BAB 2

LANDASAN TEORI

2.1 Proposal Skripsi

Skripsi adalah istilah yang digunakan di Indonesia untuk mengilustrasikan suatu karya tulis ilmiah berupa paparan tulisan hasil penelitian sarjana S1 yang membahas suatu permasalahan/fenomena dalam bidang ilmu tertentu dengan menggunakan kaidah-kaidah yang berlaku[1]. Proposal skripsi sendiri merupakan Suatu permintaan, pengajuan ide kepada pihak lain untuk mendapatkan perizinan topik, tempat penelitian, dan lain sebagainya yang diajukan oleh seorang mahasiswa dalam usahanya untuk melakukan skripsi sebagai syarat kelulusannya sebagai sarjana.

Fungsi porposal skripsi penting untuk seorang mahasiswa tingkat akhir yang akan melakukan penelitian sebagai syarat kelulusannya, antara lain:

1. Untuk mengajukan persetujuan topik tugas akhir.
2. Untuk mengajukan tempat penelitian.
3. Untuk mendapatkan perizinan yang menunjang penelitian seperti akses data, melakukan observasi, melakukan wawancara, dan lain sebagainya di tempat penelitian.

Dalam menyusun proposal skripsi harus memperhatikan sistematika penulisannya, sistematika penulisan sendiri disusun untuk memberikan gambaran umum tentang penelitian yang akan dilakukan. Sistematika penulisan proposal tergantung pada jenis penelitiannya, tapi pada umumnya isi dari sistematika penulisan seperti dibawah ini:

1. Halaman Judul

   Berisi judul dari proposal skripsi, logo universitas, dan keterangan fakultas, program studi dan nama universitas beserta tahun penyusunan.
2. Daftar Isi

   Berisi daftar menu dan sub menu yang ada di proposal skipsi.

3. Daftar Tabel

   Berisi daftar tabel yang digunakan pada proposal skripsi.

4. Daftar Gambar

   Berisi daftar gambar yang terdapat pada proposal skripsiDaftar Lampiran Berisi daftar data penunjang penelitian.

5. Bab I Pendahuluan

   Berisi latar belakang, rumusan masalah, maksud dan tujuan, batasan masalah, metodologi penelitian dan sistematika penulisan.

6. Bab II Landasan Teori

   Berisi pembahasan seluruh buku – buku, jurnal ilmiah, dan ataupun sumber lain yang menjadi materi pembangun penelitian.

7. Bab III Metode Penelitian

   Berisi pembahasan metode penelitian yang digunakan pada penelitian yang akan dilakukan, meliputi metode pengumpulan data dan metode analisis data,

8. Daftar Pustaka

   Berisi seluruh kepustakaan yang digunakan dalam penyusunan proposal. Dicantumkan dibagian ini kemudian disusun secara alfabet dari A – Z.

2.2 *Text Mining*

Text Mining merupakan penerapan konsep dan teknik data *mining* untuk mencari pola dalam teks, proses penganalisisan teks guna menyarikan informasi yang bermanfaat untuk tujuan tertentu. Proses data mining untuk data dokumen atau teks memerlukan lebih banyak tahapan, mengingat data teks memiliki karakteristik yang lebih kompleks dari pada data biasa. Tujuannya yakni mencari kata – kata yang dapat mewakili isi dari dokumen sehingga dapat dilakukan analisa keterhubungan antar dokumen diproses[7]. Langkah – langkah yang dilakukan pada *text mining* pada penelitian ini antara lain t*ext preprocessing*, *term weighting*, dan klasifikasi.

2.2.1 *Text Preprocessing*

Pada Tahap *text preprocessing* dilakukan beberapa proses menyiapkan judul dan deksripsi proposal skripsi agar menjadi dokumen teks yang siap diolah ditahap selanjutnya, umumnya pada tahap ini terdapat beberapa proses , diantaranya: *tokenizing, filtering, stemming.*

2.2.1.1 *Tokenizing*

Tahap *tokenizing* adalah tahap pemotongan *string input* berdasarkan tiap kata yang menyusunnya. Setiap kalimat yang ada di dokumen akan disusun menjadi kata per kata, semua tanda baca dan tanda hubung akan dihapuskan. Contoh dari tahap ini adalah sebagai berikut:

Teks *Input*:

Teknik Informatika adalah sebuah program studi di unikom.

Hasil Token:

Teknik
Informatika
Adalah
Sebuah
Program
Studi
Di
Unikom

2.2.1.2 *Filtering*

Tahap *Filtering* adalah tahap mengambil kata – kata penting dari hasil token, dalam proses ini bisa menggunakan algoritma *stop list* (membuang kata yang kurang penting) atau *word list*(menyimpan kata penting). *stop list* adalah daftar kata yang sering digunakan dan tidak menjelaskan isi dari dokumen atau *stopword.* Contoh dari tahap ini adalah sebagai berikut:

Hasil Token:

Teknik unikom Informatika Adalah Sebuah Program Studi Di

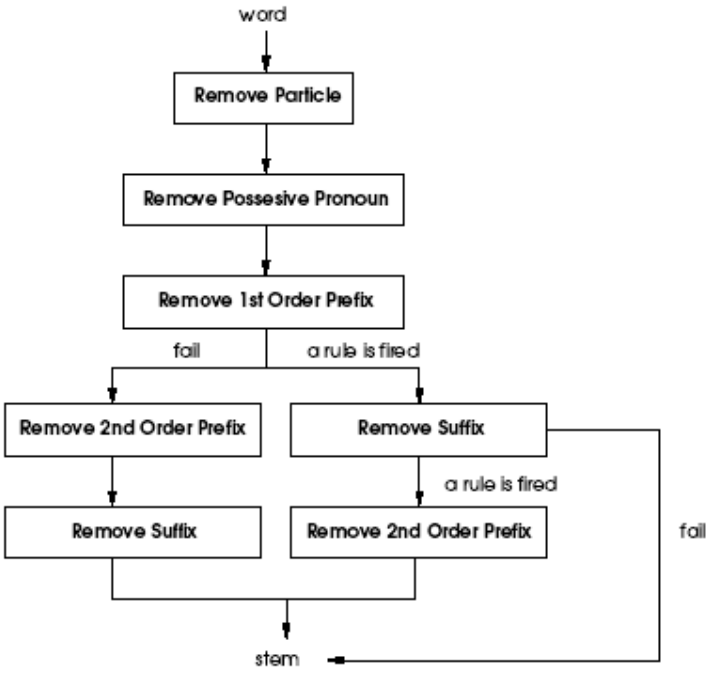
Hasil *Filter*:

Teknik Informatika Program Studi Unikom

2.2.1.3 *Stemming*

Tahap *Stemming* adalah tahap dimana kata hasil filtering akan dirubah kedalam bentuk dasarnya(*Stem*). Untuk kata dalam bahasa inggris, metode *stemming* yang paling populer adalah menggunakan algoritma Martin porter's *stammer* atau biasa disebut *porter stemmer.* Algoritma ini juga bisa diaplikasikan untuk bahasa selain inggris, dalam hal ini akan diterapkan pada bahasa indonesia juga.

Porter stemmer for Bahasa Indonesia dikembangkan oleh Fasillah Z. Tala pada tahun 2003. Implementasi *porter stemmer for* Bahasa Indonesia berdasarkan *English Porter Stemmer* yang dikembangkan oleh W.B Frake pada tahun 1992[8]. Algoritma Porter Stemmer untuk bahasa indonesia dapat dilihat pada gambar 2.1.



Gambar 2.1 Algoritma *Porter Stemmer*[8]

Adapun langkah – langkah pada algortima porter[9] adalah sebagai berikut:

1. Hapus *Particle*,
2. Hapus Possesive Pronoun.
3. Hapus awalan pertama. Jika tidak ada lanjutkan ke langkah 4a, jika ada cari maka lanjutkan ke langkah 4b.
4. a. Hapus awalan kedua, lanjutkan ke langkah 5a.
   b.Hapus akhiran, jika tidak ditemukan maka kata tersebut diasumsikan sebagai *root word.* Jika ditemukan maka lanjutkan ke langkah 5b.
5. a. Hapus akhiran. Kemudian kata akhir diasumsikan sebagai *root word*
   b.Hapus awalan kedua. Kemudian kata akhir diasumsikan sebagai *root word*.

Algoritma poter stemmer untuk bahasa indonesia ini mempunyai 5 kelompok aturan[9]. Aturan tersebut dapat dilihat pada tabel 2.1 sampai Tabel 2.5.

Tabel 2.1 Aturan untuk *Inflectional Particel*

Akhiran	Replacement	Measure Condition	Additional Condition	Contoh
-kah	NULL	2	NULL	Bukukah
-lah	NULL	2	NULL	Pergilah
-pun	NULL	2	NULL	Bukupun

Tabel 2.2 Aturan untuk *inflectional possesive pronoun*

Akhiran	Replacement	Measure Condition	Additional Condition	Contoh
-ku	NULL	2	NULL	Bukuku
-mu	NULL	2	NULL	Bukumu
-nya	NULL	2	NULL	Bukunya

Tabel 2.3 Aturan untuk *first order derivational prefix*

Awalan	Replacement	Measure Condition	Additional Condition	Contoh
Meng-	NULL	2	NULL	Mengukur > ukur
Meny-	S	2	V...	Menyapu > sapu
Men-	NULL	2	NULL	Menduga > duga
Mem-	P	2	V...	Memaksa > Paksa
Mem-	NULL	2	NULL	Membaca > baca
Me-	NULL	2	NULL	Merusak > rusak
Peng-	NULL	2	NULL	Pengukur > ukur
Peny-	S	2	V...	Penyapu > sapu
Pen-	NULL	2	NULL	Penduga > duga
Pem-	P	2	V...	Pemaksa > paksa
Pem-	NULL	2	NULL	Pembaca > baca
di-	NULL	2	NULL	Diukur > ukur
Ter-	NULL	2	NULL	Tersapu > sapu
Ke-	NULL	2	NULL	Kekasih > kasih

Tabel 2.4 Aturan untuk *second order derivational prefix*

Awalan	Replacement	Measure Condition	Additional Condition	Contoh
Ber-	NULL	2	NULL	Berlari > lari
Bel-	NULL	2	Ajar	Belajar > ajar
Be-	NULL	2	Ker	Bekerja > kerja
Per-	NULL	2	NULL	Perjelas > jelas
Pel-	NULL	2	Ajar	Pelajar > ajar
Pe-	NULL	2	NULL	Pekerja > kerja

Tabel 2.5 Aturan untuk *derivational suffix*

Akhiran	Replacement	Measure Condition	Additional Condition	Contoh
-kan	NULL	2	Prefix bukan anggota {ke, peng}	Tarikkan > tarik, Mengambilkan > ambil
-an	NULL	2	Prefix bukan anggota {di, meng, ter}	Makanan > makan, Perjanjian > janji.
-i	NULL	2	Prefix bukan anggota {ber, ke, peng}	Tandai > tanda, mendapati > dapat.

Contoh umum dari tahap ini adalah sebagai berikut:

Hasil *Filter*:

Membacakan
Membersihkan
Dikurangi
Pandangan
mining
text

Hasil *Stemming*:

Baca
Bersih
Kurang
Pandang
Mine
Text

2.3 *Term Weighting*

Tahap *Term weighting* adalah tahap pembobotan kata yang dihasilkan dari proses *stemming.* Metode yang sering digunakan yaitu menggunakan metode TF-IDF (*Term Frequency - Inverse Document Frequency*). Pada metode ini, perhitungan bobot *term*(kata) dalam sebuah dokumen dilakukan dengan mengalikan nilai *Term Frequency* dengan *Inverse Document Frequency*.

Term Frequency (TF) adalah faktor yang menentukan bobot *term* pada suatu dokumen berdasarkan jumlah kemunculannya dalam dokumen tersebut. Nilai jumlah kemunculan suatu kata (*term frequency*) diperhitungkan dalam pemberian bobot terhadap suatu kata. Semakin besar jumlah kemunculan suatu term dalam dokumen, semakin besar pula bobotnya dalam dokumen atau akan memberikan nilai kesesuian yang semakin besar.

Inverse Document Frequency (IDF) adalah pengurangan dominansi *term* yang sering muncul di berbagai dokumen. Hal ini diperlukan karena term yang banyak muncul di berbagai dokumen, dapat dianggap sebagai term umum (*common term*) sehingga tidak penting nilainya. Sebaliknya faktor kejarangmunculan kata (*term scarcity*) dalam koleksi dokumen harus diperhatikan dalam pemberian bobot. Kata yang muncul pada sedikit dokumen harus dipandang sebagai kata yang lebih penting (*uncommon term*) daripada kata yang muncul pada banyak dokumen. Pembobotan akan memperhitungkan faktor kebalikan frekuensi dokumen yang mengandung suatu kata (*inverse document frequency*). Hal ini merupakan usulan dari George Zipf. Zipf mengamati bahwa frekuensi dari sesuatu cenderung kebalikan secara proposional dengan urutannya[2].

Untuk menghitung bobot kata (Wi) dalam dokumen dihitung dengan menggunakan rumus:

$$w_i = TF(t_i, d) \times IDF(t_i) \quad (2.1)$$

Dimana:

w_i = bobot kata *(term)* dalam dokumen d.

i = indeks kata (*term).*

$TF(t_i, d)$= banyaknya kata (*term*) t_i yang muncul dalam dokumen d.

$IDF(t_i)$ = *inverse document frequency* dari kata (*term*) t_i.

Untuk menghitung $IDF(t_i)$ dapat menggunakan rumus:

$$IDF(t_i) = \log\left(\frac{D}{DF(t_i)}\right) \quad (2.2)$$

Dimana:

$IDF(t_i)$ = *inverse document frequency* dari kata (*term*) t_i.

D = jumlah dokumen keseluruhan.

$DF(t_i)$ = jumlah dokumen yang memiliki kata (*term*) (t_i).

2.4 Logika *Fuzzy*

Dasar logika *fuzzy* adalah teori himpunan *fuzzy*. Pada teori himpunan *fuzzy*, peranan derajat keanggotaan sebagai penentu keberadaan elemen dalam suatu himpunan sangatlah penting. Nilai keanggotaan atau derajat keanggotaan atau *membership function* menjadi ciri utama dari penalaran dengan logika fuzzy tersebut.

Logika fuzzy digunakan untuk menerjemahkan suatu besaran yang diekspresikan menggunakan bahasa (*linguistik*). Misalnya besaran kecepatan laju kendaraan yang diekspresikan dengan pelan , agak cepat, cepat, dan sangat cepat[10]. *Fuzzy logic* pertama kali diperkenalkan oleh Jan Lukasiewicz pada tahun 1920-an sebagai teori kemungkinan. Logika kemungkinan ini memperluas jangkauan dari nilai kebenaran untuk semua bilangan riil pada interval antara 0 dan 1. Pada tahun 1965, profesor dan kepala departemen teknik elektrik di university of california, Berkeley, yaitu Lotfi Zadeh, menemukan kembali , mengidentifikasi, mengeksplorasi, mempromosikan, dan berjuang untuk *fuzzy logic*. Beliau memperluas ruang kerja teori kemungkinan menjadi sistem logika matematika formal dan konsep baru untuk mengaplikasikan istilah bahasa alami pada penelitiannya, yaitu '*Fuzzy sets*'. Logika baru ini dinamakan *fuzzy logic, fuzzy logic* banyak digunakan karena mirip dengan cara berfikir manusia karena dapat mempresentasikan pengetahuan manusia dalam bentuk matematis.

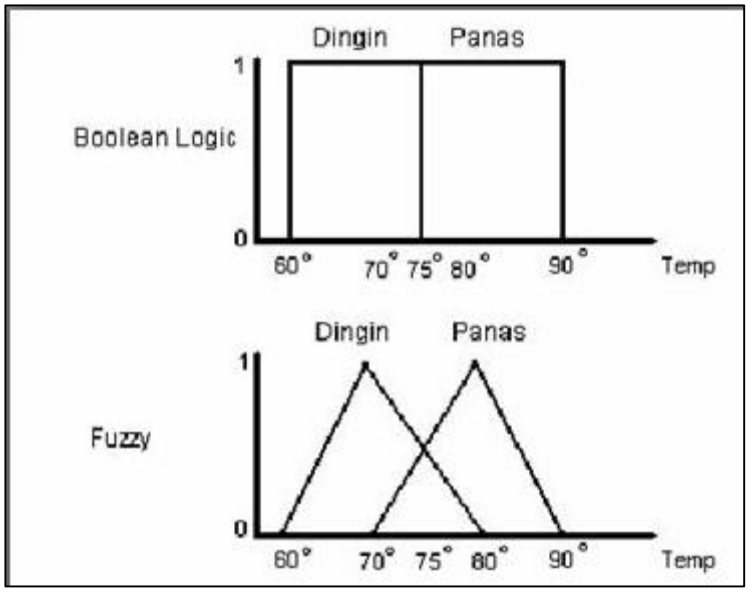
2.4.1 Konsep *Fuzzy Logic*

Fuzzy logic berkaitan dengan kebenaran proporsional, namun terkadang didunia nyata sering tidak didefinisikan secara jelas. Contohnya sulit untuk memberikan gambaran kebenaran "amin sudah tua" berbeda dengan menentukan nilai benar atau salah pada frasa "jika john berumur 60 tahun". Karena dalam beberapa kasus umur 60 tahun sudah cukup tua untuk memenuhi syarat mendapat keuntungan warga senior diberbagai segi. Tetapi, bila dilihat dari segi yang lain umur 60 tahun tidak cukup tua karena tidak memenuhi syarat jaminan sosial. Jadi, diperlukan nilai kebenaran untuk memahami frasa "amin sudah tua" guna mendapat nilai antara [0,1], tidak hanya 0 atau 1.

Fuzzy logic memiliki banyak nilai, tidak seperti logika boolean yang memiliki 2 nilai. *Fuzzy logic* menangani derajat keanggotaan dan derajat kebenaran. *Fuzzy logic* menangani nilai yang berkelanjutan dari 0(sepenuhnya salah) sampai 1(sepenuhnya benar).

2.4.1.1 Himpunan *Fuzzy*

Himpunan *fuzzy* merupakan kumpulan prinsip matematik sebagai penggambaran pengetahuan berdasarkan derajat keanggotaan. Sebuah himpunan *fuzzy* adalah himpunan yang memiliki elemen – elemen derajat keanggotaan yang bervariasi dalam himpunan. Ini berlawanan dengan himpunan klasik karena anggota dari sebuah himpunan klasik tidak mungkin menjadi anggota kecuali memiliki derajat keanggotaan penuh dalam himpunan. Untuk lebih jelasnya dapat dilihat pada gambar 2.2.

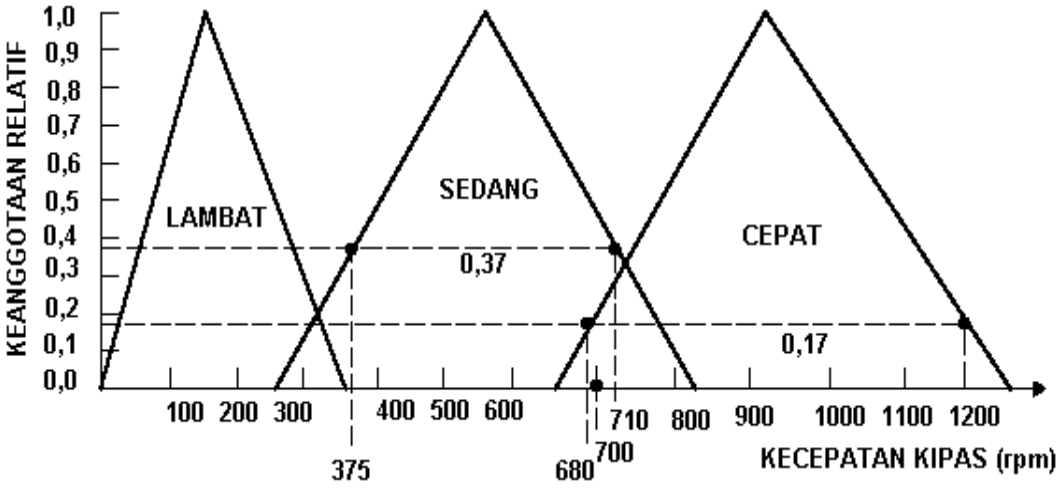


Gambar 2.2 Perbedaan *fuzzy logic* dengan *boolean logic*

2.4.1.2 Fungsi Keanggotaan

Tingkat keanggotaan $\mu_A(x)$ memetakan objek atau atributnya (x) ke bilangan riil positif pada interval [0,1]. Karena karakteristik pemetaannya seperti sebuah fungsi, maka disebut sebagai fungsi kenaggotaan. Definisi formalnya adalah:

"sebuah fungsi keanggotaan $\mu_A(x)$ dikarakteristikkan dengan pemetaan $\mu_A : (x) \rightarrow [0,1], x \in X$ dimana x sebuah bilangan riil yang mendeskripsikan sebuah objek atau atributnya dan X adalah semesta pembicaraan dan A adalah himpunan bagian dari x ". fungsi keanggotaan dari variabel linguistik dapat dilihat pada gambar 2.3.



Gambar 2.3 fungsi keanggotaan dari variabel linguistik kecepatan

fungsi keanggotaan adalah pemetaan sebuah elemen x pada semesta nilai keanggotaan menggunakan sebuah bentuk fungsi teoritis. Beberapa fungsi keanggotaan *fuzzy*:

1. Representasi Linear

   Pemetaan input ke derajat keanggotaannya digambarkan sebagai suatu garis lurus. Terdapat 2 bentuk yaitu representasi liner naik dan turun.

   a. Kurva representasi linear naik

      Himpunan dimulai pada nilai dominan yang memiliki derajat keanggotaan[0] dan bergerak ke kanan menuju domain dengan derajat keanggotaan yang lebih tinggi. Fungsi keanggotaannya adalah:

$$\mu(x) = \begin{cases} 0; & x \leq a \\ \frac{x-a}{b-a}; & a \leq x \leq b \\ 1; & x \geq b \end{cases} \qquad (2.3)$$

   b. Kurva representasi linear turun

      Himpunan dimulai pada nilai domain yang memiliki derajat keanggotaan [1] dan bergerak kekanan menuju domain dengan derajat keanggotaan yang lebih rendah. Fungsi keanggotaannya adalah:

$$\mu(x) = \begin{cases} \frac{b-x}{b-a}; & a \leq x \leq b \\ 0: & x \geq b \end{cases} \quad (2.4)$$

2. Representasi Kurva Segitiga

Representasi kurva segitiga merupakan gabungan antara 2 garis (linear). Fungsi keanggotaannya adalah:

$$\mu(x) = \begin{cases} 0; & x \leq a \; atau \; x \geq c \\ \frac{x-a}{b-a}; & a \leq x \leq b \\ \frac{c-x}{c-b}; & x \geq b \end{cases} \quad (2.5)$$

3. Representasi Kurva Trapesium

Representasi kurva trapesium menyerupai bentuk segitiga, namun memiliki beberapa titik dengan derajat keanggotaan 1. Fungsi keanggotaannya adalah:

$$\mu(x) = \begin{cases} 0; & x \leq a \; atau \; x \geq d \\ \frac{x-a}{b-a}; & a \leq x \leq b \\ 1: & b \leq x \leq c \\ \frac{d-x}{d-c}; & c \leq x \leq d \end{cases} \quad (2.6)$$

4. Representasi Kurva Betuk Bahu

Daerah yang terletak pada sisi kanan dan kiri yang tidak mengalami perubahan, digunakan untuk mengakhiri variabel suatu daerah *fuzzy*. Pada bahu kiri kurva bergerak dari benar ke salah, dan pada bahu kanan kurva bergerak dari salah ke benar.

2.5 Algoritma *K-Nearest Neighbour*

Klasifikasi menggunakan algoritma K-*nearest neighbour* adalah dengan menentukan jarak untuk mengukur seberapa dekat jarak kesamaan data pembelajaran dengan objek tersebut. Berdasarkan konsep "*learning by analogy*",data *learning* dideskripsikan dengan atribut numerik *n*-dimensi. Tiap data *learning* merepresentasikan sebuah titik yang ditandai dengan c, dalam ruang *n*-dimensi. Jika sebuah data *query* yang labelnya tidak diketahui diinputkan, maka K-*nearest neighbour* akan mencari *k* buah data *learning* yang berjarak paling dekat dengan data *query* dalam ruang *n*-dimensi. Jarak antara data *query* dengan data *learning* dihitung dengan cara mengukur jarak antara titik yang merepresentasikan

data *query* dengan semua titik yang merepresentasikan data *learning* dengan rumus *euclidean distance*.

Pada fase *training*, algoritma ini hanya akan melakukan penyimpanan vektor – vektor fitur dan kasifikasi data *training sample*. Pada fase klasifikasi, fitur – fitur yang sama dihitung untuk data *testing*(klasifikasinya belum diketahui). Jarak dari vektor yang baru 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; secara umumnya nilai *k* yang tinggi akan mengurangi efek *noise* pada klasifikasinya, tetapi membuat batasan antara setiap klasifikasi menjadi lebih kabut, nilai *k* yang bagus dapat dipilih dengan optimasi paramater, misalnya dengan menggunakan cross-validation. Kasus khusus dimana klasifikasi diprediksi berdasarkan data pembelajaran yang paling dekat (dengan kata lain, $k = 1$) disebut algoritma *nearest neighbour*.

Tepat atau tidaknya algoritma K-*nearest neighbour* dipengaruhi oleh ada atau tidaknya fitur – fitur yang tidak relevan atau jika bobot fitur tersebut tidak setara dengan relevansinya terhadap klasifikasi. Riset yang dilakukan terhadap algoritma ini sebagian besar membahas bagaimana memilih dan memberi bobot terhadap fitur, agar performa klasifikasi menjadi lebih baik.

K buah data *learning* terdekat akan melakukan *voting* untuk menentukan label mayoritas. Label data *query* akan ditentukan berdasarkan label mayoritas dan jika ada lebih dari satu label mayoritas maka label data query dapat dipilih secara acak di antara label – label mayoritas yang ada.

Untuk dapat menentukan hasil dari klasifikasi menggunakan algoritma k-*nearest neighbour*¸ hal pertama yang harus dilakukan adalah menentukan jarak untuk mengukur seberapa dekat jarak antara data *training* dengan data *testing*. Metode *euclidean distance* dapat digunakan untuk menentukan jarak tersebut, selain itu terdapat metode lain yang bisa digunakan pada kasus ini yakni dengan

menggunakan metode pengukuran kesamaan (*similarity*) yang digunakan untuk mengukur kemiripan antara data *training* dan data *testing*, nama metode tersebut adalah *cosine similarity*[2]. Rumusnya adalah sebagai berikut:

$$sim(q,d) = \frac{\sum_t W_{t,d} \,.\, W_{t,q}}{\sqrt{\sum_t W_{t,d}^2} \cdot \sqrt{\sum_t W_{t,q}^2}} \quad (2.7)$$

Dimana:

$sim(q,d)$ = kemiripan (*similarity*) antara data *testing* dan data *training.*

q = data *testing.*

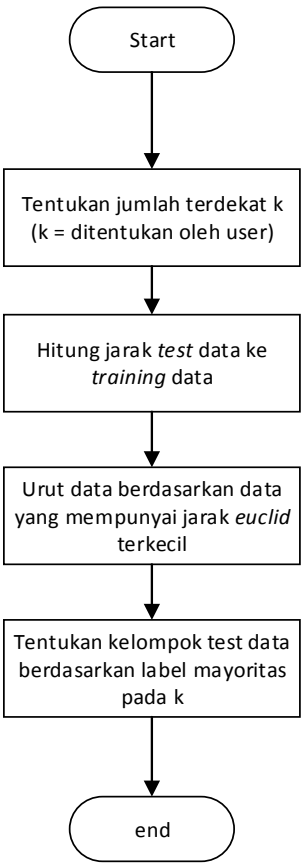
d = data *training.*

$W_{t,d}$ = nilai bobot *term* data *training*.

$W_{t,q}$ = nilai bobot *term* data *testing*.

t = indeks *term.*

Adapun algoritma K- *nearest neighbour* ditunjukan pada gambar 2.4 berikut:



Gambar 2.4 FlowChart Algoritma K-NN

2.6 *Fuzzy K- Nearest Neighbour*

Algoritma ini memberikan nilai membership sebagai fungsi pola jarak dari sejumlah himpunan K-*nearest neighbour* dan pemberian nilai keanggotaan *neighbour* pada kelas tertentu. Sehingga pada algoritma ini data *testing* yang akan diklasifikasikan akan memiliki nilai keanggotaan pada semua kelas. Klasifikasi algoritma ini nantinya akan memilih nilai keanggotaan kelas pada data *testing* yang paling tinggi[2]. Berikut merupakan persamaan untuk memberikan nilai keanggotaan pada data *testing x* dengan menggunkan metode *euclidean distance*. Jarak *eucidean distance* antara vektor data *testing xi* dengan vektor data *training xj* dihitung menggunakan rumus dibawah ini:

$$\|x - x_j\| = \left(\sum_{i=1}^{n} \left|N_i - N_j^i\right|^2\right)^{1/2} \quad (2.8)$$

Dimana :

$\|x - x_j\|$= jarak e*uclidean* vektor data *testing* dan dokumen *training* k-*j*.

Ni = bobot *term* dalam dokumen *testing*.

N_i^j = bobot *term* pada dokumen *training* ke-*j*.

i = index *term* ke-*i.*

n = jumlah *term* keseluruhan hasil *text preprocessing*.

Untuk penentuan nilai keanggotaan kelas ke-*i* pada tetangga ke-*j* dihitung menggunakan rumus dibawah ini:

$$\mu ij = \begin{cases} 1, & j = i \\ 0, & j \neq i \end{cases} \quad (2.9)$$

Dimana :μij = nilai keanggotaan kelas ke-*i* pada tetangga ke-*j*.

i = kelas *genre*.

j = Tetangga terdekat (K).

Penentuan nilai membership kelas pada data testing x dengan metode *euclidean distance* dihitung menggunakan rumus dibawah ini:

$$\mu(x) = \frac{\sum_{i=1}^{k} \mu_{ij} \left(\frac{1}{\|x - x_j\|^{2/(m-1)}} \right)}{\sum_{i=1}^{k} \left(\frac{1}{\|x - x_j\|^{2/(m-1)}} \right)} \quad (2.10)$$

Dimana:

$\mu(x)$ = nilai keanggotaan kelas ke-*i* pada data *testing x*.

m = bobot pangkat (*weight exponent*).

Jika jarak yang digunakan adalah Manhattan maka m = 1.

Jika jarak yang digunakan adalah Euclidean maka m = 2.

Jika jarak yang digunakan adalah Chebyshev m = ∞ maka.

Sedangkan untuk memberikan nilai keanggotaan pada data *testing* x, dengan menggunakan metode *cosine similiarity* dihitung dengan menggunakan rumus dibawah ini:

$$\mu(x) = \frac{\sum_{i=1}^{k} \mu_{ij}(sim(q,d))}{\sum_{i=1}^{k}(sim(q,d))} \qquad (2.11)$$

Dimana:

$\mu(x)$ = nilai keanggotaan kelas ke-*i* pada data *testing x*.

μ_{ij} = nilai keanggotaan kelas ke-*i* pada tetangga ke-*j* yang telah dihitung dengan menggunakan rumus 2.9.

$sim(q, d)$ = kemiripan (*similarity*) antara data *testing* q dengan data *training* d yang telah dihitung dengan menggunakan rumus 3.

2.7 Metode *Linear*

Model Sekuensial Linier atau sering disebut Model Pengembangan Air Terjun, merupakan paradigma model pengembangan perangkat lunak paling tua, dan paling banyak dipakai. Model ini mengusulkan sebuah pendekatan perkembangan perangkat lunak yang sistematik dan sekunsial yang dimulai pada tingkat dan kemajuan sistem pada seluruh tahapan analisis, desain , kode, dan pengujian.

1. Analisis

   Pada fase ini pengumpulan kebutuhan diintensifkan dan difokuskan pada sistem yang akan dibangun meliputi identifikasi domain informasi, tingkah laku sistem, unjuk kerja, dan antarmuka sistem.

2. Desain

   Fase ini difokuskan pada proses desain struktur data, arsitektur sistem, representasi interface, dan algoritma program.

3. Kode

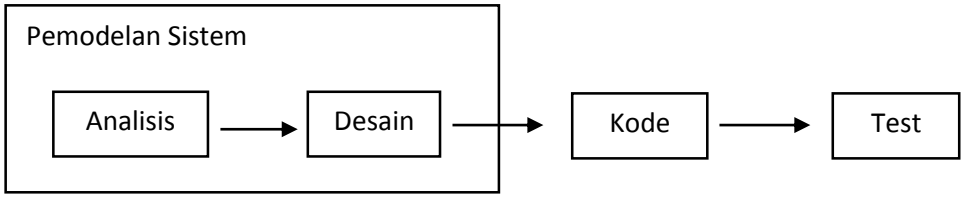
   Fase ini dilakukan setelah menyelesaikan proses desain, hasil yang ada di

4. proses desain akan diterjemahkan kedalam suatu bahasa pemrograman tertentu untuk menghasilkan sebuah sistem.

5. Test

   Pengujian dilakukan untuk menemukan kesalahan-kesalahan yang mungkin terjadi

6. pada proses pengkodean serta memastikan bahwa input yang dibatasi memberikan hasil yang sesuai dengan kebutuhan.



Gambar 2.5 Metode Pembangunan Perangkat Lunak[4]

Kelebihan:

1. Tahapan proses pengembangannya tetap (pasti), mudah diaplikasikan, dan prosesnya teratur.
2. Cocok digunakan untuk produk software/program yang sudah jelas kebutuhannya di awal, sehingga minim kesalahannya.
3. Software yang dikembangkan dengan metode ini biasanya menghasilkan kualitas yang baik.
4. Documen pengembangan sistem sangat terorganisir, karena setiap fase harus terselesaikan dengan lengkap sebelum melangkah ke fase berikutnya.

Kelemahan:

1. Proyek yang sebenarnya jarang mengikuti alur sekuensial seperti diusulkan, sehingga perubahan yang terjadi dapat menyebabkan hasil yang sudah

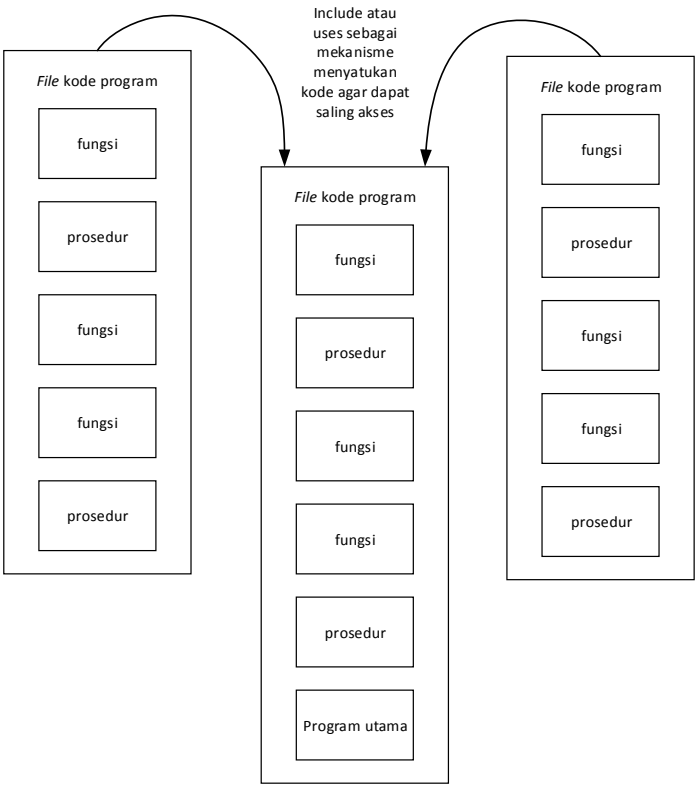
didapatkan tim pengembang harus diubah kembali/iterasi sering menyebabkan masalah baru.

2. Terjadinya pembagian proyek menjadi tahap-tahap yang tidak fleksibel, karena komitmen harus dilakukan pada tahap awal proses.
3. Sulit untuk mengalami perubahan kebutuhan yang diinginkan oleh customer/pelanggan.
4. Perubahan ditengah-tengah pengerjaan produk akan membuat bingung tim pengembang yang sedang membuat produk.
5. Adanya waktu kosong (menganggur) bagi pengembang, karena harus menunggu anggota tim proyek lainnya menuntaskan pekerjaannya.

2.8 Pemrograman Terstruktur

Pemrograman terstruktur adalah konsep atau paradigma atau sudut pandang pemrograman yang membagi-bagi program berdasarkan fungsi-fungsi atau prosedur-prosedur yang dibutuhkan program komputer. Modul-modul (pembagian program) biasanya dibuat dengan mengelompokkan fungsi-fungsi dan prosedur-prosedur yang diperlukan sebuah proses tertentu.

Fungsi-fungsi dan prosedur-prosedur ditulis secara sekuensial atau terurut dari atas ke bawah sesuai dengan kebergantungan antar fungsi atau prosedur(fungsi atau prosedur yang dapat dipakai oleh fungsi atau prosedur dibawahnya harus yang sudah ditulis atau dideklarasikan di atasnya)[5]. Berikut adalah contoh ilustrasi untuk pemrograman terstruktur:



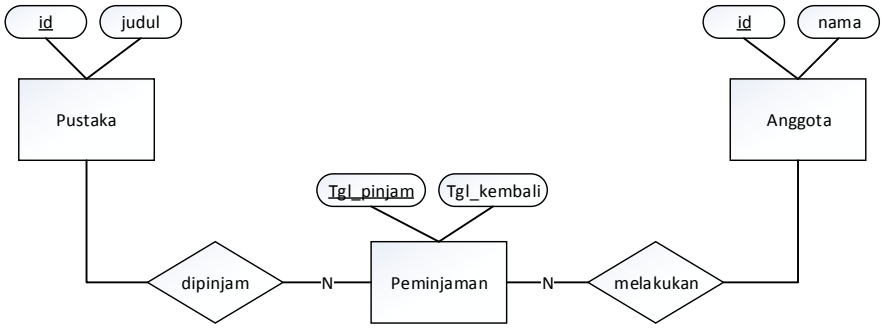
Gambar 2.6 ilustrasi pemrograman terstruktur[5]

Pemodulan pada pemrograman terstruktur dibagi berdasarkan fungsi-fungsi dan prosedur-prosedur. Oleh karena itu, pemodelan pada pemrograman terstruktur lebih fokus bagaimana memodelkan data dan fungsi-fungsi dan prosedur-prosedur yang harus dibuat. Jenis paradigma pemrograman yang digunakan dapat dideteksi dari bahasa pemrograman apa yang akan digunakan untuk membuat program, baru setelah itu ditentukan paradigma pemrograman apa yang akan digunakan.

2.8.1 Entity Relationship Diagram (ERD)

Pemodelan awal basis data yang paling banyak digunakan adalah menggunakan Entity Relationship Diagram (DFD). ERD dikembangkan berdasarkan teori himpunan dalam bidang

matematika. ERD digunakan untuk pemodelan basis data relasional. Sehingga jika penyimpanan basis data menggunakan OODBMS maka perancangan basis data tidak perlu menggunakan ERD. ERD memiliki beberapa aliran notasi seperti notasi Chen (dikembangkan oleh peter chen), Barker (dikembangkan oleh Richard Barker, Ian Parmer, Harry Ellis), notasi Crow's Foot, dan beberapa notasi lain. Namun yang banyak digunakan adalah notasi dari Chen. Berikut adalah contohnya:



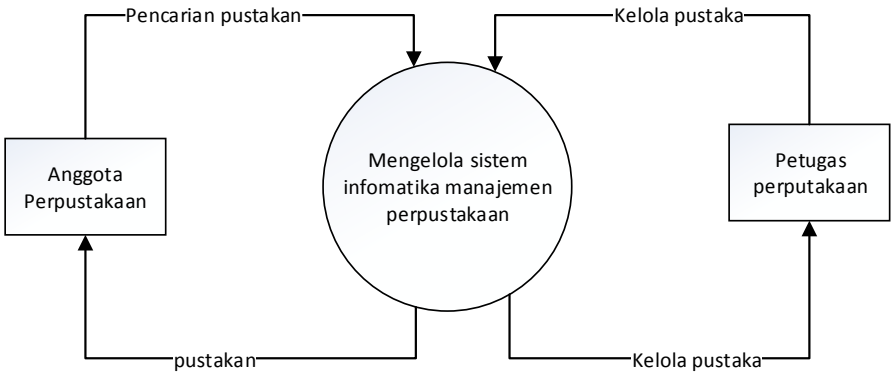
Gambar 2.7 ERD notasi Chen

2.8.2 Data Flow Diagram (DFD)

Informasi yang ada di dalam perangkat lunak dimodifikasi dengan beberapa transformasi yang dibutuhkan. *Data flow diagram* (DFD) atau dalam bahasa indonesia menjadi diagram alir data (DAD) adalah representasi grafik yang menggambarkan aliran informasi dan transformasi informasi yang diaplikasikan sebagai data yang mengalir dari masukan (*input*) dan keluaran (*output*)[5].

DFD dapat digunakan untuk merepresentasikan sebuah sistem atau perangkat lunak pada beberapa level abstraksi. DFD dapat dibagi menjadi beberapa level yang lebih detail. DFD menyediakan mekanisme untuk pemodelan fungsional ataupun pemodelan aliran informasi. Oleh karena itu, DFD lebih sesuai digunakan untuk memodelkan fungsi-fungsi perangkat lunak yang akan diimplementasikan menggunakan pemrograman terstruktur karena pemrograman terstruktur membagi-bagi bagiannya dengan fungsi-fungsi dan prosedur-prosedur[5].

DFD Edward Yourdon dan Tom DeMarco populer digunakan sebagai model analisis sistem perangkat lunak yang akan diimplementasikan dengan pemrograman terstruktur. Contohnya adalah sebagai berikut:



Gambar 2.8 DFD Edward Yourdon dan Tom DeMarco[5]

2.8.3 Kamus Data

Kamus data (data dictionary) dipergunakan untuk memperjelas aliran data yang digambarkan pada DFD. Kamus data adalah kumpulan daftar elemen data yang mengalir pada sistem perangkat lunak sehingga masukan (inputan) dan keluaran (output) dapat dipahami secara umum (memiliki standar secara penulisan). Kamus data dalam imlementasi program dapat menjadi parameter masukan dan keluaran dari sebuah fungsi atau prosedur.

2.9 C#

C# adalah bahasa pemrograman yang dikembangkan oleh perusahaan pengembang perangkat lunak besar dunia yakni microsoft. Bahasa pemrograman C# sudah distandarisasi dengan standar internasional oleh ECMA. Seperti halnya bahasa pemrograman lain bahasa pemrograman C# bisa digunakan untuk membangun berbagai macam jenis aplikasi, seperti aplikasi berbasis dekstop, aplikasi berbasis web, ataupun aplikasi berbasis web service.

Ada beberapa alasan seorang programmer memilih C# untuk digunakan dalam mengembangkan program, alasan – alasannya antara lain:

1. Sederhana(*Simple*)

   C# menghilangkan beberapa hal yang bersifat kompleks yang terdapat dalam beberapa macam bahasa pemrograman seperti java dan C++, termasuk diantaranya menghilangkan *macro, templates, multiple inheritance,* dan *virtual base classes*. Hal – hal tersebut yang dapat

menyebabkan kebingungan pada saat menggunakannya dan juga berpotensi dapat menjadi masalah bagi para programmer C++.

2. Modern

   Adanya beberapa fitur seperti *exception handling, garbage collection, extensible data types,* dan *code security* menjadikan bahasa C# menjadi bahasa pemrograman yang modern.

3. *Object Oriented Language*

   Kunci dari bahasa pemrograman yang bersifat *Object Oriented Language* adalah adanya *encapsulation, inheritance,* dan *polymorphism.* Dan sifat – sifat tersebut telah dimiliki oleh bahas C#. Sehingga C# merupakan bahasa pemrograman yang bersifat *Object Oriented Language*.

4. *Powerful* dan Fleksibel

   C# dapat digunakan untuk mebangun berbagai macam aplikasi, seperti aplikasi pengolahan kata, grafik, spreadsheets, atau bahkan membuat *compiler* untuk sebuah bahasa pemrograman.

5. Efisien

   C# adalah bahasa pemrograman yang menggunakan jumlah kata – kata yang tidak terlalu banyak. C# hanya berisi kata – kata yang disebut dengan *keyword. Keyword* ini digunakan untuk menjelaskan berbagai macam informasi. Jika kita berpikir bahwa dengan menggunakan banyak *keyword* kemudian akan membuat aplikasi lebih *powerful* itu tak selalu benar, karena justru akan membuat pengembang lain menjadi kesulitan saat membangun aplikasi.

6. Modular

   Kode C# ditulis dengan pembagian masing class – class yang terdiri dari beberapa *routines* yang disebut dengan *member methods*. Class – class dan metode – metode ini dapat digunakan kembali oleh program atau aplikasi lain.

7. C# akan menjadi populer

Dengan dukungan penuh dari microsoft yang akan mengeluarkan produk – produk utamanya dengan dukungan framework.NET, maka masa depan bahasa C# sebagai salah satu bahasa pemrograman yang ada di dalam lingkungan framework.NET akan lebih baik.

2.10 *Confusion Matrix*

Confusion matrix merupakan sebuah metode perhitungan yang digunakan untuk mencari keakuratan pada hasil klasifikasi. Berikut merupakan tabel *confusion matrix*:

Tabel 2.6 *Confusion Matrix*

Kelas		Prediksi	
		1	0
Kelas sebenarnya	1	TP	FN
	0	FP	TN

Keterangan:

a. *True Positive* (TP), merupakan jumlah dokumen dari kelas 1 yang benar diklasifikasikan sebagai kelas 1.
b. *False Positive* (FP), merupakan jumlah dokumen dari kelas 0 yang salah diklasifikasikan sebagai kelas 1.
c. *False Negative* (FN), merupakan jumlah dokumen dari kelas 1 yang salah diklasifikasikan sebagai kelas 0.
d. *True Negative* (TN), merupakan jumlah dokumen dari kelas 0 yang benar diklasifikasikan sebagai kelas 0. [12]

Kemudian akan dihitung akurasinya, akurasi merupakan perhitungan antara jumlah prediksi benar dan dibagikan dengan jumlah prediksi yang dilakukan yang mempunyai tujuan untuk mengetahui berapa presentase keberhasilan suatu sistem melakukan klasifikasi, untuk menghitung akurasi digunakan rumus sebagai berikut:

$$akurasi = \frac{jumlah\ prediksi\ benar}{jumlah\ seluruh\ prediksi\ yang\ dilakukan} \times 100\ \%$$