BAB 2
LANDASAN TEORI

2.1 Emosi

Emosi adalah suatu perasaan yang mendorong individu untuk merespon atau bertingkah laku terhadap stimulus, baik yang berasal dari dalam maupun dari luar dirinya. Emosi cenderung terjadi dalam kaitannya dengan perilaku yang mengarah (*approach*) atau menyingkiri (*avoidance*) terhadap sesuatu, dan perilaku tersebut pada umumnya disertai adanya ekspresi kejasmanian, sehingga orang lain dapat mengetahui bahwa seseorang sedang mengalami emosi.

Beberapa studi tentang emosi manusia telah dilakukan sehingga ada kesepakatan tentang emosi dasar yaitu, takut digambarkan sebagai ancaman fisik atau sosial bagi diri sendiri, kemarahan sebagai ketidakpuasan atau frustrasi dari peran atau tujuan yang dicapai oleh orang lain, jijik saat yang jarak, kelalaian, atau penolakan dari ide untuk diri sendiri, kesedihan sebagai kegagalan atau penderitaan pada peran dan tujuan, kebahagiaan atau sukacita sebagai keberhasilan atau gerakan untuk prestasi peran berharga [6].

2.2 ISEAR (International Survey On Emotion Antecedents And Reaction)

ISEAR adalah sebuah data tentang emosi yang selama bertahun-tahun dimulai pada tahun 1990-an, sebuah kelompok besar psikolog di seluruh dunia mengumpulkan data dalam proyek ISEAR, dipimpin oleh Klaus R. Scherer dan Harald Wallbott. Mahasiswa responden, baik psikolog dan non-psikolog, diminta untuk melaporkan situasi di mana mereka mengalami semua 7 emosi utama (senang, takut, marah, sedih, jijik, malu, dan rasa bersalah). Dalam setiap kasus, pertanyaan-pertanyaan meliputi cara mereka telah mengenali situasi dan bagaimana mereka bereaksi. Data akhir ini memuat laporan tentang tujuh emosi masing-masing sekitar 3000 responden di 37 negara di 5 benua [7].

Dalam penelitian ini data latih yang digunakan diambil dari ISEAR yang berbahasa inggris kemudian diterjemahkan kedalam bahasa Indonesia tanpa mengurangi maksud dari kalimat – kalimat dalam ISEAR. Data yang diambil

sebanyak 1000 data dengan banyak masing – masing emosi adalah 250 data karena ada empat emosi yang akan diklasifikasikan.

2.3 Text Mining

Teks mining adalah salah satu bidang khusus dari data mining. Teks mining merupakan sebuah teknologi baru yang dapat digunakan untuk menambang data yang telah ada dalam sebuah database dengan membuat suatu data berupa teks yang tidak terstruktur menjadi data yang dapat dianalisa. Text mining juga dapat didefinisikan sebagai proses mendapatkan informasi secara intensif dimana pengguna berinteraksi dengan koleksi-koleksi dokumen menggunakan seperangkat tools analisis [8].

Secara umum, proses-proses pada text mining adalah mengadopsi dari proses data mining. Oleh karena itu, text mining dan data mining mempunyai banyak kesamaan arsitekturnya. Proses-proses yang ada pada text mining juga hampir sama dengan data mining. Proses-proses utama pada text mining diantaranya pemrosesan awal (*text preprocessing*), penemuan pola (*pattern discovery*), transformasi teks (*text transformation*), dan pemilihan fitur (*feature selection*).

2.4 Preprocessing

Text preprocessing adalah tahapan untuk mempersiapkan teks menjadi data yang akan diolah di tahapan berikutnya. Inputan awal pada proses ini adalah berupa dokumen teks. Teks yang akan dilakukan proses text mining pada umumnya memiliki beberapa karakteristik, diantaranya adalah memiliki dimensi yang tinggi, terdapat *noise* pada data, dan terdapat struktur teks yang tidak baik. Agar dapat dihasilkan fitur yang baik dan mewakili data dengan baik, perlu dilakukan tahapan *preprocessing* [10].

Adapun tahapan *preprocessing* yang akan dilakukan antara lain *case folding, tokenizing, stopword removal dan stemming*.

1. Case Folding

   Pada tahap ini dilakukan perubahan pada semua huruf dalam dokumen menjadi huruf kecil dan menghilangkan karakter selain a-z, dengan tujuan untuk menyeragamkan karakter dalam dokumen tersebut

2. Convert Negation

Convert Negation merupakan proses konversi kata-kata negasi yang terdapat pada suatu kalimat, karena kata negasi mempunyai pengaruh dalam merubah nilai emosi pada sebuah kalimat. Jika terdapat kata negasi maka akan disatukan dengan kata setelahnya. Kata - kata negasi tersebut meliputi kata "bukan", "tidak", "tak", "tanpa" dan "jangan".

Langkah – langkah pada tahap *convert negation* adalah sebagai berikut :

a. Kata yang digunakan adalah hasil dari *case folding*
b. Jika ditemukan lirik yang mengandung kata – kata negasi maka akan disatukan kata negasi tersebut dengan kata setelah kata negasi tersebut.

3. Tokenizing

Tokenizing merupakan tahap pemotongan kalimat berdasarkan tiap kata yang menyusunnya. Proses ini melakukan penguraian deskripsi yang semula berupa kalimat-kalimat menjadi kata-kata dan menghilangkan simbol seperti titik(.), tanda seru(!), tanda tanya (?), koma(,), spasi, *emoticon.*

4. Stopword removal

Tahapan ini bertujuan untuk menghilangkan kata yang dianggap tidak dapat memberikan pengaruh dalam menentukan suatu kategori tertentu dalam suatu dokumen. Proses ini dilakukan karena kata tersebut sering muncul hampir disetiap dokumen sehingga dianggap tidak dapat menjadi pembeda yang baik dalam membedakan kategori yang satu dengan kategori yang lain.

Setiap kata akan diperiksa apakah masuk dalam daftar *stopword*, jika sebuah kata masuk di dalam daftar *stopword* maka kata tersebut tidak akan diproses lebih lanjut dan kata tersebut akan dihilangkan. Sebaliknya apabila sebuah kata tidak termasuk di dalam daftar stopword maka kata tersebut akan masuk keproses berikutnya.

Untuk daftar *stopword* yang digunakan diambil dari websitenya budi susanto seorang praktisi IT yang fokus terhadap *software open source, semantic web and information retrieval* [11]. Banyak kata yang disimpan didatabase *stopword* sebanyak 937 kata. Kata-kata yang termasuk *stopword* tersebut biasanya berupa kata ganti orang, kata penghubung, pronominal penunjuk, dan lain sebagainya.

5. Stemming

Stemming merupakan tahap untuk mentransformasi kata-kata yang terdapat dalam suatu dokumen ke kata-kata akarnya (*root word*) dengan menggunakan aturan-aturan tertentu. Dengan menggunakan stemming dapat mengurangi variasi kata yang sebenarnya memiliki kata dasar yang sama. Salah satu algoritma stemming yaitu Algoritma Nazief dan Adriani.

Algoritma stemming Nazief dan Adriani menggunakan morfologi imbuhan sebagai berikut :

a. *Inflection suffixes* merupakan kumpulan akhiran (suffixes) yang tidak merubah kata dasar. Seperti : ‘-lah’, ‘-kah’, ‘-nya’, ‘-mu’
b. *Derivation suffixes* (DS) merupakan kumpulan akhiran (suffixes) yang langsung menempel pada kata dasar, seperti : ‘-i’, ‘-an’ dan ‘kan’.
c. *Derivation prefixes* (DP) merupakan himpunan awalan (prefixes) yang menempel langsung pada kata dasar maupun terhadap kata yang telah mempunyai sampai dua derivation prefixes. Seperti : ‘mem-‘ dan ‘per-‘.

2.5 Pembobotan (*Term Weighting*)

Term weighting merupakan tahapan untuk memberikan suatu nilai/bobot pada *term* yang terdapat pada suatu dokumen yang telah berhasil diekstrak. Metode yang akan digunakan dalam penelitian dalam pembobotan adalah pembobotan WIDF. Metode WIDF (*Weighted Inverse Document Frequency*) adalah sebuah metode pengembangan dari metode *Inverse Document Frequency* (IDF), IDF adalah proses mengukur *term* yang jarang muncul pada *corpus*. Penilaiannya menggunakan seluruh dokumen latih yang digunakan. Jika sebuah *term* sering muncul, maka *term* tersebut tidak bisa dianggap sebagai *term* yang mewakili dokumen tersebut. Sebaliknya, jika *term* yang jarang muncul pada *corpus*, maka term tersebut dikatakan memiliki hubungan dengan dokumen. *Inverse Document Frequency* merupakan perhitungan dari jumlah seluruh dokumen (D) dibagi dengan Frekuensi Dokumen (DF) dari kata (ti), dapat dituliskan dengan persamaan sebagai berikut [11]

$$\text{IDF} = \log\left(\frac{N}{Dfi}\right) \qquad (2.1)$$

Keterangan :

IDF = *inverse document frequency*

N = Jumlah kalimat yang berisi term(t)

Dfi = Jumlah kemunculan *term* terhadap D

Kelemahan dari metode IDF adalah semua teks yang berisi *term* tertentu diperlakukan sama. Artinya, IDF tidak memperhitungkan jumlah kemunculan suatu *term* pada suatu dokumen. Contohnya pada tabel 2.1 berikut, dimana d (kolom) adalah text dan t (row) adalah term yang dicari, pertemuan antara di dan tj adalah *termfrequency* dari term tj di dalam text di

	d_1	d_2	d_3	d_4	d_5
⋮	⋮	⋮	⋮	⋮	⋮
t_x	2	50	3	2	4
t_y	3	2	3	2	3
⋮	⋮	⋮	⋮	⋮	⋮

Gambar 2.1 Contoh Perhitungan Kemunculan Term

Dapat dilihat dari persamaan (2.1) bahwa IDF (tx) dan IDF (ty) masing-masing bernilai 0, karena *term* yang dicari muncul pada semua text. Sedangkan, distribusi frekuensi tx dan ty terhadap d1 sampai d5 berbeda, IDF tidak dapat mengidentifikasi perbedaannya, karena bersifat binary counting. Sehingga, tidak dapat menentukan jumlah frekuensi dari term.

IDF mengasumsikan bahwa *term* berbanding terbalik dengan jumlah text yang mengandung *term* tersebut. Oleh karena itu, bagian penting dari persamaan (1) di atas adalah faktor 1/df(t). Metode WIDF menghitung factor ini dengan *term* frekuensi. Sebagai contoh, 1/df(t) dari tabel 2.1 di atas menjadi

$$\frac{1}{1+1+1+1+1}$$

untuk semua text. Kemudian angka "1" diganti dengan frekuensi dari masing-masing term, menjadi

$$\frac{50}{2+50+3+2+4}$$

pada kasus d2. Faktor inilah yang yang disebut WIDF. Tidak seperti IDF, WIDF dapat membedakan masing-masing text d1…d5. Sehingga, WIDF dengan term t dalam text d dapat dituliskan sebagai persamaan (2) berikut :

$$WIDF(d,t) = \frac{TF(d,t)}{\sum_{i \epsilon D} TF(i,t)} \qquad (2.2)$$

dimana TF(d,t) adalah *term frequency* dari term t di dalam text d dan i menyatakan jumlah text. WIDF dari term t menjumlahkan semua *term frequency* dari dari semua kumpulan text. Dengan kata lain WIDF adalah bentuk normalisasi term frequency dari semua kumpulan text [5].

2.6 K-NN

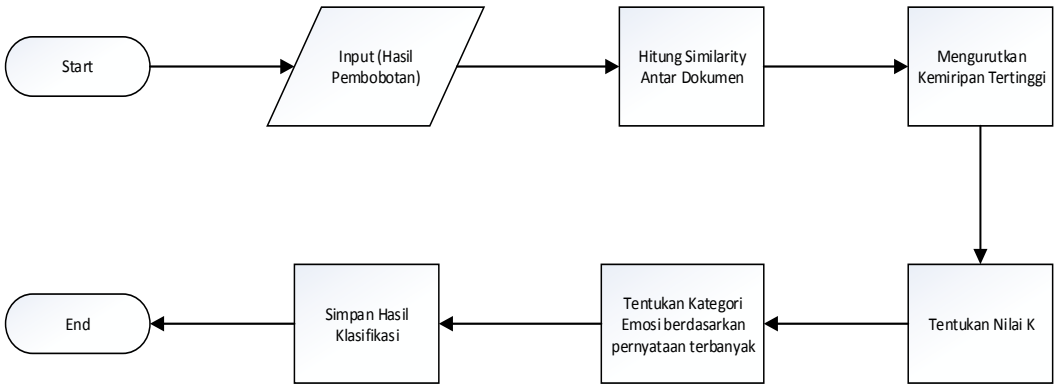
K-Nearest Neighbor (KNN) adalah suatu metode yang menggunakan algoritma supervised dimana hasil dari query instance yang baru diklasifikan berdasarkan mayoritas dari kategori pada KNN. Tujuan dari algoritma ini adalah mengklasifikasikan obyek baru berdasarkan atribut dan training sample. *Classifier* tidak menggunakan model apapun untuk dicocokkan dan hanya berdasarkan pada memori. Diberikan titik query, akan ditemukan sejumlah k obyek atau (titik training) yang paling dekat dengan titik query. Klasifikasi menggunakan voting terbanyak diantara klasifikasi dari k obyek. algoritma KNN menggunakan klasifikasi ketetanggaan sebagai nilai prediksi dari query instance yang baru.

Algoritma metode KNN sangatlah sederhana, bekerja berdasarkan jarak terpendek dari query instance ke *training sample* untuk menentukan KNN-nya. Training sample diproyeksikan ke ruang berdimensi banyak, dimana masing-masing dimensi merepresentasikan fitur dari data. Ruang ini dibagi menjadi bagian-bagian berdasarkan klasifikasi *training sample*. Pada *fase training*, algoritma ini hanya melakukan penyimpanan vektor-vektor fitur dan klasifikasi data training sample. Pada fase klasifikasi, fitur-fitur yang sama dihitung untuk testing data (yang klasifikasinya tidak diketahui). Jarak dari vektor baru yang ini terhadap seluruh vektor training sample dihitung dan sejumlah k buah yang paling dekat diambil. Titik yang baru klasifikasinya diprediksikan termasuk pada klasifikasi terbanyak dari titik-titik tersebut.

Nilai k yang terbaik untuk algoritma ini tergantung pada data *training*. k dipilih dengan nilai ganjil akan mendapatkan hasil yang lebih baik dibandingkan dengan k dengan nilai genap, karena akan relevan terhadap kasus, dengan kategori yang sama nilai jumlahnya. Nilai k yang tinggi akan mengurangi efek noise pada klasifikasi, tetapi membuat batasan antara setiap klasifikasi menjadi semakin kabur. Ketepatan algoritma KNN sangat dipengaruhi oleh ada atau tidaknya fitur-fitur yang tidak relevan atau jika bobot fitur tersebut tidak setara dengan relevansinya terhadap klasifikasi. Riset terhadap algoritma ini sebagian besar membahas bagaimana memilih dan memberi bobot terhadap fitur agar performa klasifikasi menjadi lebih baik.

KNN memiliki beberapa kelebihan yaitu ketangguhan terhadap training data yang memiliki banyak noise dan efektif apabila training data-nya besar. Sedangkan, kelemahan KNN adalah KNN perlu menentukan nilai dari parameter k (jumlah dari tetangga terdekat), training berdasarkan jarak tidak jelas mengenai jenis jarak apa yang harus digunakan dan atribut mana yang harus digunakan untuk mendapatkan hasil terbaik, dan biaya komputasi cukup tinggi karena diperlukan perhitungan jarak dari tiap *query instance* pada keseluruhan *training sample*.

Adapun algortima k-NN dapat digambarkan dalam diagram alir K-*Nearest Neighbor* sebagai berikut.



Gambar 2.2 Diagram Alur K-Nearest Neighbor

1. Hitung jarak antara data sampel (data uji) dengan data latih yang telah dibangun. Salah satu persamaan dalam menghitung jarak kedekatan dapat menggunakan persamaan 2.3 *Cosine Similarity*.
2. Menentukan parameter nilai k = jumlah tetanggaan terdekat bebas.
3. Mengurutkan jarak terkecil dari data uji
4. Pasangkan kategori sesuai dengan kesesuaian
5. Cari jumlah terbanyak dari tetanggaan terdekat kemudian tetapkan kategori.

Jarak yang digunakan dalam penelitian ini adalah *Cosine Similarity*.

$$CosSim(i,k) = \frac{\sum_k (d_i d_k)}{\sqrt{\sum_k d^2{}_{ik}}\sqrt{\sum_k d^2{}_{jk}}} ..... \quad (2.3)$$

Keterangan :

$\sum_k (d_i d_k)$: Vector dot produk dari i dan k

$\sqrt{\sum_k d^2{}_{ik}}$: Panjang vector i

$\sqrt{\sum_k d^2{}_{jk}}$: Panjang vector k

i : data uji ke-i

k : data latih ke-j

2.7 Pemrograman Beorientasi Objek

Pemrogramman berorientasi objek pertama kali muncul pada pertengahan 60-an dengan bahasa pemgrogramman yang disebut Simula dan terus berkembang higga pada tahun 2002 microsoft memperkenalkan bahasa pemgoramman yaitu C# yang benar-benar merupakan bahasa pemrograman berorientasi objek. Pemrograman berorientasi objek adalah sebuah pendekatan yang berdasarkan kepada interaksi antar objek dalam menyelesaikan tugas [12]. Terdapat karakteristrik pada pemrograman berorientasi objek yaitu:

1. Objects

Dalam pemrograman berorientasi objek, sebuah objek adalah struktur untuk menggabungkan data dan prosedur untuk bekerja dengan data tersebut.

2. *Abstraction*

   Saat objek berinteraksi dengan objek lain perlu diperhatikan mengenai bagian dari property objek tersebut. Sebagai contoh abstraksi jika suatu objek berinteraksi dengan 2 objek lain di mana 2 objek tersebut memiliki atribut yang sama dan nilainya berbeda maka akan berbeda pula hasil interaksi tersebut.

3. *Encapsulation*

   Enkapsulasi yaitu proses di mana transaksi dengan sebuah data dilakukan secara tidak langsung, contohnya jika kita ingin berinteraksi data mahasiswa maka kita harus berinteraksi melalui objek mahasiswa tersebut, jika kita ingin melihat data mahasiswa maka kita mengirimkan pesan kepada objek mahasiswa lalu objek tersebut membaca pesan kita dan memberikan respon. Hal ini membuat program lebih mudah dan menyederhanakan proses debugging.

4. *Polymorphism*

   Dengan *polymorphism,* 2 objek yang berbeda dapat menerima pesan yang sama namun dengan aksi yang berbeda. Dalam pemrograman berorientasi objek kita dapat menerapkan jenis *polymorphism* yang bernama *overloading* yaitu penerapan metode atau aksi yang berbeda pada objek yang memiliki nama yang sama. Objek tersebut dapat membedakan dari objek yang memiliki nama yang sama tersebut mana objek yang harus dijalankan seperti membedakan berdasarkan parameter.

5. *Inheritance*

   Sebuah pewarisan fungsi dan karakteristik dari objek yang umum pada objek yang lebih spesifik seperti objek orangtua ke objek anak sehingga objek anak tersebut mewarisi fungsi dan karakteristik yang ada pada objek orangtua dan menggunakannya. Contohnya yaitu objek bangun datar sebagai objek orangtua yang memiliki fungsi luas dan keliling, dan objek segitiga menjadi objek anak yang mempunyai karakteristik alas dan tinggi, sehingga dengan *inheritance* objek segitiga dapat memakai fungsi luas dan keliling.

6. *Agregation*

   Agregasi yaitu ketika sebuah objek merupakan hasil dari gabungan objek-objek lain yang bekerja sama.

2.8 UML (*Unified Modeling Language*)

UML merupakan suatu alat dalam memodelkan analisis dan desain berorientasi objek. UML adalah sebuah bahasa pemodelan standar untuk perangkat lunak dan pengembangan sistem. Desain sistem dengan skala besar merupakan hal yang sulit, bagaimana pengembang perangkat lunak dan timnya menjaga komponen yang dibutuhkan, bagaimana membagi komponen-komponen kedalam beberapa bagian agar dapat dikerjakan bersama secara tim dan terdapat pemahaman yang sabar dalam pembangunan sistem tersebut sehingga detail pekerjaan tidak disalahartikan. Agar desain model lebih efektif maka diperlukan bahasa pemodelan yang dapat menjelaskan kompleksitas tersebut yang disebut UML. Berikut adalah beberapa diagram pada UML beserta apa yang dimodelkannya [13].

1. Use Case Diagram

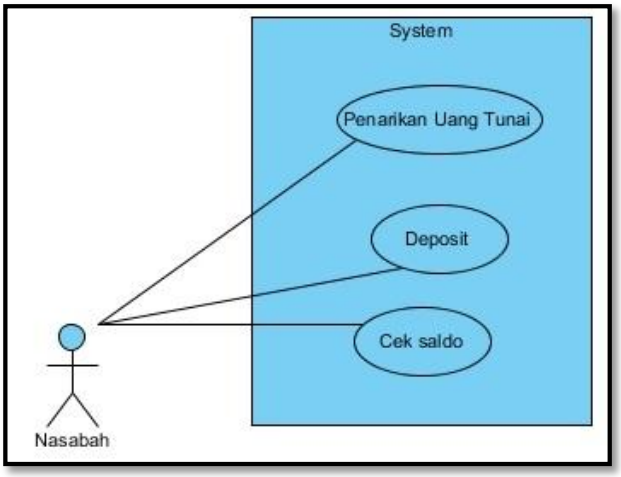
Use Case Diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Di dalam *use case diagram* ini sendiri lebih ditekankan kepada apa yang diperbuat sistem dan bagaimana sebuah sistem itu bekerja. Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan bentuk dari sebuah pekerjaan tertentu, misalnya *login* ke dalam sistem, *posting*, dan sebagainya, sedangkan seorang aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu [13]. Adapun komponen-komponen dalam use case diagram diantaranya:

a. Aktor

Aktor merupakan suatu entitas yang berkaitan dengan sistem tapi bukan dari bagian dalam sistem itu sendiri. Aktor berada diluar sistem namun berkaitan erat dengan fungsionalitas didalamnya. Aktor dapat memiliki hubungan secara langsung terhadap fungsi utama baik terhadap salah satu atau semua fungsionalitas utama. Aktor juga dapat dibagi terhadap berbagai jenis atau tingkatan dengan cara digeneralisasi atau dispesifikasi tergantung kebutuhan sistemnya. Aktor biasanya dapat berupa pengguna atau database yang secara pandang berada dalam suatu ruang lingkup sistem tersebut.

b. Use Case

Use case merupakan gambaran umum dari fungsi atau proses utama yang menggambarkan tentang salah satu perilaku sistem. Perilaku sistem ini terdefinisi dari proses bisnis sistem yang akan dimodelkan. Tidak semua proses bisnis digambarkan secara fungsional pada use case, tetapi yang digambarkan hanya fungsionalitas utama yang berkaitan dengan sistem. Use case menitik beratkan bagaimana suatu sistem dapat berinteraksi baik antar sistem maupun diluar sistem. Contoh *use case diagram* dapat dilihat pada gambar 2.3 berikut.



Gambar 2.3 Contoh *Use Case Diagram*

2. Use Case Scenario

Use case description digunakan untuk menunjukkan use case dan actor, mungkin menjadi titik awal yang bagus tapi ini tidak memberikan cukup detail untuk desainer sistem agar benarbenar memahami persis bagaimana sistem akan dipenuhi [13].

Tidak ada aturan baku untuk apa yang masuk ke dalam *use case description* menurut UML, tetapi beberapa contoh jenis informasi yang ditampilkan dalam tabel 2.1

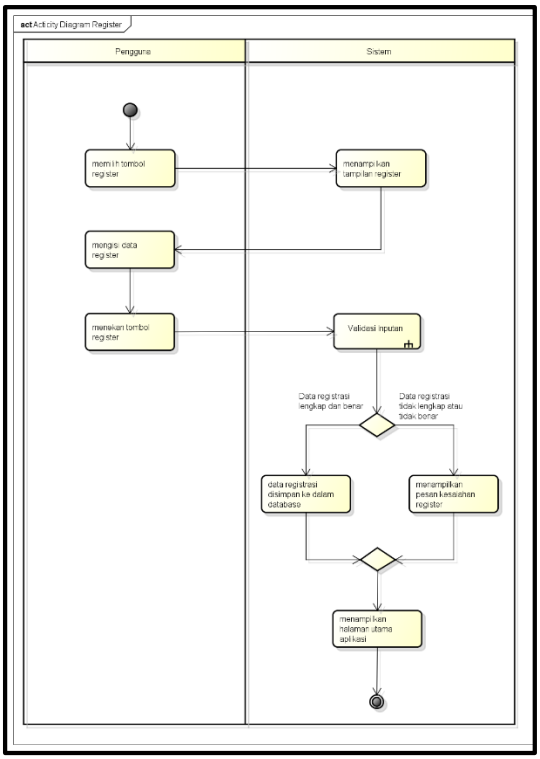
Tabel 2.1 Contoh *Use Case Scenario*

Use case name	**Create a new Blog Account**
Related Requirements	Requirement A.1.

Goal In Context	A new or existing author requests a new blog account from the Administrator.	
Preconditions	The system is limited to recognized authors and so the author needs to have appropriate proof of identity.	
Successful End Condition	A new blog account is created for the author.	
Failed End Condition	The application for a new blog account is rejected.	
Primary Actors	Adminidtrator	
Secondary Actors	Author Credentials Database.	
Trigger	The Administrator asks the CMS to create a new blog account.	
Main Flow	Step	Action
	1	The Administrator asks the system to create a new blog account.
	2	The Administrator selects an account type.
	3	The Administrator enters the author's details.
	4	The author's details are verified using the Author Credentials Database.
	5	The new blog account is created.
	6	A summary of the new blog account's details are emailed to the author.
Extensions	Step	Branching Action
	4.1	The Author Credentials Database does not verify the author's details.
	4.2	The author's new blog account application is rejected.

3. Activity Diagram

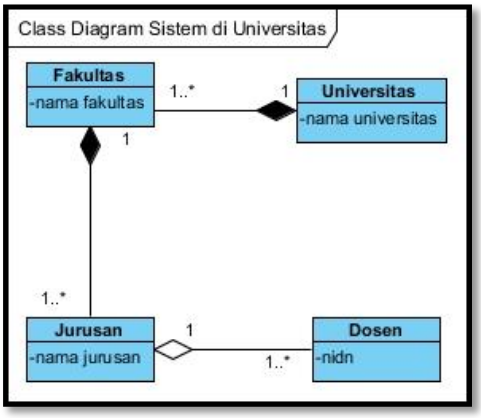
Activity Diagram adalah sebuah tahapan yang lebih fokus kepada menggambarkan proses bisnis dan urutan aktivitas dalam sebuah proses. Di mana biasanya dipakai pada *business modeling* untuk memperlihatkan urutan aktifitas proses bisnis. *Activity Diagram* ini sendiri memiliki struktur diagram yang mirip flowchart atau data flow diagram pada perancangan terstruktur. *Activity Diagram* dibuat berdasarkan sebuah atau beberapa *use case* pada *use case diagram*. Contoh *activity diagram* dapat dilihat pada gambar 2.4 berikut.



Gambar 2.4 Contoh Activity Diagram

4. Class Diagram

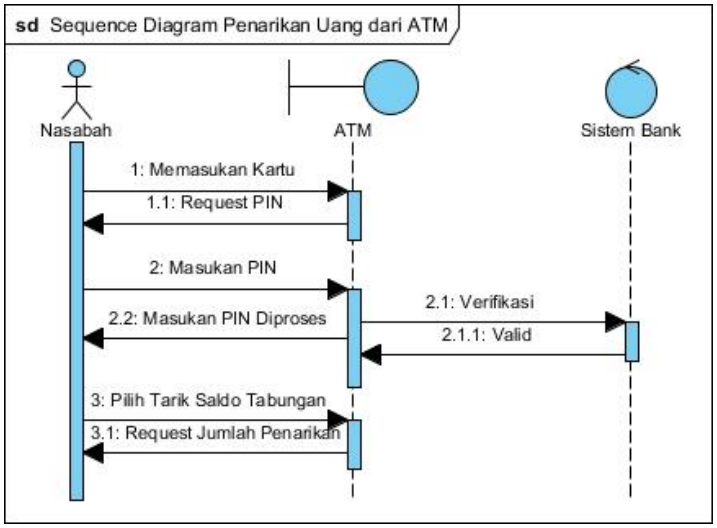
Class Diagram adalah sebuah *class* yang menggambarkan struktur dan penjelasan *class*, paket, dan objek serta hubungan satu sama lain. *Class* diagram juga menjelaskan hubungan antar *class* secara keseluruhan di dalam sebuah sistem yang sedang dibuat dan bagaimana caranya agar mereka saling berkolaborasi untuk mencapai sebuah tujuan [13]. Contoh *class diagram* dapat dilihat pada berikut.



Gambar 2.5 Contoh *Class Diagram*

5. Sequence Diagram

Sequence Diagram digunakan untuk menggambarkan perilaku pada sebuah skenario. Diagram jenis ini memberikan kejelasan sejumlah objek dan pesan-pesan yang diletakkan diantaranya di dalam sebuah *use case*. Komponen utamanya adalah objek yang digambarkan dengan kotak segi empat atau bulat, *message* yang digambarkan dengan garis penuh, dan waktu yang ditunjukkan dengan *progress vertical*. Manfaat dari *sequence diagram* adalah memberikan gambaran detail dari setiap *use case diagram* yang dibuat sebelumnya [13]. Contoh *sequence diagram* dapat dilihat pada berikut.



Gambar 2.6 Contoh Sequence Diagram

1.9 Pengujian

Pengujian yang dilakukan pada penelitian ini menggunakan precision, recall, dan F-measure.

1. Precision

Precision bersama *recall* merupakan salah satu pengujian dasar dan paling sering digunakan dalam penentuan efektifitas *information retrival system* maupun recommendation system. *True positive* (tp) pada information retrival merupakan item relevan yang dihasilkan oleh sistem. Sedangkan *false positive* (fp) merupakan semua item yang dihasilkan oleh sistem. Sehingga dalam *information retrival*, *precision* dihitung dengan persamaan 2.4 berikut [14].

$$\text{Precision} = \frac{tp}{tp+fp} = \frac{relevant\ item\ retrieved}{retrieved\ item} \quad (2.4)$$

Istilah *positive* dan *negative* mengacu pada prediksi yang dilakukan oleh sistem. Sedangkan istilah *true* dan *false* mengacu pada prediksi yang dilakukan oleh pihak luar atau pihak yang melakukan observasi. Pembagian kondisi tersebut dapat dilihat pada Tabel 2.2

Tabel 2.2 Pembagian Kondisi Hasil

	Relevant	**Non relevant**
Retrieved	True positif (tp)	False positive (fp)
Not retrieved	False negatif (fn)	True negatif (tn)

2. Recall

Recall digunakan sebagai ukuran dokumen yang relevan yang dihasilkan oleh sistem. False negative (fn) merupakan semua item relevan yang tidak dihasilkan oleh sistem. Dalam evaluasi information retrival system, recall dihitung dengan persamaan 2.5 [14]

$$\text{Recall} = \frac{tp}{tp+fn} = \frac{relevant\ item\ retrieved}{relevant\ item} \quad (2.5)$$

3. F-measure

F-measure merupakan nilai tunggal hasil kombinasi antara nilai precision dan nilai recall. F-measure dapat digunakan untuk mengukur kinerja dari recommendation system ataupun information retrival system. Karena merupakan rata-rata harmonis dari precision dan recall, F-measure dapat memberikan penilaian kinerja yang lebih seimbang. Persamaan (2.6) merupakan persamaan untuk menghitung F-measure [14]

$$\text{F-measure} = 2 \cdot \frac{precision \cdot recall}{precision+recall} \quad (2.6)$$