih dan juga data pengujian. Sebuah *script* dengan bahasa pemrograman JavaScript akan dibuat untuk melakukan validasi dengan menghitung akurasi dari model klasifikasi sentimen tersebut.

```
for (let project in config.projects) {
   console.log(`[${project}]`);
   let topics = require(`./data/${project}/structure`);
   let models = require(`./data/${project}/models`);
   results[project] = {};
   for (let classifier in config.classifiers) {
      let result = {};
      console.log(classifier + ":");
      result = results[project][classifier] = {
         topics: runTestProject(models[classifier + "_topics"],
project)
      };
      printResult("topics", result.topics);
      for (let topic in topics) {
```

```
            if (topics[topic].length > 0) {
                result[topic] =
runTestProjectCompound(models[classifier + "_topics"],
models[classifier + "_" + topic], project, topic);
                printResultCompound(topic, result[topic]);
            }
        }
    }
}
```

Kode 4.26 Kode utama untuk validasi model klasifikasi

Validasi dengan Kode 4.26 akan dilakukan pada tiap proyek yang ada dengan membaca file struktur dan models. Setelah itu tiap klasifikasi pada file config akan lakukan uji coba. Setiap topik maupun subtopik pada proyek akan dilakukan uji coba secara bergantian.

```
let file = topic ? "topic_" + topic : "topics";
    let training =
require(`./data/${project}/training/${file}.json`);
    let test = null;
    try {
        test = require(`./data/${project}/test/${file}.json`);
    } catch (e) {}
    let trainingResult = runTestEnd(modelId, training);
    let testResult = test ? runTestEnd(modelId, test) : {
        count: {
            total_data: 0
        },
        failed: [],
        success: []
    };
```

Kode 4.27 Validasi menggunakan data pelatihan dan pengujian

Kode 4.27 akan mengambil data pelatihan dan pengujian yang telah dipisah sebelumnya untuk melakukan validasi terhadap klasifikasi model non-hirarki.

```
let count = {
        total_data: test.length
```

```
    }
    let success = [];
    let failed = [];
    let resultData = verstand.batchPredict(test.map(item =>
item.text), id).map((result, idx) => {
        let data = {
            expected: test[idx].label,
            result,
            text: test[idx].text
        }
        try {
            count[data.expected]["total_data"]                    =
count[data.expected]["total_data"] + 1;
            if                (data.result.toUpperCase()                !=
data.expected.toUpperCase()) {
                failed.push(data);
                count[data.expected]["failed"]                    =
count[data.expected]["failed"] + 1;
            } else {
                success.push(data);
                count[data.expected]["success"]                   =
count[data.expected]["success"] + 1;
            }
        } catch (error) {
            count[data.expected] = {
                total_data: 1,
                failed: 0,
                success: 0
            };
            if                (data.result.toUpperCase()                ==
data.expected.toUpperCase()) {
                failed.push(data);
                count[data.expected]["failed"]                    =
count[data.expected]["failed"] + 1;
            } else {
                success.push(data);
                count[data.expected]["success"]                   =
count[data.expected]["success"] + 1;
```

```
        }
    }
```

Kode 4.28 Pengujian dengan memanggil fungsi batchPredict pada verstand

Pada tiap validasi baik menggunakan data pelatihan atau pengujian, Kode 4.28 akan memanggil fungsi batchPredict pada verstand dengan memasukkan sumber data pengujian. batchPredict merupakan fungsi verstand untuk melakukan prediksi dengan data yang banyak. Hasil dari klasifikasi menggunakan batchPredict akan di-*mapping* ke dalam sebuah array dan melakukan pengecekan apakah hasil dari klasifikasi sesuai dengan label pada sumber data pengujian. Hasil dari pengecekan tersebut akan dihitung sesuai kelasnya masing-masing dan data pengujian yang berhasil akan dipisahkan dengan data yang gagal. Untuk melakukan validasi model klasifiaski hirarki maka fungsi batchPredict akan dilakukan sebanyak dua kali atau sebanyak level hirarki.

```
let compoundTraining = runTestCompound(topic, topicModelId, training);
  let compoundTest = test ? runTestCompound(topic, topicModelId, test) : {
      count: {
         total_data: 0
      },
      success: [],
      failed: []
  };
  };
```

Kode 4.29 Validasi model klasifikasi hirarki

Untuk melakukan validasi model hirarki hampir sama dengan Kode 4.28 perbedaannya adalah sebelum melakukan validasi terhadap akhir maka data akan diuji menggunakan model klasifikasi topik seperti tampak pada Kode 4.29. Setelah itu data akan yang berhasil akan diteruskan menuju model klasifikasi selanjutnya dan menentukan akurasi akhir dari total semua data yang diuji dari model klasifikasi topik.

4.3.5 Analisa model klasifikasi sentimen

Berikut adalah parameter validasi dan hasil dari percobaan dengan skenario sesuai dengan Table 4.4. Parameter default dari percobaan menggunakan algoritma logistic regression dijelaskan pada Table 4.14 sedangkan parameter default menggunakan algoritma *fasttext* dijelaskan pada Table 4.15, setelah itu digunakan skenario Table 4.4 untuk memperbanyak variasi pengujian, setelah itu model juga dibagi menjadi 2 jenis yaitu model datar dan model hirarki.

Table 4.14 Parameter awal logistic regression

Parameter	Isian
type	tfidf
solver	logr
normalisasi	"true"
punctuation	“true”
penalty	l2
solver_opt	newton-cg
multi_class	multinomial
ngram_range	[1,3]
min_word	2
max_features	1000
class_weight	balanced

Sedangkan pada algoritma *fasttext* memiliki beberapa parameter yang berbeda yaitu seperti dijelaskan pada Table 4.15.

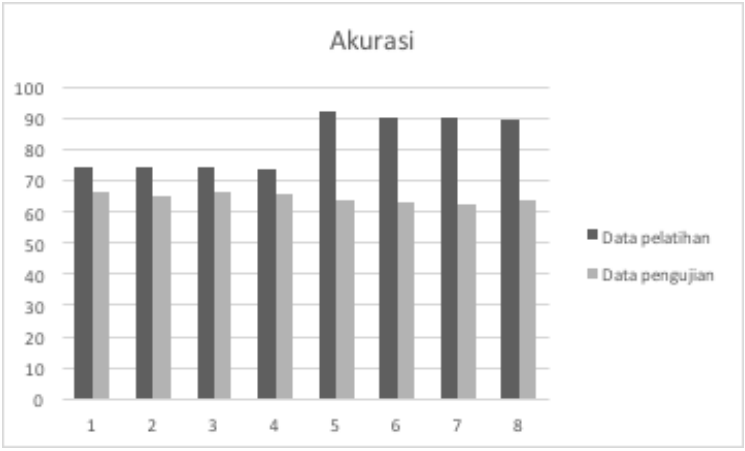
Table 4.15 Parameter awal fasttext

Parameter	Isian
type	fasttext
solver	supervised
normalisasi	"true"

punctuation	"true"
dim	512
ngram_range	3
epoch	100

4.3.5.1 Hasil validasi model datar

Pada bagian ini akan dipaparkan hasil perhitungan akurasi pada model datar atau non-hirarki menggunakan dua buah data yaitu data pelatihan dan data pengujian.



Grafik 4.2 Hasil akurasi model datar

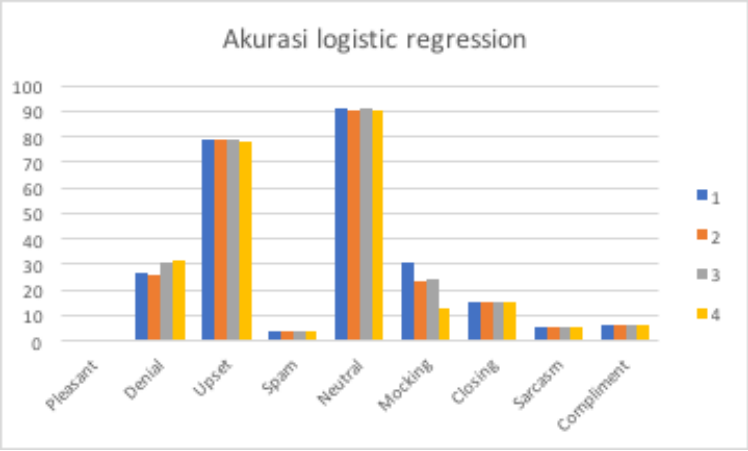
Pada Table 4.16 dijelaskan secara spesifik hasil dari pengujian akurasi terhadap model klasifikasi sentimen.

Table 4.16 Hasil akurasi model datar dengan akurasi total

Skenario	Akurasi	
	Data pelatihan	**Data pengujian**
1	74.46	66.14
2	74.00	65.07
3	74.32	66.29

4	73.71	65.62
5	92.03	63.66
6	90.39	62.99
7	90.08	62.53
8	89.19	63.51

Pada Table 4.16 diketahui bahwa pada skenario dengan algoritma *logistic regression* memiliki akurasi menggunakan data pelatihan yang lebih kecil dibanding dengan algoritma *fasttext,* namun hasil tersebut sedikit lebih besar pada data pengujian. Rata-rata akurasi memiliki nilai yang tinggi, untuk melakukan validasi terhadap akurasi perkelas maka Grafik 4.3 memberikan visualisasi terhadap prediksi per kelas sentimen pada algoritma *logistic regression.*



Grafik 4.3 Hasil akurasi per kelas model datar dan logistic regression

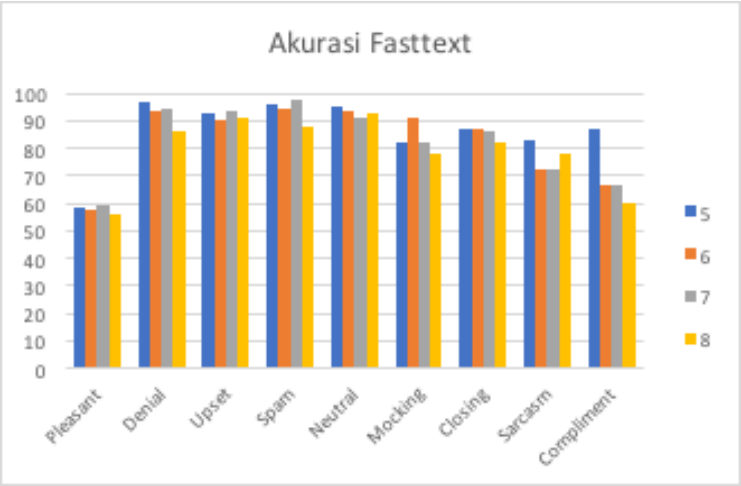
Table 4.17 memperlihatkan data akurasi secara akurat yang mendukung visualisasi Grafik 4.3. Pada tabel ini juga ditambahkan hasil validasi menggunakan data pengujian. Kelas *pleasant* memiliki akurasi 0% sedangkan beberapa kelas

lainnya seperti *spam, sarcasm, compliment* memiliki tingkat akurasi yang kecil.

Table 4.17 Presisi dengan model datar dan logistic regression

Sentimen	Skenario							
	1		2		3		4	
	D1	D2	D1	D2	D1	D2	D1	D2
Pleasant	0.28	0.56	0.28	0.56	0.28	0.56	0.28	0.56
Denial	26.54	26.83	25.93	26.83	30.86	32.93	31.48	34.15
Upset	79.15	69.17	78.82	67.22	78.87	68.43	78.41	67.69
Spam	4.13	1.83	4.13	1.83	4.13	1.83	4.13	1.83
Neutral	90.89	81.89	90.65	81.24	90.95	82.37	90.68	81.89
Mocking	30.81	16.13	23.24	12.90	24.32	16.13	12.43	8.60
Closing	15.29	20.93	15.29	20.93	15.29	20.93	15.29	20.93
Sarcasm	5.56	10.00	5.56	10	5.56	10.00	5.56	10.00
Compli-ment	6.67	12.5	6.67	12.50	6.67	12.50	6.67	12.50

Grafik 4.4 memberikan gambaran bahwa menggunakan algoritma *fasttext* hasil yang diberikan lebih merata akurasinya. Bila dibandingkan *logistic regression* akurasi menggunakan algoritma *fasttext* memiliki tingkat akurasi yang jauh lebih tinggi.



Grafik 4.4 Hasil akurasi per kelas model datar dan fasttext

Pada Table 4.21 akan dilanjutkan validasi menggunakan data pengujian. Seperti yang dapat dilihat bahwa akurasi menggunakan data pengujian jauh lebih kecil dibanding data pelatihan.

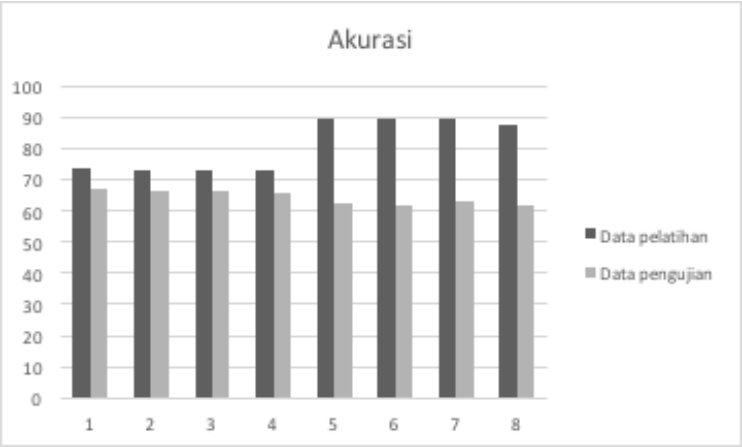
Table 4.18 Presisi dengan model datar dan fasttext

Sentimen	Skenario							
	5		6		7		8	
	D1	D2	D1	D2	D1	D2	D1	D2
Pleasant	58.82	4.47	57.42	5.03	59.66	3.35	55.74	3.91
Denial	96.91	60.98	93.83	62.20	94.44	56.10	86.42	51.22
Upset	92.96	65.19	90.22	65.28	93.61	69.44	90.73	67.87
Spam	96.33	17.43	94.50	16.51	97.71	64.22	88.07	13.76
Neutral	95.20	74.70	93.82	72.90	91.10	66.79	92.99	73.44

Mocking	82.16	30.11	91.35	36.56	82.16	34.41	77.84	32.26
Closing	87.06	60.47	87.06	60.47	85.88	60.47	82.35	55.81
Sarcasm	83.33	10.00	72.22	10.00	72.22	10.00	77.78	10.00
Compliment	86.67	12.50	66.67	12.5	66.67	12.50	60.00	12.50

4.3.5.2 Hasil pengujian model hirarki

Pada bagian ini akan dipaparkan hasil perhitungan akurasi pada model datar atau non-hirarki menggunakan dua buah data yaitu data pelatihan dan data pengujian. Grafik 4.5 memaparkan hasil akurasi dari model hirarki, tren yang didapat pada tiap skenario tidak begitu berbeda dengan Grafik 4.2.



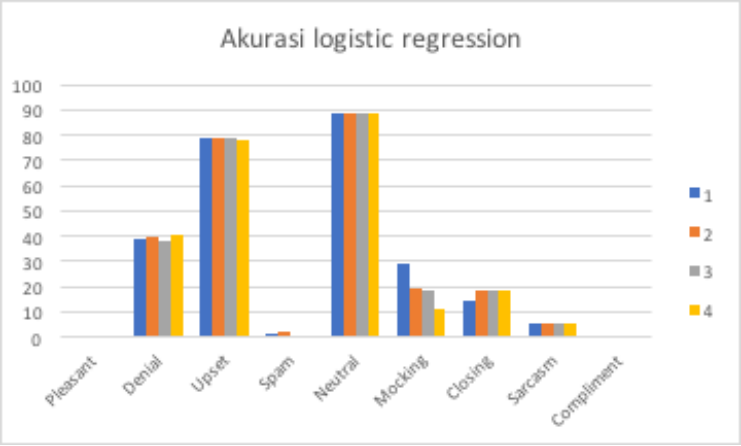
Grafik 4.5 Hasil pengujian model hirarki

Detail dari hasil pengukuran akurasi ditunjukan pada Table 4.19. Akurasi tertinggi dengan data pelatihan dimiliki oleh skenario 6 namun dengan data pengujian skenario 1 memiliki akurasi tertinggi.

Table 4.19 Hasil pengujian model hirarki akurasi total

Skenario	Akurasi	
	Data pelatihan	Data pengujian
1	73.57	66.96
2	73.07	66.44
3	73	66.16
4	72.65	65.43
5	89.43	62.59
6	89.67	61.98
7	89.34	63.23
8	87.63	61.67

Dari hasil Table 4.19 data akurasi dihitung dari total banyaknya data yang divalidasi, sehingga pada Grafik 4.6 menunjukan hasil klasifikasi dengan model hirarki dan algoritma *logistic regression.*



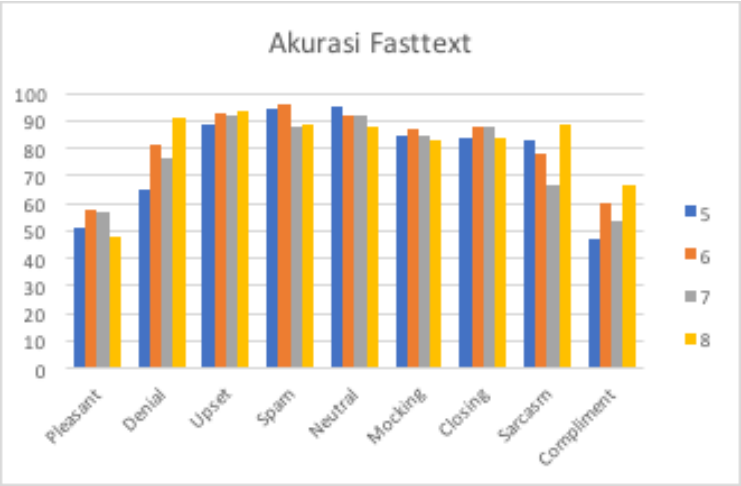
Grafik 4.6 Hasil pengujian akurasi per kelas model hirarki dan logistic regression

Table 4.21 menunjukan detail dari hasil pengukuran akurasi pada Grafik 4.6 di mana terdapat validasi dengan menghitung akurasi dengan data pengujian sehingga didapatkan hasil sebagai berikut.

Table 4.20 Pengujian presisi dengan model hirarki dan logistic regression

Sentimen	Skenario							
	1		2		3		4	
	D1	D2	D1	D2	D1	D2	D1	D2
Pleasant	0.00	.00	0.00	0.00	0.00	0.00	0.00	0.00
Denial	38.89	29.27	40.12	28.05	38.27	31.71	40.74	32.93
Upset	78.92	68.70	79.10	68.89	78.87	67.13	78.04	67.41
Spam	1.38	0.92	1.83	0.92	0.46	0.92	0.46	0.92
Neutral	89.06	83.33	88.31	82.73	88.55	82.73	88.67	81.71
Mocking	29.19	24.73	19.46	15.05	18.38	21.51	11.35	10.75
Closing	14.12	23.26	18.82	25.58	18.82	27.91	18.82	25.58
Sarcasm	5.56	10.00	5.56	10.00	5.56	10.00	5.56	10.00
Complim-ent	0.00	.00	0.00	0.00	0.00	0.00	0.00	0.00

Visualisasi hasil pengukuran akurasi menggunakan algoritma fasttext dan model klasifikasi hirarki ditampilkan pada Grafik 4.7.



Grafik 4.7 Hasil pengujian akurasi per kelas model hirarki dan fasttext

Detail nilai hasil pengukuran pada Grafik 4.7 diperjelas pada Table 4.21 dari hasil tersebut dapat dilihat bahwa akurasi memiliki nilai yang seimbang.

Table 4.21 Pengujian presisi dengan model hirarki dan fasttext

Sentimen	Skenario							
	5		6		7		8	
	D1	D2	D1	D2	D1	D2	D1	D2
Pleasant	51.54	1.68	57.70	2.79	56.58	2.79	48.18	1.68
Denial	64.81	29.27	81.48	42.68	76.54	46.34	91.36	50.00
Upset	88.69	59.54	92.40	65.19	92.26	66.11	93.93	70.46
Spam	94.50	15.60	96.33	74.31	87.61	9.17	88.99	64.22
Neutral	95.47	78.06	91.67	68.17	92.27	73.92	87.95	64.99
Mocking	84.86	38.71	87.03	37.63	84.86	44.09	83.24	35.48
Closing	83.53	53.49	88.24	67.44	88.24	60.47	83.53	55.81

Sarcasm	83.33	0.00	77.78	10.00	66.67	10.00	88.89	10.00
Compliment	46.67	0.00	60.00	12.50	53.33	12.50	66.67	12.50

4.4 Analisa model klasifikasi sentimen

Dari hasil diatas dapat dilihat performa dari algoritma *logistic regression* dan *fasttext*. Di mana rata-rata total akurasi didapat dengan menjumlahkan semua akurasi pada data pengujian dan data pelatihan dan di bagi dari total data. Perbandingan hasil akurasi dari tiap algoritma dapat dilihat pada Table 4.22.

Table 4.22 Perbandingan rata-rata tiap algoritma

Algoritma	**Rata-rata (%)**	
	Akurasi	**Presisi**
Logistic Regression	69.8	27.19
Fasttext	76.25	59.46

Ketika membandingkan presisi kelas pada algoritma *logistic regression* dan *fasttext* pada Table 4.23 terlihat bahwa algoritma *fasttext* memiliki mayoritas presisi lebih tinggi dibandingkan dengan algoritma *logistic regression.* Dimana hanya terdapat satu kelas dengan akurasi tertinggi pada *logistic regression* yaitu neutral.

Table 4.23 Perbanding presisi tiap kelas

Sentimen	**Rata-rata presisi(%)**	
	Logistic Regression	**Fasttext**
Pleasant	0.2	29.5
Denial	32.2	67.8
Upset	73.4	79.0
Spam	2.0	63.7

Neutral	86.0	82.1
Mocking	18.4	60.2
Closing	19.9	72.5
Sarcasm	7.8	43.3
Compliment	4.8	37.1

Table 4.22 terlihat bahwa alogirtma *fasttext* secara keseluruhan memiliki rata-rata akurasi yang lebih tinggi sekitar 6.5% dibandingkan dengan *logistic regression*. Perhitungan ini menggabungkan akurasi dari model datar dan non-hirarki. Sedangkan pada Table 4.24 meunujukan akurasi pada model datar dan hirarki. Akurasi dihitung berdasarkan total data yang gagal dibandingkan jumlah keseluruhan data.

Table 4.24 Hasil akurasi model datar dan hirarki

	Total akurasi model(%)	
	Model datar	**Model hirarki**
Data pelatihan	81.04	82.27
Data pengujian	64.48	64.3

Pada Table 4.24 hasil akurasi yang diberikan tidak memiliki perbedaan yang signifikan, namun akurasi tetap nilai tertinggi dengan menggunakan data pelatihan adalah model hirarki sedangkan pada data pengujian model datar unggul 0.18% dibanding model hirarki.

Table 4.25 Total Presisi tiap skenario

	Total presisi(%)	
Skenario	**Data pelatihan**	**Data pengujian**
1	28.69	26.67
2	27.99	25.85

3	28.10	27.09
4	27.14	25.97
5	81.82	34.01
6	82.21	40.12
7	80.10	39.04
8	80.26	38.11

Ketika menghitung presisi total pada Table 4.25, perbedaan yang didapatkan sangat signifikan antara skenario 1 – 4 dan 5 – 8. Rata-rata akurasi pada skenario dengan algoritma *logistic regression* adalah 27.16%. Pada algoritma *fasttext* rata-rata presisi yang didapatkan yaitu sebesar 59.45%. Namun algoritma *fasttext* validasi rasio validasi menggunakan data pelatihan dan data pengujian cukup besar. Nilai rata-rata pada data pelatihan sebesar 81.09% sedangkan pada data pengujian sebesar 37.82%. Presisi tertinggi dimiliki oleh skenario 6. Pada Table 4.26 presisi dipecah berdasarkan model datar dan hirarki.

Table 4.26 Hasil rata-rata presisi pada model datar dan hirarki

Skenario	**Rata-rata presisi(%)**		**Rata-rata(%)**
	Model datar	**Model hirarki**	
1	27.63	27.73	27.68
2	26.91	26.92	26.92
3	27.27	27.92	27.60
4	26.27	26.84	26.56
5	53.88	61.96	57.92
6	61.85	60.47	61.16
7	56.88	62.26	59.57
8	60.99	57.37	59.18

Table 4.26 menunjukan presisi pada skenario dengan algoritma *logistic regression* pada model datar atau hirarki tetap bernilai

kecil. Skenario 6 memliki nilai rata-rata presisi tertinggi dengan presisi model datar 61.85%. Skenario 7 dengan model hirarki memiliki nilai yang lebih tinggi dibandingkan skenario 6.

Table 4.27 Hasil presisi berdasarkan data pelatihan dan pengujian

	Model datar		Model hirarki	
	Data			
Skenario	Pelatihan	Pengujian	Pelatihan	Pengujian
1	28.57	26.69	28.81	26.65
2	28.13	25.69	27.84	26.00
3	27.66	26.88	28.55	27.30
4	27.07	25.48	27.21	26.46
5	77.04	30.71	86.60	37.32
6	81.40	42.30	83.01	37.94
7	77.60	36.15	82.61	41.92
8	81.42	40.57	79.10	35.64

Pada Table 4.27 mempertegas bahwa validasi menggunakan data pelatihan pada algoritma *fasttext* mendapatkan presisi yang tinggi. Presisi dengan data pengujian hasil yang didapatkan hampir ½ dari presisi dengan data pelatihan. Berdasarkan hasil diatas maka model dengan nilai akurasi dan presisi yang baik akan diimplementasikan ke dalam *web service.*

Table 4.28 Model yang dipilih untuk diimplementasikan

Skenario model	Tipe model	Akurasi Total(%)	Presisi Total(%)
6	Datar	76.69	61.85
7	Hirarki	76.26	62.26

5. *Halaman ini sengaja dikosongkan*

BAB V
IMPLEMENTASI

Pada bab ini, akan dijelaskan mengenai implementasi dari perancangan yang telah dilakukan sesuai dengan metode pengembangan yang dibuat. Bagian implementasi akan menjelaskan mengenai lingkungan implementasi, pembuatan fitur-fitur aplikasi dalam bentuk kode, serta pengujian aplikasi.

5.1 Lingkungan Implementasi

Pengembangan aplikasi ini menggunakan komputer dengan spesifikasi pada Tabel 6.1.

Tabel 6.1 Spesifikasi

Prosesor	2.7 GHz Intel Core i5
Memory	8 GB RAM
Sistem Operasi	*MacOS*
Graphics	*Intel Iris Graphics 6100 1536 MB*

Aplikasi dikembangkan dengan menggunakan beberapa teknologi seperti editor, database, server, bahasa pemrograman, dan *library* yang disajikan dalam Tabel 6.2.

Tabel 6.2 Daftar teknologi yang digunakan web service

Webserver	ExpressJS, NodeJS
Bahasa Pemrograman	Javascript
Database	MySQL
***Editor* (IDE)**	Visual Code Editor
Browser	Google Chrome 56

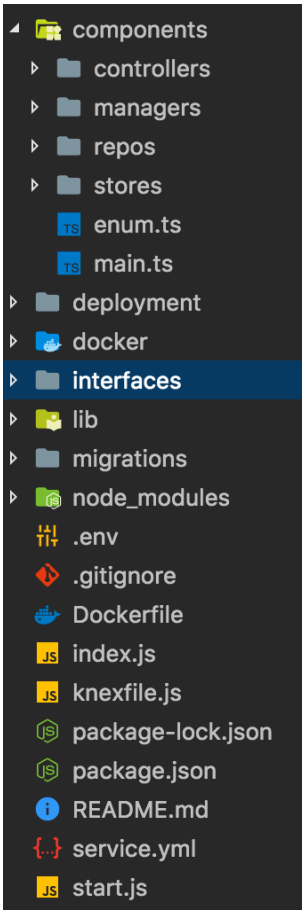
Library	• Merapi • Knex • MomentJS • MySQL • TypeScript • Yaml • Request • ExpressJS

5.2 Pembuatan *web service*

Pada tahap ini model telah berhasil dibuat dalam database *verstand service.* Namun untuk mengaksesnya memerlukan *merapi proxy*. Oleh karena itu, pada tahap ini akan dibuat sebuah *web service* sehingga model klasifikasi sentimen dapat diakses secara umum menggunakan protokol http.

5.2.1 Struktur direktori

Berikut adalah struktur direktori untuk pengembangan aplikasi pencarian dan penyaringan produk aplikasi web service untuk mengintegrasikan model klasifikasi sentimen pada *verstand service*. Aplikasi ini akan menggunakan merapi sebagai *framework* aplikasinya.



Gambar 6.1 Struktur direktori

File service.yml akan merupakan file pengaturan utama untuk menjalankan *web service*. Direktori *components* merupakan direktori utama untuk logika aplikasi sedangkan direktori interfaces berisikan kelas *interfaces* untuk membatasi fungsi atau *method* yang akan diimplementasikan pada direktori *components*. Direktori *migrations* berisikan file knex untuk membuat atau mengubah skema database. Pengaturan utama yang dibutuhkan pada service.yml didefinisikan pada Kode 6.1.

```
schema: kata.ai/schema/merapi/1.0

name: ${package.name}
version: ${package.version}

plugins:
  - express@yesboss

components:
  main: Main
  app:
    type: express
    config: app

  # data stores
  knex:
    type: component
    path: stores/knex
    load: true
  # managers
  ModelManager: managers/ModelManager

  #repos
  modelRepo: repos/ModelRepo
  compoundModelRepo: repos/CompoundModelRepo
  databaseRepo: repos/DatabaseRepo


  #controllers
  ModelController: controllers/ModelController

app:
  host: 0.0.0.0
  port: 8000
  routes:
    "POST                                    /compound/:id?":
ModelController.createOrUpdateCompound
```

```
    "GET /predict/compound/:id": ModelController.predictCompound
    "GET /predict/:id": ModelController.predict

# entry point
main: main
```

Kode 6.1 Pengaturan service.yml

5.2.2 Pembuatan *database*

Basis data yang akan digunakan adalah MySQL dengan bantuan sebuah library SQL *query builder*. Sesuai dengan pembuatan skema pada Bab 4.2.4.1 yaitu terdiri dari dua buah tabel model dan compound_model. Untuk membuat file skema baru menggunakan fungsi dari knex yaitu knex *migration:make* <*name*>.

```
exports.up = function (knex, Promise) {
  return knex.schema.createTable("model", function (table)
{
    table.uuid("id").primary().notNullable();
    table.string("name");
    table.text("class");
    table.string("providerProtocol");
    table.string("providerModelId");
    table.string("providerUrl");
    table.integer("providerPort");
    table.timestamps(true, true);
  });
};

exports.down = function (knex, Promise) {
  return knex.schema.dropTableIfExists("model");
};
```

Kode 6.2 Pembuatan tabel model dengan knex

```
exports.up = function (knex, Promise) {
  return knex.schema.createTable("compound_model",
function (table) {
    table.uuid("id").primary().notNullable();
```

```
        table.string("name");
        table.text("flow");
        table.timestamps(true, true);
    });
};

exports.down = function (knex, Promise) {
    return
knex.schema.dropTableIfExists("compound_model");
};
```

Kode 6.3 Pembuatan tabel compound_model menggunakan knex

Untuk menjalankan Kode 6.2 dan Kode 6.3 diperlukan pengaturan pada file konfigurasi service.yml mengenai pengaturan database. Setelah itu dapat diaktivasi dengan perintah *knex migrate:latest* pada terminal.

```
database:
    client: mysql
    connection:
        host: mysqldb
        user: kata
        password: kata-kata888!
        port: 3306
        database: ml_unity
        encrypt: true
    pool:
        min: '2'
        max: '10'
    migrations:
        tableName: migrations
```

Kode 6.4 Konfigurasi database

Setelah itu membuat repositori untuk mengakses tabel menggunakan query yang meliput fungsi *create*, *read*, *update* dan *delete* secara umum pada tiap tabel.

```
export interface IRepo<T> {


```

```
    get(id: string, entityName: string): Promise<Id<T>>;

    createOrUpdate(object: T | T[], entityName: string): Promise<Array<Id<T>>>;

}
```

Kode 6.5 Database repo interface

Kode 6.6 membuat file database repo sesuai dengan interface IRepo<T>.

```
export default class DatabaseRepo<T> extends Component implements IRepo<T> {

}
```

Kode 6.6 Buat database repo

Kode 6.7 merupakan implementasi fungsi get dengan parameter id, entityName. Di mana id merupakan nama kolom dari entityName yang merupakan nama tabel yang nantinya akan diakses. Hasil dari fungsi ini berupa Promise yang berarti menggunakan pemanggilan asynchronous. Hasil promise ini akan berisi sebuah objek dari baris pertama yang ditemukan menggunakan id. Apabila terjadi error dalam pemanggilan basis data menggunakan knex maka akan dikembalikan message error dan menampilkan error tersebut dalam log konsol.

```
async get(id: string, entityName: string): Promise<Id<T>> {
        let res;
        try {
            res = await this.knex(entityName).where('id', id).first();
        } catch (error) {
            console.log(error);
            return error;
        }
```

```
    return res;
  };
```

Kode 6.7 Implementasi fungsi get

Kode 6.8 memungkinkan untuk melakukan pembuatan data atau pembaruan data dalam basis data. Pertama akan dilakukan pengecekan apakah object berupa array atau sebuah object biasa. Setelah itu *query* ON DUPLICATE KEY UPDATE adalah *query* yang digunakan bila pada saat melakukan pembuatan data baru id sudah terdapat sebelumnya pada basis data. Maka secara otomatis akan dilakukan pembaruan terhadap data dengan id tersebut.

```
public async createOrUpdate(object: T | T[], entityName: string): Promise<Array<Id<T>>> {
    const item = { ...(object as any) };
    try {
      const firstData = Array.isArray(object) ? object[0] : object;
      await this.knex.raw(this.knex(entityName).insert(object).toQuery() + " ON DUPLICATE KEY UPDATE " +

Object.getOwnPropertyNames(firstData).map((field) => `${field}=VALUES(${field})`).join(", "));
    } catch (error) {
      console.log(error);
      return null;
    }
    return item;
  }
```

Kode 6.8 Impelmentasi fungsi create or update

Untuk memudahkan pengembangan kedepannya maka dibuat dua kelas untuk mengakses masing-masing tabel secara lebih spesifik yaitu tabel model dan tabel compound_model. Kedua kelas tersebut merupakan anak dari kelas DatabaseRepo sehingga mewarisi fungsi get dan createOrUpdate. Untuk membantu dalam mengetahui variabel dari tabel model dan compound_model maka dibuat sebuah file deskriptor untuk mendeskripsikan kunci dan isian dalam tabel yaitu ModelDescriptor dan CompoundModelDescriptor.

```
export interface ModelDescriptor {
   id?: string;
   name: string;
   class: Array<string>;
   providerProtocol: string;
   providerModelId: string;
   providerUrl: string;
   providerPort: number;

}

export interface CompoundModelDescriptor {
   id?: string;
   name: string;
   flow: string;
}
```

Kode 6.9 Tabel descriptor

```
export default class ModelRepo extends DatabaseRepo<ModelDescriptor> { }

export default class CompoundModelRepo extends DatabaseRepo<CompoundModelDescriptor> { }
```

Kode 6.10 Implementasi model dan compound_model tabel

5.2.3 Implementasi fungsi prediksi

Pada bagian ini akan diimplementasikan fungsionalitas untuk memprediksi kalimat menggunakan satu model klasifikasi.

```
async predict(request: any, response: any): Promise<JsonObject> {
    let result: [string, number];
    let channel: ModelDescriptor;
    const modelId = request.params.id;
    const msg = request.query.msg;
    if (msg) {
      try {
        result = await this.ModelManager.predict(msg, modelId);
        if (result) {
          response.json({
            status: "success",
            message: "You've succeed",
            predict: result
          });
        }
        else {
          response.json({
            status: "error",
            error: "Model not found"
          });
        }
        return response;

      } catch (e) {
        this.logger.error("Service error: ", e);
        if (e.code) response.status(e.code);
        response.json({
          status: "error",
          error: e.message || e.toString()
        });
        return response;
      }
    }
    response.json({
      status: "error",
      message: "Text not specified",
```

```
            predict: result
        });
        return response;
    }
```

Kode 6.11 Implementasi fungsi predict

Kode 6.11 akan mengatur ketika terjadi panggilan menggunakan method GET /predict/{id}?msg=. Kembalian yang dihasilkan terdiri dari success dan error. Ketika error karena id model tidak ditemukan maka akan diberi peringatan bahwa model tidak ditemukan. Ketika id model ditemukan maka fungsi predict dari ModelManager akan diakses dengan parameter berupa message dan modelId. Parameter message didapatkan dari query get msg, sedangkan parameter id didapatkan dari request.params.id.

```
export interface IModelManager {

    predict(msg: string, id: string): Promise<[string, number]>;

    predictCompoundModel(msg: string, id: string): Promise<[string, number]>;

    createOrUpdateCompoundModel(body: JsonObject): Promise<CompoundModelDescriptor | CompoundModelDescriptor[]>;

}
```

Kode 6.12 Model manager descriptor

```
export default class ModelManager extends Component implements IModelManager {

    private readonly modelEntity = "model";

    private readonly compoundModelEntity = "compound_model";

    constructor(private modelRepo: IRepo<ModelDescriptor>, private compoundModelRepo: IRepo<CompoundModelDescriptor>) {
```

```
        super();
    }
}
```

Kode 6.13 Model manager class

Kode 6.13 merupakan kelas ModelManager dengan beberapa injeksi dari modelRepo dan compoundModelRepo. Merapi memungkinkan untuk melakukan injeksi komponen yang telah didefinisikan pada service.yml. Implementasi fungsi akan sesuai dengan deskripsi pada Kode 6.12.

```
async predict(msg: string, id: string): Promise<[string, number]> {
        let result: [string, number];
        let model = await this.modelRepo.get(id, this.modelEntity);
        if (model && model.id) {
            if (model.providerProtocol == "merapi") {
                let provider: IVerstandClassifier;
                provider = await proxy<IVerstandClassifier>(`${model.providerUrl}:${model.providerPort}`, { secret: "" });
                result = await provider.predictProba(msg, model.providerModelId);
                return result;
            }
        }
        return null;
    }
```

Kode 6.14 Implementasi pemanggilan predict pada verstand service

Ketika fungsi predict pada ModelManager dieksekusi maka Kode 6.14 akan dieksekusi. Pertama mencari apakah id dari model tersedia dalam basis data menggunakan modelRepo. Hasil dari pencarian tersebut adalah informasi mengenai data model seperti url, port modelId dan lainnya. Setelah mendapatkan informasi tersebut *verstand service* akan dipanggil untuk melakukan prediksi terhadap *message*. Hasil

dari pemanggilan predict pada *verstand service* berupa array [<label>, <threshold>]. Label merupakan hasil klasifikasi sedangkan threshold adalah berapa tingkat kepercayaan dari model terhadap hasil klasifikasi.

5.2.4 Implementasi fungsi pembuatan klasifikasi hirarki

Pada bagian ini akan diimplementasikan fungsi untuk membuat klasifikasi secara hirarki menggunakan model yang ada atau telah didaftarkan pada basis data *web service.* Untuk mengakses fungsi ini dilakukan dengan methode POST /compound/{id}. Apabila id telah terdaftar pada basis data maka data secara otomatis akan memperbarui data tersebut.

```
async createOrUpdateCompound(request: any, response: any): Promise<JsonObject> {
    const body = request.body;
    body.id = request.params.id;
    if (body.name && body.flow) {
      const result = await this.ModelManager.createOrUpdateCompoundModel(body);
      console.log(result);
      if (result) {
        response.json({
          status: "success",
          message: "You've succeed",
          compound: result
        });
      }
      else {
        response.json({
          status: "error",
          error: "Failed to create flow compound",
        });
      }
    }
    else {
      response.json({
```

```
            status: "error",
            error: "Failed to create flow compound",
        });
    }
    return response;
}
```

Kode 6.15 Pemanggilan fungsi pembuatan klasifikasi hirarki

Kode 6.15 akan memastikan bahwa pada request terdapat isian dengan kunci flow. Di mana flow merupakan sebuah json object yang mendefinisikan alur dari klasifikasi hirarki.

```
async createOrUpdateCompoundModel(body: JsonObject):
Promise<CompoundModelDescriptor |
CompoundModelDescriptor[]> {
    try {
        // if (this.checkFlow(body.flow as JsonObject)) {
        let item: CompoundModelDescriptor = {
            id: body.id as string || uuid(),
            name: body.name as string,
            flow: JSON.stringify(body.flow)
        };

        const result = await
this.compoundModelRepo.createOrUpdate(item,
this.compoundModelEntity);
        if (result) {
            item.flow = JSON.parse(item.flow);
            return item;
        }
        return null;
        // }

    } catch (error) {
        console.log(error);
        return null
    }
}
```

Kode 6.16 Create or update pada model manager

Setelah dipastikan bahwa request memiliki *body* dan *flow*. Maka fungsi createOrUpdate pada modelManager akan dipanggil dan memanggil lagi compoundModelRepo untuk memasukkan data yang telah dibuat. Data flow akan dirubah menjadi string agar dapat dimasukan ke dalam basis data. Setelah itu id, nama dan alur akan dijadikan isian kembalian.

5.2.5 Implementasi fungsi prediksi klasifikasi hirarki

Pada bagian ini pengguna akan dapat menggunakan compoundModel yang telah dibuat pada tahap sebelumnya untuk mengklasifikasikan teks secara dengan struktur hirarki.

```
async predictCompound(request: any, response: any): Promise<JsonObject> {
    const modelId = request.params.id;
    const msg = request.query.msg;
    let result = await this.ModelManager.predictCompoundModel(msg, modelId);
    if (result) {
      response.json({
        status: "success",
        message: "You've succeed",
        predict: result
      });
    }
    else {
      response.json({
        status: "error",
        error: "Model not found"
      });
    }
    return response;
  }
```

Kode 6.17 Implementasi fungsi predict compound

Kode 6.17 akan menangai pemanggilan dengan fungsi GET /predict/compound/{id}?msg=. Parameter id akan berbeda dengan modelId dimana id merupakan hasil id dari proses

pembuatan modelCompound yang mendefinisikan alur. Setelah itu id dan msg akan diteruskan pada fungsi predictCompoundModel pada ModelManager.

```
async predictCompoundModel(msg: string, id: string): Promise<[string, number]> {
    const compoundModel = await this.compoundModelRepo.get(id, this.compoundModelEntity);
    if (compoundModel && compoundModel.id) {
      let flow = JSON.parse(compoundModel.flow);
      let currentPredict: [string, number] = null;
      while (true) {
        if (typeof flow === 'string') {
          currentPredict = await this.predict(msg, flow);
          break;
        } else {
          let modelId = flow.model_id as string;
          currentPredict = await this.predict(msg, modelId);
          const classifier = Object.keys(flow.class);
          if (classifier.indexOf(currentPredict[0]) > -1) {
            flow = (<JsonObject>flow.class)[currentPredict[0]] as JsonObject;
          } else {
            break;
          }
        }
      }
      return currentPredict;
    }

    return null;
  }
```

Kode 6.18 Looping predict model manager

Kode 6.18 akan mencari baris sesuai dengan id yang didefinisikan oleh pengguna, apabila data ditemukan maka informasi mengenai alur (*flow*) akan dikembalikan. *Looping*

akan dilakukan sebanyak pengguna mendefinisikan hirarki dari model klasifikasi. Pada putaran pertama teks akan diklasifikasikan menggunakan hirarki pertama, setelah itu klasifikasi akan diteruskan sesuai dengan hasil klasifikasi pertama. Perputaran akan berhenti apabila klasifikasi terakhir adalah string. Hasil akhir yang diberikan adalah klasifikasi terakhir dengan threshold terakhir pada model klasifikasi.

5.3 *Deployment*

Pada bagian ini kebutuhan fungsional sudah terpenuhi dari segi pembuatan model klasifikasi sentimen sampai dengan pembuatan *web service*. Namun, untuk memudahkan proses *deployment* maka *web service* yang telah dibuat akan dibuat ke dalam sebuah image sehingga dapat dimanfaatkan secara lebih fleksibel. Pembuatan image akan menggunakan bantuan Docker.

```
version: "2"

services:
  mysqldb:
    image: mariadb:10.2.6
    ports:
      - "3306:3306"
    environment:
      - MYSQL_USER=kata
      - MYSQL_PASSWORD=kata-kata888!
      - MYSQL_ROOT_PASSWORD=rootisgod
      - MYSQL_DATABASE=ml_unity
```

Kode 6.19 Pengaturan basis data mysql pada docker

Kode 6.19 Dibuat pada sebuah file docker-compose.yml yang berfungsi untuk mendefinisikan mysql untuk dijadikan sebuah image. Karena mysql memiliki image dari docker secara default maka kita tidak perlu membuat service mysql. Hal yang perlu diperhatikan adalah *environment* di mana hal tersebut berguna untuk mendefinisikan kredential untuk basis data.

```
service-starter:
  build: .
  volumes:
    - .:/code
  command: pm2-docker start.js
  links:
    - mysqldb
  ports:
    - "8000:8000"
```

Kode 6.20 Pengaturan web service pada docker

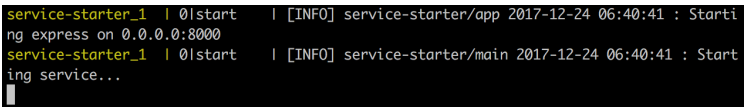
Kode 6.20 mendefinisikan *web service* untuk dijadikan sebuah image. Sebuah images nantinya bisa berisikan banyak *service*, *web service* akan dijalankan dengan port 8000 dan memiliki relasi terhadap service mysqldb seperti yang telah dibuat pada Kode 6.19. *Web service* memiliki beberapa pengaturan sebelum dapat dijalankan yaitu *library* yang harus diinstal seperti halnya typescript, merapi dan lainnya. Oleh karena itu pada Kode 6.21 menjelaskan step yang dilakukan sebelum menjalankan *web service* tersebut dengan menginstall pm2, typescript dan mengunduh semua dependensi menggunakan library *yarn*. Untuk melakukan hal tersebut butuhkan *library npm*.

```
# Dockerfile
FROM node:6.9
MAINTAINER "kata.ai"

ENV NPM_TOKEN xxx

# Set environment variables
ENV APPDIR /code

# Set the work directory
RUN mkdir -p ${APPDIR}
WORKDIR ${APPDIR}

ADD . ${APPDIR}
```

```
COPY ./.npmrc .

RUN npm i -g pm2
RUN npm i -g typescript@2.3.4
RUN tsc; exit 0
RUN npm i -g yarn
RUN yarn install

ARG NODE_ENV
ENV NODE_ENV ${NODE_ENV:-production}

RUN rm .npmrc
RUN rm -rf docker

EXPOSE 8000


HEALTHCHECK --interval=5s --timeout=3s --retries=3
CMD curl -f http://localhost:5000/info || exit 1
```

Kode 6.21 Pengaturan docker file

Kode 6.21 akan melakukan tahapan tersebut dan pada akhirnya membuka port 8000 sehingga dapat diakses secara umum. Kode tersebut disimpan dalam sebuah file Dockerfile. Setelah semua telah siap maka Docker dapat dinyalan dan mengetikkan perintah *docker-compose up* pada terminal dan direktori *web service* berada.



Gambar 6.2 Web service berjalan pada port 8000

Setelah proses *deployment* berhasil maka model yang sebelumnya telah dipilih akan diimplementasikan pada *web service* ini. Table 6.1 menjelaskan informasi dari model yang diimplementasikan.

Table 6.1 Daftar model yang diimplementasikan

Nama model	Tipe model	Id model	Kelas
fasttext-repl_topics	datar	a7405625-e61a-48df-934e-2a5b9bccb218	["upset", "mocking", "sarcasm", "denial", "spam","closing", "neutral", "pleasant", "compliment"]
fasttext-rnn_topics	hirarki	61182486-4696-49a5-bef3-f959803782e2	["positive", "negative", "neutral"]
fasttext-rnn_positive	hirarki	6dae8727-8a5f-47fc-966f-30e1e4e80a5b	["pleasant", "compliment"]
fasttext-rnn_negative	hirarki	b83b8da2-2185-4188-a829-4b00ac548289	["upset", "mocking", "sarcasm", "denial"]
fasttext-rnn_neutral	hirarki	653c744e-5796-4fd3-b2b6-b6783ceb66ce	["spam","closing", "neutral"]

Untuk memasukan model ke dalam *web service* dapat dilakukan secara manual dengan query pada MySQL, karena *verstand service* dijalankan pada komputer *local* maka Table 6.2 menjelaskan beberapa parameter dari informasi yang digunakan oleh *verstand service*.

Table 6.2 Informasi verstand service

Parameter	Isi
providerUrl	http://docker.for.mac.localhost
providerProtocol	merapi
providerPort	5001

Setelah mengetahui informasi verstand service maka lakukan query insert menggunakan dbms yang tersedia. Query yang digunakan untuk memasukkan data model dideskripsikan pada Table 6.3.

Table 6.3 Query untuk mengisikan model pada basis data

```
INSERT INTO model (id, name, class, providerProtocol, providerModelId, providerUrl, providerPort)VALUES

('sentimen', 'fasttext-repl_topics','["upset", "mocking", "sarcasm", "denial", "spam","closing", "neutral", "pleasant", "compliment"]','merapi","a7405625-e61a-48df-934e-2a5b9bccb218","http://docker.for.mac.localhost",5001),

('hirarki-topik', 'fasttext-repl_topics ','["positive", "negative", "neutral"]','merapi","61182486-4696-49a5-bef3-f959803782e2","http://docker.for.mac.localhost",5001),

('hirarki-positif', 'fasttext-repl_positive ','["pleasant", "compliment"]','merapi","6dae8727-8a5f-47fc-966f-30e1e4e80a5b","http://docker.for.mac.localhost",5001),

('hirarki-negatif', 'fasttext-repl_negative ','["upset", "mocking", "sarcasm", "denial"]','merapi","b83b8da2-2185-4188-a829-4b00ac548289","http://docker.for.mac.localhost",5001),
```

```
('hirarki-netral', 'fasttext-repl_neutral ','["spam","closing",
"neutral"]',"merapi","653c744e-5796-4fd3-b2b6-
b6783ceb66ce","http://docker.for.mac.localhost",5001)
```

Setelah memasukkan semua model pada basis data selanjutnya masukkan alur untuk menggunakan model hirarki menggunakan query yang didefinisikan pada Table 6.4. Kolom flow berguna untuk mendefinisikan model yang digunakan untuk mengklasifikasikan data secara hirarki. Isian dari model_id dan class <key>:<value> harus mengikuti informasi id dan class pada Table 6.3.

Table 6.4 Query model hirarki

```
INSERT INTO compound_model (id, name, flow) VALUES
("sentimen", "hirarki-sentimen", '{
                "model_id": "hirarki-topik",
                "class":{
                        "positive": "hirarki-positif",
                        "negative":"hirarki-negatif",
                        "neutral":"hirarki-netral"
                }}')
```

BAB VI
HASIL DAN PEMBAHASAN

6.1 Pengujian

Pada bagian ini *web service* yang telah dikembangkan akan dilakukan *unit testing* dan uji integrasi pada fungsionalnya sehingga memenuhi kebutuhan fungsional.

6.1.1 Pengujian unit *web service*

Pada bagian ini akan dilakukan pengujian unit sesuai dengan kebutuhan fungsional yaitu predict, predictCompound dan membuat modelCompound. *Library* yang digunakan untuk melakukan pengujian ini adalah *mocha*, file yang akan diuji ada ModelManager karena disini merupakan fungsionalitas sistem. Oleh karena itu buat file model_manager.spec.ts pada folder test. Untuk menjalankan pengujian ini digunakan sebuah modul npm dengan perintah *npm test.*

```
import { suite, test, skip, only } from "mocha-typescript";
import { equal } from "assert";
import { IConfigReader, Config } from "merapi";
import ModelManager from "../../components/managers/model_manager";
import { ModelDescriptor, CompoundModelDescriptor } from "interfaces/descriptors";
import ModelRepo from "../../components/repos/model_repo";
import CompoundModelRepo from "../../components/repos/compound_model_repo";
import * as yaml from "js-yaml";
import * as fs from "fs";
import * as sinon from "sinon";

const knexTestConfig = {
  client: "sqlite3",
  connection: { filename: ":memory:" },
  migrations: {
```

```
        directory: "./migrations"
    },
    useNullAsDefault: true
}
const knexTest = require("knex")(knexTestConfig);
const modelEntity = "model";
const compoundModelEntity = "compound_model";
const topicModel: ModelDescriptor = {
    id: "topic",
    name: "test",
    class: ["positive", "neutral", "negative"],
    providerProtocol: "verstand",
    providerModelId: "test",
    providerUrl: "http://test.com",
    providerPort: 5001,

}
const subtopicModel: ModelDescriptor = {
    id: "subtopic",
    name: "test",
    class: ["pleasant", "compliment"],
    providerProtocol: "verstand",
    providerModelId: "test",
    providerUrl: "http://test.com",
    providerPort: 5001,

}
const compoundModelTest = {
    id: "test",
    name: "test",
    flow: {
        model_id: "topic",
        class: {
            compliment: "subtopic"
        }
    }
}
```

Kode 7.1 Kebutuhan library, modul dan static variabel

Pada Kode 7.1 mendefinisikan kebutuhan dari pengujian dan beberapa variabel static yang digunakan sebagai nilai default. Dalam melakukan pengujian maka lingkungan pengujian harus terisolasi atau bersifat independen. Sehingga perlu dilakukan pengaturan terhadap lingkungan pengujian seperti database, *stubbing*. Sehingga pada Kode 7.2 menginisiasi modul yang dibutuhkan lingkungan dalam menjalankan proses pengujian sesuai dengan skenario. Di mana sebelum pengujian dijalankan basis data akan diisi dengan beberapa contoh dari model seperti yang didefinisikan pada Kode 7.1. Setelah pengujian selesai maka tabel tersebut akan dibersihkan.

```
private modelManager: ModelManager;
   private modelRepo: ModelRepo;
   private compoundModelRepo: CompoundModelRepo;
   private config: IConfigReader;

   private predictTrue = {
      msg: "Halo",
      label: "neutral",
      id: "topic"
   };
   constructor() {
      console.debug = function (message: string) {
         return "debuged";
      }

      let configJson = yaml.safeLoad(fs.readFileSync("./service.yml", "utf8"));
      this.config = Config.create(configJson);

      this.modelRepo = new ModelRepo(this.config, console, knexTest);
      this.compoundModelRepo = new CompoundModelRepo(this.config, console, knexTest);
      this.modelManager = new ModelManager(this.modelRepo, this.compoundModelRepo);
```

```
    }
    static async before() {
        await knexTest.migrate.latest();
        await knexTest(modelEntity).insert(topicModel);
        await knexTest(modelEntity).insert(subtopicModel);
        await
knexTest(compoundModelEntity).insert(compoundModelT
est);
    }

    async after() {
        await knexTest(modelEntity).truncate();
        await knexTest(compoundModelEntity).truncate();
    }
```

Kode 7.2 Inisiasi modul

Pada Kode 7.3 dilakukan pengujian untuk mengetahui apakah *web service* dapat menerima masukan dan meneruskannya ke *verstand service* untuk memprediksi sebuah teks menggunakan model klasifikasi sentimen non-hirarki. Proses pemanggilan *verstand service* akan digantikan menggunakan *stub* karena *verstand service* bukan merupakan layanan yang terdapat dalam *web service.* Sedangkan Kode 7.4 akan menguji apabila model tidak dapat ditemukan dengan memberikan parameter kosong.

```
@test async "should success to predict message"() {
        let expected = this.predictTrue.label;
        sinon.stub(this.modelManager,
"callApi").returns([this.predictTrue.label, 1]);
        let actual = await
this.modelManager.predict(this.predictTrue.msg,
this.predictTrue.id);
        equal(expected, actual[0]);
    }
```

Kode 7.3 Pengujian prediksi menggunakan model non-hirarki

```
@test async "should return null to predict message with
wrong model id"() {
```

```
        let expected = this.predictTrue.label;
        sinon.stub(this.modelManager,
"callApi").returns([this.predictTrue.label, 1]);
        let actual = await
this.modelManager.predict(this.predictTrue.msg, "");
        equal(null, actual);
    }
```

Kode 7.4 Pengujian prediksi dengan model id yang salah

Kode 7.5 akan memasukkan data untuk mendefinisikan model klasifikasi hirarki ke dalam basis data. Setelah itu akan dilihat apakah keluaran dari fungsi tersebut sudah sesuai dengan masukannya.

```
@test async "should create to compound model"() {
        sinon.stub(this.compoundModelRepo,
"createOrUpdate").returns(compoundModelTest);
        let result = await
this.modelManager.createOrUpdateCompoundModel(comp
oundModelTest) as CompoundModelDescriptor;
        equal(compoundModelTest.id, result.id);
    }
```

Kode 7.5 Pengujian mendefinisikan model hirarki

Setelah sebelunya berhasil mendefinisikan model hirarki, maka pada Kode 7.6 pengujian terhadap akses model tersebut akan diuji. Di mana keluaran yang diharapkan adalah teks memiliki label *compliment* yang sama dengan hasil prediksi menggunakan model hirarki. Sedangkan Kode 7.7 akan menguji apabila id dari compound model tidak dapat ditemukan dengan memberikan parameter kosong.

```
@test async "should success to predict message compound
model"() {
        sinon.stub(this.compoundModelRepo, "get").returns({
id: compoundModelTest.id, name:
compoundModelTest.name, flow:
JSON.stringify(compoundModelTest.flow) });
        sinon.stub(this.modelManager,
"predict").returns(["compliment", 1]);
```

```
    let result = await this.modelManager.predictCompoundModel(this.predictTrue.msg, compoundModelTest.id);
    equal("compliment", result[0]);
  }
```

Kode 7.6 Pengujian prediksi menggunakan model hirarki

```
@test async "should return null to predict message with wrong compound model id"() {
    sinon.stub(this.modelManager, "predict").returns(["compliment", 1]);
    let result = await this.modelManager.predictCompoundModel(this.predictTrue.msg, "");
    equal(null, result);
  }
```

Kode 7.7 Pengujian prediksi dengan compound model id yang salah

6.1.2 Pengujian integrasi

Pada pengujian ini *web service* akan dijalankan bersamaan dengan *verstand service* untuk diuji secara manual menggunakan aplikasi postman. Terdapat dua skenario utama dalam pengujian ini yaitu pengujian untuk klasifikasi sentimen dan pengujian terhadap penanganan kesalahan terhadap request *web service*. Uji coba klasifikasi sentimen akan dilakukan menggunakan teks yang didefinisikan pada Table 7.1 yang masing-masing teks dijalankan pada skenario pada Table 7.2, data pada Table 7.1 diambil secara acak pada data pelatihan.

Table 7.1 Daftar teks uji coba fungsionalitas klasifikasi sentimen

No	Teks	Kelas
1	Wihh belum pi sa cerita tapi ada mi saran mu . Hebat hebat	Compliment
2	Aku tuh punya pacar tapi dia di korea, trus aku liat dia suka foto foto sama cewek cewek aku kan jadi cemburu	Upset
3	Halo jemma	Neutral

4	Spotify Premium RESMI\nlagi PROMO\n\nhttps://line.me/R/ti/p/%40wth0732m\nhttps://line.me/R/ti/p/%40wth0732m\nhttps://line.me/R/ti/p/%40wth0732m\n\n\nSPOTIFY PREMIUM SELAMANYA ONLY 70K\n(3 months guarantee)\n\n\nSpotify ini RESMI, login via email password. Buruan orde	Spam

Table 7.2 Skenario uji coba fungsionalitas klasifikasi sentimen

No	Method	Uri	Tipe model
1	GET	/predict/ sentimen	datar
2	GET	/predict/compound/sentimen	hirarki

Pengujian terhadap penanganan kesalahan sistem ketika sebuah *request* diterima oleh *web service* akan dilakukan dengan memasukkan beberapa parameter yang tidak sesuai dengan fungsionalitas. Skenario didefinisikan pada Table 7.3 dimana terdapat *method,* Uri dan tipe kesalahan yang diakan diujikan.

Table 7.3 Skenario pengujian penanganan kesalahan

No	Method	Uri	Kesalahan
1	GET	/predict/ ?msg	modelId datar dan teks tidak didefinisikan
2	GET	/predict/compound/	modelId hirarki tidak didefinisikan
3	GET	/predict/null?msg=test	modelId tidak ada dalam database
4	GET	/predict/sentimen?msg=	teks kosong
5	GET	/predict/sentimen	teks tidak didefinisikan

6	GET	/predict/compound/sentimen?msg=	teks kosong
7	GET	/predict/compound/sentimens?msg=aa	modelId tidak ada dalam database

6.2 Hasil

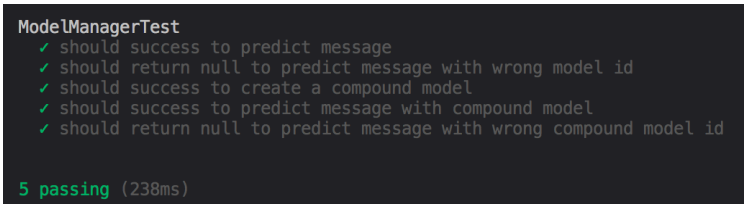
Setelah melalui beberapa validasi maka pada bagian ini hasil dari pengujian terhadap *web service* yang mengakses model klasifikasi sentimen akan dipaparkan.

6.2.1 Hasil pengujian web service

Bagian ini menjelaskan hasil *unit testing* yang dilakukan pada *web service*. Kasus yang di tes berjumlah 5 buah dan hasil yang didapatkan yaitu,

Table 7.4 Hasil pengujian unit web service

Kode case	Pass
C1	Ya
C2	Ya
C3	Ya
C4	Ya
C5	Ya



Gambar 7.1 Hasil pengujian unit web service

Aplikasi *web service* memiliki hasil *unit testing* dengan tingkat keberhasilan 100%. Pengujian yang dilakukan merupakan pengujian yang tertutup dengan membuat *stub* untuk

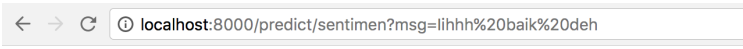
melakukan pengujian terhadap *verstand service*. Selain itu pada pengujian Table 7.4 skenario negasi diuji dengan menggunakan id kosong pada parameter modelId, sehingga aplikasi akan mengembalikan nilai null bila modelId tidak ditemukan.

Table 7.5 Hasil uji web service

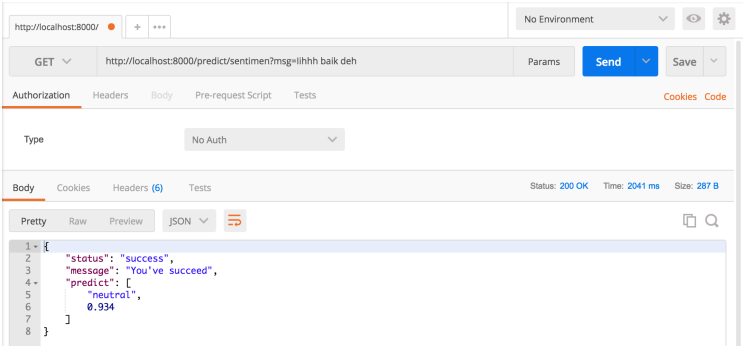
Total kebutuhan fungsional	**Total *test case***	**Total *test case* yang berhasil**	**Total *test case* yang gagal**
3	5	5	0

6.2.2 Hasil uji integrasi web service

Uji integrasi digunakan untuk mengetahui apakah fitur yang telah dikembangkan pada aplikasi *web service* dapat berjalan sesuai dengan kebutuhan fungsional. Oleh karena itu pada Gambar 7.2 dan Gambar 7.3 menunjukan hasil dari penggunaan aplikasi *web service.* Dimana pada Gambar 7.2 digunakan sebuah browser Google Chrome untuk mengakses *web service* sedangkan pada Gambar 7.3 digunakan aplikasi postman untuk mengakses *web service.*



Gambar 7.2 Hasil akses model datar melalui Google Chrome



Gambar 7.3 Hasil akses model datar melalui Postman

Berikut merupakan hasil uji coba yang telah disesuaikan dengan skenario Table 7.1 untuk melihat hasil klasifikasi sentimen dari *web service.*

Table 7.6 Hasil uji coba klasifikasi sentimen pada web service

Teks	Kelas	Skenario 1(model datar)			Skenario 2(model hirarki)		
		Prediksi	Skor	Waktu	Prediksi	Skor	Waktu
1	compliment	compliment	0.971	36	compliment	0.968	62
2	upset	upset	0.742	38	upset	1	65
3	neutral	neutral	1	27	neutral	1	50
4	spam	spam	0.995	42	spam	0.997	90

Pada Table 7.6 beberapa hal yang diuji yaitu prediksi kelas, skor prediksi dan waktu respon yang dierima dalam satuan *milisecond.* Semua model memiliki akurasi sebesar 100%. Namun terdapat sedikit perbedaan yaitu pada model datar skor untuk teks nomor 2 mendapatkan skor lebih kecil yaitu 0.74 dibandingkan dengan model hirarki 1, semakin kecil skor maka prediksi dinilai oleh sistem semakin kecil kebenarannya. Waktu respon pada teks dengan jumlah karakter yang banyak berpengaruh pada waktu respon dari sistem, yaitu semakin banyak maka waktu yang dibutuhkan juga semakin besar. Waktu respon pada model hirarki lebih besar dari model datar yaitu dengan rata-rata pada model hirarki sebesar 66.75 ms sedangkan pada model datar sebesar 35.75.

Pada Table 7.7 merupakan hasil dari pengujian penanganan kesalahan oleh sistem dengan skenario Table 7.3. Beberapa uri diuji coba dengan parameter yang tidak sesuai dengan seharusnya seperti menghilangkan parameter msg atau mengisi modelId yang tidak sesuai.

Table 7.7 Hasil uji coba penanganan kesalahan

Sken-ario	Uri	Respon
1	/predict/ ?msg	{ "status": "error", "message": "Text not specified" }
2	/predict/compound/	{ "status": "error", "message": "Text not specified" }
3	/predict/null?msg=test	{ "status": "error", "error": "Model not found" }
4	/predict/sentimen?msg=	{ "status": "error", "message": "Text not specified" }
5	/predict/sentimen	{ "status": "error", "message": "Text not specified" }
6	/predict/compound/sentimen?msg=	{ "status": "error", "message": "Text not specified" }
7	/predict/compound/sentimens?msg=aa	{ "status": "error", "error": "Model not found" }

Dari Table 7.7 dapat dilihat bahwa secara umum penanganan kesalahan pada sistem sudah cukup baik dengan memberikan dua respon yaitu apabila modelId tidak ditemukan dan apabila teks atau *message* berisikan kosong atau tidak didefinisikan sama sekali. Sistem akan mengembalikan *status error* dan memberikan error yang dilakukan oleh pengguna.

Halaman ini sengaja dikosongkan

BAB VII
KESIMPULAN DAN SARAN

Pada bab ini dibahas mengenai kesimpulan dari semua proses yang telah dilakukan dan saran yang dapat diberikan untuk pengembangan yang lebih baik.

7.1 Kesimpulan

Hasil dari pembuatan model klasifikasi sebelumnya didapatkan bahwa pengukuran performa menggunakan akurasi saja tidak cukup, Table 4.22 terlihat walaupun akurasi logistic regression cukup tinggi namun presisinya hampir 60% lebih kecil dari akurasinya. Pembuatan model klasifiasi sentimen menggunakan algoritma *fasttext* memiliki hasil yang lebih optimal untuk melakukan klasifikasi pada dataset yang memiliki jumlah kelas yang tidak seimbang. Hal tersebut dapat dilihat pada Table 4.23 dimana nilai presisi tertinggi tiap kelas hampir dimiliki oleh *fasttext.* Namun, akurasi menggunakan algoritma *fasttext* kemungkinan memiliki *overfitting* terhadap data pelatihan karena pada validasi menggunakan data pengujian hasil presisi lebih kecil dibandingkan *logisitc regression* dan presisi 50% lebih kecil dari presisi data pelatihan. Pemecahan solusi dengan model hirarki memiliki nilai presisi yang sedikit lebih baik yaitu sekitar 1% diatas model datar. Hal tersebut dicapai menggunakan skenario 7 yaitu skenario dengan memanfaatkan fitur POS Tagging tanpa menggunakan pengubahan kata yang memiliki unsur negatif dan positif.

Aplikasi *web service* tersebut akan memanggil fungsi *predict* dari *verstand service* untuk melakukan klasifikasi menggunakan model yang dipilih yaitu model dengan pembelajaran *fasttext*. Prediksi menggunakan model hirarki dan teks yang panjang meningkatkan waktu proses dari aplikasi *web service* sehingga hasil respon cenderung lebih besar dari menggunakan model datar dan teks yang lebih sedikit. Apabila terdapat sebuah parameter yang tidak sesuai diakses pada *web*

service, maka sistem dapat memberikan respon yang memperlihatkan parameter yang salah kepada pengguna.

7.2 Saran

Dari pengerjaan tugas akhir ini, adapun beberapa saran untuk pengembangan penelitian ke depan.

1. Dalam menggunakan pelatihan untuk model klasifikasi sentimen usahakan menggunakan dataset yang memiliki kelas dengan jumlah yang setara, terutama apabila menggunakan algoritma *logistic regression.*
2. Untuk pengembangan selanjutnya mungkin perlu menambahkan skenario dengan model hirarki yang kelasnya dikelompokkan menggunakan sebuah model *unsupervised.*
3. Menambahkan skenario dengan fitur lain karena pada pengembangan ini fitur yang digunakan tidak terlalu memberikan dampak yang signifikan terhadap akurasi.
4. Menambahkan fungsionalitas yang lebih dinamis pada *web service* untuk mengatur model yang didefinisikan seperti membuat model, menghapus model, memperbarui model yang dilakukan melalui pemanggilan API pada *web service*.
5. Memberikan *user interface* untuk pemanfaatan fitur fungsionalitas.
6. Menambahkan dan mengintegrasikan model dari berbagai sumber dan berbagai protokol yang digunakan untuk mengakses model tersebut. Sehingga model yang dapat diakses bersifat lebih umum dan global.

DAFTAR PUSTAKA

[1] H. Banirestu, "Startup Kata.ai Raih Pendanaan Sekitar Rp46,5 Miliar," *SWA.co.id*, 30-Aug-2017. .

[2] "Chatbot Dongkrak Pendapatan Kata.Ai 36 Kali." [Online]. Available: http://industri.bisnis.com/read/20171212/105/717663/chatbot-dongkrak-pendapatan-kata.ai-36-kali-l. [Accessed: 21-Dec-2017].

[3] Peter Jarman, "Microservices – A New Application Paradigm," 2017.

[4] "IBM Knowledge Center - Advantages of web services." [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SSB23S_1.1.0.14/gtps6/s6wsadv.html. [Accessed: 21-Dec-2017].

[5] B. Liu, *Sentiment analysis and opinion mining*. San Rafael: Morgan & Claypool, 2012.

[6] Nuke Y. A. Faradhillah,Renny Pradina Kusumawardani,Irmasari Hafidz, "Eksperimen Sistem Klasifikasi Analisa Sentimen Twitter Pada Akun Resmi Pemerintah Kota Surabaya Berbasis Pembelajaran Mesin," *Seminar Nasional Sistem Informasi Indonesia (SESINDO)*, Nov. 2016.

[7] G. Buntoro, "Analisis Sentimen Calon Gubernur DKI Jakarta 2017 Di Twitter," vol. 1, Mar. 2017.

[8] Markus and Nico Saputro, "Perbandingan Statistik Bahasa N-Gram Bahasa Indonesia Dalam Cryptanalysis Simple Substitution Cipher Berbasis Algoritma Genetik," *Seminar Nasional Teknologi Informasi*, 2006.

[9] Sukhnandan Kaur and Rajni Mohana, "Prediction of Sentiment from Textual Data Using Logistic Regression Based on Stop Word Filteration and Volume of Data," *International Journal of Control eory and Applications*, vol. 9, 2016.

[10] H. Hamdan, P. Bellot, and F. Bechet, "Lsislif: CRF and Logistic Regression for Opinion Target Extraction and

Sentiment Polarity Analysis.," in *SemEval@ NAACL-HLT*, 2015, pp. 753–758.
[11] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of Tricks for Efficient Text Classification," *arXiv:1607.01759 [cs]*, Jul. 2016.
[12] Y. Meng, "Sentiment analysis: A study on product features," 2012.
[13] K. Cacciatore *et al.*, "Exploring Opportunities: Containers and OpenStack," *OpenStack White Paper*, vol. 19, 2015.
[14] *merapi: Multi-purpose Dependency Injection Container*. kata.ai, 2017.
[15] J. D. Fernández and F. Vico, "AI methods in algorithmic composition: A comprehensive survey," *Journal of Artificial Intelligence Research*, vol. 48, pp. 513–582, 2013.
[16] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011.
[17] S. PERMISSION, "Generative and discriminative classifiers: Naive bayes and logistic regression," 2005.
[18] "Differences between L1 and L2 as Loss Function and Regularization." [Online]. Available: http://www.chioka.in/differences-between-l1-and-l2-as-loss-function-and-regularization/. [Accessed: 10-Dec-2017].
[19] C. Chatterjee and R. R. Sarkar, "Multi-Step Polynomial Regression Method to Model and Forecast Malaria Incidence," *PLoS ONE*, vol. 4, no. 3, p. e4726, Mar. 2009.
[20] "Text classification · fastText." [Online]. Available: fasttext.cc/index.html. [Accessed: 21-Dec-2017].
[21] "fastText," *Facebook Research*. [Online]. Available: https://research.fb.com/fasttext. [Accessed: 21-Dec-2017].
[22] Cornell University, "Performance Measures for Machine Learning."
[23] "Precision and recall," *Wikipedia*. 13-Dec-2017.
[24] S. McConnell, *Code complete*, 2nd ed. Redmond, Wash: Microsoft Press, 2004.

[25] “Traceability from Use Cases to Test Cases,” 04-May-2006. [Online]. Available: http://www.ibm.com/developerworks/rational/library/04/r-3217/index.html. [Accessed: 10-Jan-2018].

[26] The Testing Consultancy, “Integration Testing White Paper.”

[27] K. K. Dobbin and R. M. Simon, “Optimally splitting cases for training and testing high dimensional classifiers,” *BMC medical genomics*, vol. 4, no. 1, p. 31, 2011.

Halaman ini sengaja dikosongkan

BIODATA PENULIS

Penulis lahir di Jakarta pada tanggal 27 De- sember 1995. Merupakan anak kedua dari 2 bersaudara dan telah menempuh pendidikan formal yaitu; SD Negeri 03 Cipinang Melayu, SMP Negeri 109 Jakarta, dan SMA Negeri 71 Jakarta.

Pada tahun 2014 melanjutkan pendidikan di Jurusan Sistem Informasi FTIF - Institut Teknologi Sepuluh Nopember (ITS) Surabaya dan terdaftar sebagai mahasiswa dengan NRP 5214100119. Selama menjadi mahasiswa penulis telah mengikuti berbagai ke- giatan kemahasiswaan, baik berupa kepanitiaan ditingkat Institut. Disamping itu serta aktif sebagai Staff Riset dan teknologi HMSI ITS 2015/2016. Disamping aktif dalam kegiatan kemahasiswaan, penulis juga pernah menjadi asisten praktikum pada mata kuliah Sistem Operasi. Pada tahun keempat karena penulis tertarik dengan bidang pem- belajaran mesin dan arsitektur microservice, sehingga meng- ambil bidang minat Infrastruktur Sistem dan Keamanan Teknologi Informasi (IKTI). Penulis dapat dihubungi melalui email rama14@mhs.is.its.ac.id