Hak cipta dan penggunaan kembali:

Lisensi ini mengizinkan setiap orang untuk menggubah, memperbaiki, dan membuat ciptaan turunan bukan untuk kepentingan komersial, selama anda mencantumkan nama penulis dan melisensikan ciptaan turunan dengan syarat yang serupa dengan ciptaan asli.

Copyright and reuse:

This license lets you remix, tweak, and build upon work non-commercially, as long as you credit the origin creator and license it on your new creations under the identical terms.

BAB II

TINJAUAN PUSTAKA

2.1 Landasan Teori

Pada penelitian ini, penulis membuat sebuah rancang bangun aplikasi *speech recognition* kamus geografi yang bernama Gaia pada *smartphone* berbasis Android. Pada penelitian terdahulu, Benedictus Riski Adiyanto [2] telah membuat sebuah aplikasi Android kamus geografi versi bahasa Indonesia. Hanya saja, aplikasi tersebut tidak menyertakan fitur *speech recognition* di dalamnya. Sementara Tony Wijaya dan Samuel Susanto [18] telah membuat aplikasi *speech recognition* bahasa Indonesia untuk Android. Namun, aplikasi tersebut terbatas hanya *speech to text* (suara ke teks). Selain itu, fonem yang digunakan adalah aturan fonem Inggris.

Rancang bangun aplikasi bernama Gaia yang dibuat oleh penulis kali ini merupakan aplikasi yang dibangun pada sistem operasi Android dengan menggunakan sebuah *smartphone* sebagai perangkat keras untuk menerima input berupa kata yang diucapkan oleh *user*. Proses *training* (pelatihan) kata menggunakan metode *Hidden Markov Model* (HMM) yang terdapat pada *library* Pocketsphinx. Fonem yang digunakan adalah aturan fonem Indonesia yang dikembangkan oleh Very Ferdiansyah [20]. Kelebihan dari aplikasi ini adalah dapat digunakan tanpa akses internet dan menyertakan fitur *speech recognition* di dalamnya sehingga diharapkan dapat membuat anak-anak Indonesia lebih tertarik untuk mempelajari istilah-istilah yang berkaitan dengan Geografi.

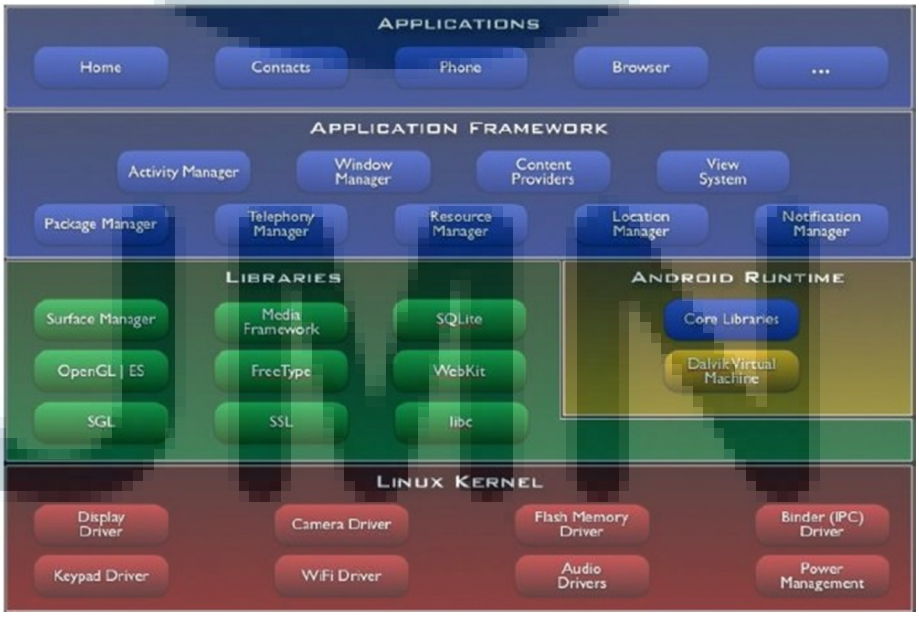
Rancang Bangun ..., Edbert Hansel, FTI UMN, 2015

2.2 Sistem Operasi Android

Android merupakan sistem operasi berbasis Linux yang dirancang secara khusus untuk alat komunikasi *mobile* yang *touch screen* seperti *smartphone* dan *tablet*. Sistem ini dihadirkan dengan berbagai fasilitas seperti *background process*, berbagai macam *user interface*, akses ke *system file*, dan sebagainya. Sistem ini mempunyai keunggulan yaitu terdiri dari beberapa komponen dan dapat menggunakan komponen dari aplikasi lain. Android dikembangkan oleh Google yang bekerja sama dengan Open Handset Alliance [1].

Android mendukung proyek-proyek yang *open source* sehingga banyak pengguna Android yang dapat membangun aplikasinya sendiri dalam ponsel atau tablet Androidnya. Para pengembang Android menyediakan Android *Software Development Kit* (SDK) dan *Native Development Kit* (NDK) yang bisa diperoleh secara gratis untuk setiap *developer* yang ingin mengembangkan aplikasi berbasis Android.

Secara umum, arsitektur dari sistem operasi Android dapat dilihat pada Gambar 2.1.



Gambar 2.1 Arsitektur Sistem Operasi Android [17]

Rancang Bangun ..., Edbert Hansel, FTI UMN, 2015

Dari arsitektur tersebut, seorang *developer* biasanya tidak perlu untuk mengakses hingga ke *libraries* dan Android *Runtime* ataupun hingga Kernel untuk membangun suatu aplikasi baru. Namun, *developer* hanya perlu mengakses pada 2 *layer* di atas yaitu *Applications* dan *Application Framework* saja. Hal ini dikarenakan *libraries* dan *runtime* sudah terenkapsulasi di dalam *Application Framework* secara umum.

Sistem operasi Android juga memberikan fasilitas kepada aplikasi di dalamnya untuk menggunakan komponen dari aplikasi lainnya. Hal ini mengacu pada konsep *task.* Sebagai contoh kamera dapat digunakan untuk mengambil foto dan hasil foto tersebut langsung dapat diterima dalam aplikasi dan dapat diproses lebih lanjut melalui aplikasi.

Hal lain yang juga ditawarkan dalam sistem operasi Android adalah Google Play. Di dalam Google Play ini *developer* diberikan kesempatan untuk menawarkan aplikasinya kepada masyarakat luas sehingga masyarakat luas dapat mendownload dan menggunakannya. Bila pembuat aplikasi mengunggah versi terbaru dan memilih untuk mempublikasikannya maka secara otomatis para pengguna aplikasi yang telah membeli atau mengunduh aplikasi tersebut dari Google Play akan diberikan notifikasi tentang *update* terbaru pada aplikasi tersebut.

Google Play juga dapat mengidentifikasi kompaitibilitas aplikasi dengan sistem Android pengguna. Bila versi sistem operasi Android pengguna tidak mendukung dengan aplikasi maka pengguna akan diberikan peringatan tentang hal tersebut sehingga bila tidak kompaitibel maka tidak dapat digunakan dan pengguna tidak dirugikan bila aplikasi tersebut berbayar.

Dalam pengembangan aplikasi untuk *platform* Android dibutuhkan sebuah *tool* yang disebut *Android Software Development Kit (*Android SDK) yang dapat diperoleh secara gratis. Aplikasi ini dapat digunakan untuk membuat, kompilasi, dan mengemas aplikasi Android. Bahasa pemrograman yang digunakan dalam aplikasi ini adalah Java. Di dalam aplikasi ini terdapat Android Debug Bridge (ADB) yang merupakan suatu alat untuk mengirim dan menjalankan aplikasi yang telah dibuat ke

Rancang Bangun ..., Edbert Hansel, FTI UMN, 2015

dalam peralatan Android ataupun *virtual device* yang telah disediakan. Tersedia berbagai macam pilihan *virtual device* untuk menjalankan aplikasi ini sehingga tampilan aplikasi di berbagai perangkat dapat dilihat secara umum. Namun, untuk menjalankan *virtual device* tersebut, diperlukan spesifikasi komputer yang tinggi seperti membutuhkan *memory* minimal 4 GB.

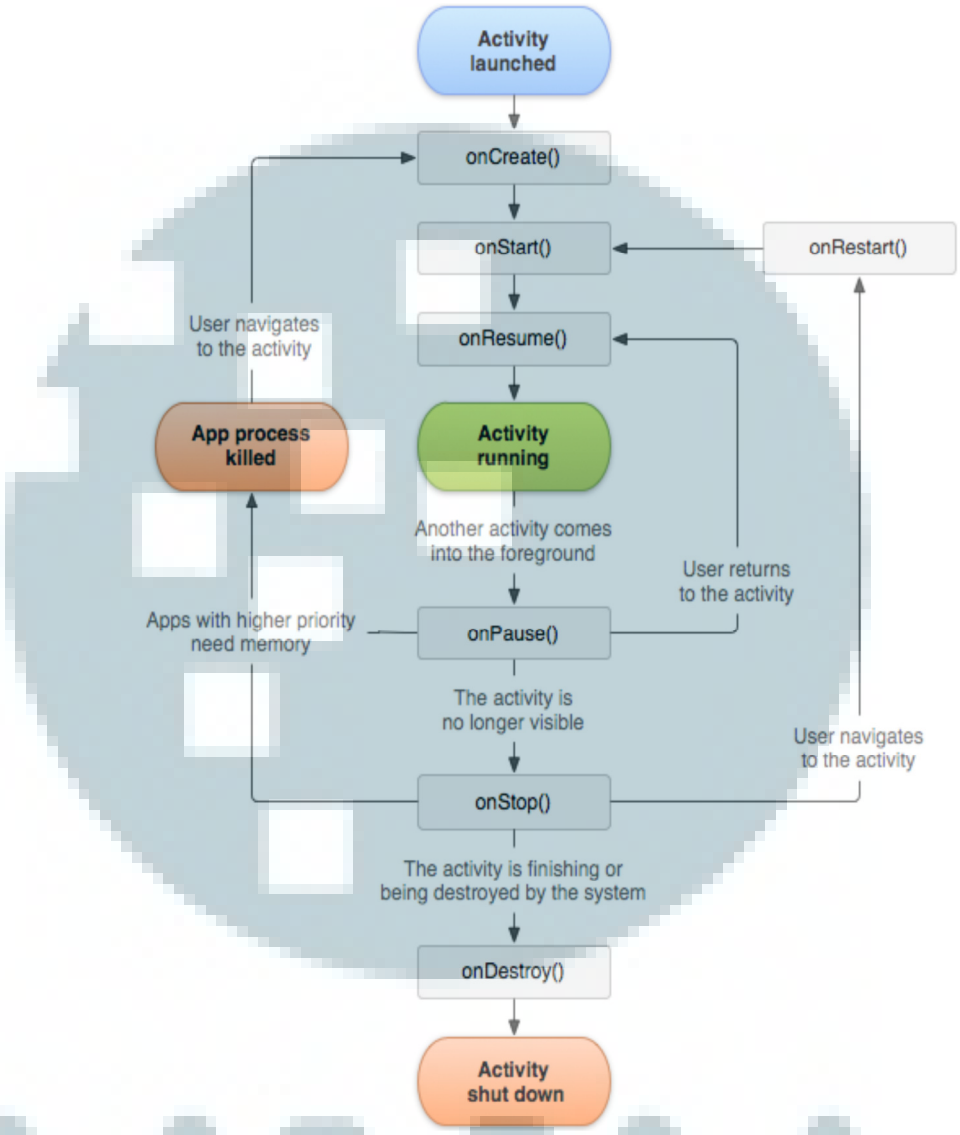
Berikut adalah elemen-elemen penting dalam pemrograman Android.

2.2.1 *Activity*

Activity merupakan representasi visual dari aplikasi. Di dalam satu aplikasi dapat memiliki beberapa *activity*. Namun, hanya 1 *activity* yang dapat aktif dalam satu waktu saja. Bila *activity* berpindah dari satu ke yang lainnya maka *activity* yang aktif akan dihancurkan terlebih dahulu. Setelah itu *activity* lain baru di-*create*. Untuk mengembalikan *activity* yang sebelumnya telah dihancurkan, perlu disimpan di dalam suatu *container* sehingga pada saat dibuat ulang seolah-olah dapat me-*resume* yang telah dihancurkan sebelumnya.

Salah satu hal yang perlu diperhatikan dalam pembuatan *activity* adalah bagaimana menyimpan kondisi terakhir dari *activity*. Bila orientasi dari peralatan Android diubah maka tidak secara otomatis *activity* menyimpan kondisi terakhirnya. Bila hal itu terjadi maka *activity* hanya akan membuat ulang secara keseluruhan sehingga perubahan yang dilakukan tidak tersimpan. Untuk itu disediakan sebuah fitur untuk menyimpan kondisi tersebut yaitu *onSaveInstanceState(Bundle).* Dengan fasilitas ini kondisi terakhir dapat disimpan dan bila terjadi perubahan orientasi atau semacamnya Android akan mengembalikan kondisi terakhir yang tersimpan dan pada saat masuk ke *onCreate()*, sistem akan melihat data tersimpan dahulu dan mengembalikannya bila ada, bila tidak maka sistem akan membuat ulang secara keseluruhan [8]. Secara keseluruhan, *activity lifecycle* dapat dilihat pada Gambar 2.2 di bawah.

Rancang Bangun ..., Edbert Hansel, FTI UMN, 2015



Gambar 2.2 *Activity Lifecycle* [1]

2.2.2 *Fragment*

Fragment merupakan komponen dari *activity* yang mengenkapsulasi kode aplikasi sehingga lebih mudah digunakan kembali dan mampu mengakomodir berbagai ukuran layar. Di dalam satu *activity* bisa memiliki beberapa *fragment* yang dapat aktif secara bersama-sama. *Fragment* dapat berperan seolah-olah seperti *activity* dan memiliki beberapa keuntungan karena *fragment* tidak

Rancang Bangun ..., Edbert Hansel, FTI UMN, 2015

langsung dihancurkan ketika selesai digunakan dan *fragment* dapat dipanggil dengan berbagai ukuran di depan *activity* ataupun *fragment* lain sesuai dengan kebutuhan [8].

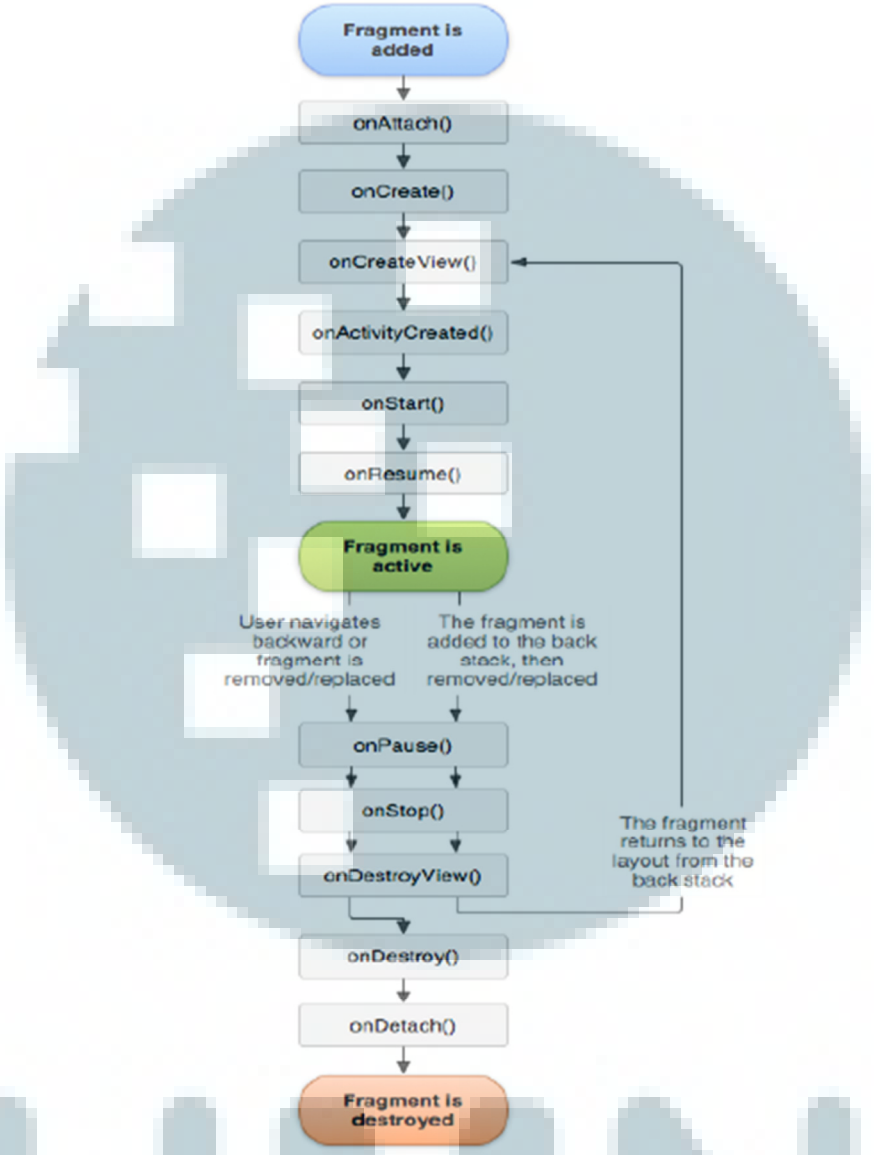
Dalam *fragment*, secara umum perlu dideklarasikan tiga metode yaitu *onCreate*(), *onCreateView(),* dan *onPause()*. Perbedaan antara *onCreate()* dengan *onCreateView()* adalah sebagai berikut. Untuk inisialisasi diletakkan pada *onCreate()* sedangkan untuk membangun dan menampilkan *User Interface* diletakkan pada *onCreateView()*. Di dalam *onCreateView*() nilai dari *view* akan dikembalikan, sedangkan di dalam *onPause*() perubahan dan kondisi dari *fragment* yang ada disimpan sehingga ketika kembali ke *fragment* tersebut kondisi terakhirnya dapat dikembalikan.

Keunggulan lain dari penggunaan *fragment* adalah adanya fasilitas *Fragment Manager*. *Fragment Manager* memungkinkan user untuk melakukan transaksi *fragment*. Melalui *Fragment Manager* user dapat membuat perubahan menggunakan beberapa metode seperti *add(), remove(), dan replace().* Untuk mengeksekusi perubahan digunakan metode *commit*(), maka seluruh perubahan diterapkan secara bersama-sama. Sebelum dilakukan *commit*(), bila salah satu *fragment* dibuang ada 2 pilihan yang dapat dipilih yaitu

a. Bila ingin menghancurkan *fragment* tanpa perlu menyimpan kondisi sebelum dihancurkan, maka *commit()* dapat langsung dipanggil.
b. Bila ingin menyimpan kondisi terakhir *fragment*, *addToBackStack()* perlu ditambahkan sebelum di-*commit* sehingga kondisi terakhir *fragment* akan tersimpan dan dapat dipanggil kembali.

Secara keseluruhan, *fragment lifecycle* dapat dilihat pada Gambar 2.3 di bawah.

Rancang Bangun ..., Edbert Hansel, FTI UMN, 2015



Gambar 2.3 *Fragment Lifecycle* [1]

2.2.3 *View*

View adalah komponen di dalam Android yang digunakan sebagai *user interface*. Di dalam Android sudah tersedia *built-in view* pada android.view.*View class* seperti *edittext, button, textfield*,dsb. Dari *built-in view* tersebut kita dapat menggunakannya langsung dan dapat mengubah atribut di dalamnya sesuai

Rancang Bangun ..., Edbert Hansel, FTI UMN, 2015

kebutuhan. Namun, bila kita ingin merubah *behavior*-nya kita perlu membuat *custom v*iew yang merupakan suatu *class* dimana *class* tersebut menambahkan ataupun mengganti keseluruhan *behavior* dari *view* standar dengan membuat desainnya serta menggunakan *adapter* untuk mengolah bagaimana *view* tersebut akan menampilkan hasil di layar.

Penggunaan *custom view* dilakukan untuk tujuan khusus tertentu dari aplikasi yang dibuat. Hal ini dilakukan bila *built-in view* yang telah tersedia tidak dapat mengakomodir kebutuhan *developer* untuk menghasilkan tampilan yang diinginkan. Pembuatan *custom view* yang membuat suatu *class* baru dengan melakukan *extend* terhadap *view* standar seperti *textView* ataupun *listView* disertai dengan suatu xml dimana xml *file* ini akan memanggil *class* yang telah dibuat tadi sehingga dapat memunculkan tampilan sesuai yang dibutuhkan dan dibuat sebelumnya.

2.2.4 *Adapter*

Adapter merupakan sebuah komponen yang bekerja sebagai jembatan antara *view* dengan data yang akan ditampilkan di dalam *view*. *Adapter* menyediakan akses kepada data yang akan ditampilkan dan bagaimana cara menampilkannya. *Adapter* juga bertanggung jawab untuk membuat *view* untuk tiap *item* yang ada di dalam *dataset*. Ada beberapa *adapter* yang sudah *built-in* di dalamnya seperti: *simpleCursorAdapter* yang digunakan untuk membuat *view* pada *listView*. Namun, *adapter built-in* ini memiliki keterbatasan dalam menampilkan *view*. *Adapter* ini tidak dapat sembarangan digunakan untuk menampilkan *view*.

Seperti halnya *view, adapter* juga dapat dimodifikasi. Hal ini dilakukan agar dapat memenuhi keinginan pengembang aplikasi untuk menampilkan sesuatu secara berbeda dari yang standar. Pengembang tidak hanya dapat melakukan perubahan pada sisi *view* saja tetapi juga dapat merubah *adapter.* Untuk merubahnya kita perlu membuat suatu *class adapter* baru yang melakukan *extend*

Rancang Bangun ..., Edbert Hansel, FTI UMN, 2015

dari *adapter* tertentu seperti *simplecursor adapter* ataupun *baseAdapter* bila hanya memerlukan basis *adapter* yang akan dikembangkan lebih lanjut. Beberapa *method* yang dimiliki dapat dirubah sesuai dengan kebutuhan dengan melakukan *override* dan menambahkan ataupun mengganti *behavior*-nya.

2.3 *Speech Recognition*

Speech Recognition adalah suatu pengembangan teknik dan sistem yang memungkinkan komputer untuk menerima input berupa kata yang diucapkan [10]. Teknologi ini memungkinkan suatu perangkat untuk mengenali dan memahami kata-kata yang diucapkan dengan cara digitalisasi kpockeata dan mencocokkan sinyal digital tersebut dengan suatu pola tertentu yang tersimpan dalam perangkat. Kata-kata yang diucapkan diubah bentuknya menjadi sinyal digital dengan cara mengubah gelombang suara menjadi sekumpulan angka yang kemudian disesuaikan dengan kode-kode tertentu untuk mengidentifikasikan kata-kata tersebut. Hasil dari identifikasi kata yang diucapkan dapat ditampilkan dalam bentuk tulisan (*speech to text*) atau dapat dibaca oleh perangkat teknologi sebagai sebuah komando untuk melakukan suatu pekerjaan, misalnya penekanan tombol pada telepon genggam yang dilakukan secara otomatis dengan komando suara.

Alat pengenal ucapan (*speech recognizer*) membutuhkan sampel kata sebenarnya yang diucapkan dari pengguna. Sampel kata akan didigitalisasi, disimpan dalam komputer, dan digunakan sebagai *database* dalam mencocokkan kata yang diucapkan selanjutnya.

Terdapat dua jenis mode pada sistem *speech recognition* yaitu [10]:

1. Mode Diktasi

   Pada mode ini pengguna komputer dapat mengucapkan kata atau kalimat yang selanjutnya akan dikenali oleh komputer dan diubah menjadi data teks. Kemungkinan jumlah kata yang dapat dikenali dibatasi oleh jumlah kata yang telah terdapat pada *database*. Pengenalan mode diktasi merupakan *speaker*

Rancang Bangun ..., Edbert Hansel, FTI UMN, 2015

dependent. Keakuratan pengenalan mode ini bergantung pada pola suara dan aksen pembicara serta pelatihan yang telah dilakukan.

2. Mode *Command and Control*

   Pada mode ini pengguna komputer mengucapkan kata atau kalimat yang sudah terdefinisi terlebih dahulu pada *database* dan selanjutnya akan digunakan untuk menjalankan perintah tertentu pada aplikasi komputer. Jumlah perintah yang dapat dikenali tergantung dari aplikasi yang telah mendefinisikan terlebih dahulu pada *database* jenis-jenis perintah yang dapat dieksekusikan. Mode ini merupakan *speaker independent* karena jumlah kata yang dikenali biasanya terbatas sekali dan ada kemungkinan pembicara tidak perlu melakukan pelatihan pada sistem sebelumnya.

Terdapat empat proses utama pada sistem *speech recognition*, antara lain:

1. Pemisahan kata

   Pemisahan kata adalah proses untuk memisahkan suara yang diucapkan oleh pengguna menjadi beberapa bagian. Masing-masing bagian bisa berupa kalimat ataupun hanya sebuah kata. Terdapat tiga metode yang bisa digunakan pada proses pemisahan kata ini:

   a. *Discrete Speech*

      Pada metode ini, pengguna diharuskan mengucapkan kalimat secara terpenggal dengan adanya jeda sejenak di antara kata. Jeda tersebut digunakan oleh sistem untuk mendeteksi awal dan akhir sebuah kata. *Discrete speech* ini mempunyai kelebihan yaitu sedikit *resource* (memori komputer, waktu proses) yang digunakan oleh sistem untuk mendeteksi suara, tetapi juga mempunyai kelemahan yaitu ketidaknyamanan pengguna dalam mengucapkan sebuah kalimat.

   b. *Word Spotting*

      Pada metode ini dalam sebuah kalimat yang diucapkan pengguna, sistem hanya mendeteksi kata yang terdapat di dalam perbendaharaan yang

Rancang Bangun ..., Edbert Hansel, FTI UMN, 2015

dimilikinya dan mengabaikan kata-kata lain yang tidak dimilikinya. Jika pengguna mengucapkan kalimat yang berbeda tetapi di dalam kalimat tersebut terdapat sebuah kata yang sama dan terdapat di perbendaharaan sistem, maka hasil pengenalan akan sama. Kelemahan dari *word spotting* adalah besar kemungkinan sistem akan melakukan kesalahan arti pengenalan dalam bentuk kalimat. Sedangkan kelebihannya adalah pengguna dapat mengucapkan kalimat secara normal tanpa harus berhenti di antara kata.

c. *Continuous Speech*

Pada metode ini sistem akan mengenali dan memproses setiap kata yang diucapkan, serta menghasilkan keakuratan dalam mengenali ucapan pengguna. Namun, metode ini memerlukan *resource* yang besar dalam prosesnya. Sistem harus bisa mendeteksi awal dan akhir setiap kata dalam kalimat tanpa adanya jeda di antara kata-kata tersebut. Setelah berhasil memisahkan kata, langkah selanjutnya adalah mencocokkan dengan perbendaharaan kata yang dipunyainya.

2. Ketergantungan terhadap pengguna

Sistem *speech recognition* memiliki tiga sifat dilihat dari ketergantungan terhadap pengguna, antara lain:

a. *Speaker Dependent*

Sistem membutuhkan pelatihan untuk setiap pengguna yang akan menggunakannya. Sistem tidak akan bisa mengenali pengguna yang belum pernah melakukan pelatihan.

b. *Speaker Independent*

Pengguna tidak perlu melakukan pelatihan sebelum dapat menggunakan sistem karena sistem mampu mengenali suara semua pengguna tidak tergantung warna suara dan dialek yang digunakan.

c. *Speaker Adaptive*

Rancang Bangun ..., Edbert Hansel, FTI UMN, 2015

Speaker adaptive merupakan perpaduan dari *speaker dependent* dan *speaker independent*, dimana pengguna tidak perlu melakukan pelatihan dan keakuratan pengenalan sistem akan makin meningkat jika pengguna yang sama bekerja terus menerus selama beberapa waktu tertentu.

3. Pencocokan kata

Pencocokan kata adalah proses untuk mencocokkan kata ucapan yang berhasil diidentifikasi dengan *database* yang dipunyai oleh sistem. Pada proses pencocokan kata terdapat dua metode yang dapat digunakan:

a. *Whole-word Matching*

Sistem akan mencari di *database* kata yang sama persis dengan kata hasil ucapan pengguna. Caranya adalah dengan membandingkan sinyal digital audio yang datang terhadap *template* rekaman kata. Teknik ini membutuhkan waktu lebih sedikit pengolahan dari pencocokan sub-kata atau fonem, tetapi mensyaratkan bahwa pengguna merekam setiap kata yang akan dikenali, kadang-kadang beberapa ratus ribu kata. *Template* seluruh kata juga membutuhkan memori penyimpanan yang besar (antara 50 dan 512 *byte* per kata) dan hanya praktis jika pengenalan kosakata tersebut sudah dikenal ketika aplikasi dikembangkan

b. *Phoneme Matching*

Pada pola ini, sistem mempunyai kamus fonem atau sub-kata. Fonem adalah bagian terkecil dan unik dari suara yang membentuk sebuah kata. Sistem mencari fonem dan kemudian melakukan pengenalan pola lanjut. Teknik ini membutuhkan lebih banyak pemrosesan dan pencocokan seluruh kata, tetapi membutuhkan penyimpanan lebih sedikit (antara 5 dan 20 byte per kata). Selain itu, pengucapan kata dapat ditebak dari teks bahasa Inggris tanpa mengharuskan pengguna untuk berbicara kata yang sebelumnya.

Pada rancang bangun aplikasi ini, metode yang diimplementasikan adalah *phoneme matching*.

Rancang Bangun ..., Edbert Hansel, FTI UMN, 2015

4. Perbendaharaan kata

   Terdapat dua hal yang harus diperhatikan pada perbendaharaan kata, yaitu ukuran dan keakuratan. Jika perbendaharaan kata berjumlah banyak, maka sebuah sistem akan mudah dalam melakukan pencocokan kata. Namun, dengan makin meningkatnya jumlah perbendaharaan kata, maka jumlah kata yang mempunyai ucapan hampir sama juga meningkat, dimana hal ini menurunkan keakuratan pengenalan. Sebaliknya, jika sebuah sistem memiliki perbendaharaan kata sedikit, maka keakuratan pengenalan akan tinggi karena sedikitnya kata yang hampir sama, tetapi akan semakin banyak kata yang tidak terkenali. Untuk sistem *speech recognition* dengan mode *command and control*, akan lebih baik jika menggunakan jumlah perbendaharaan kata sedikit (kurang dari 100 kata). Sementara untuk mode diktasi akan membutuhkan jumlah perbendaharaan kata yang banyak.

2.4 Pocketsphinx

Pocketsphinx merupakan *speech recognition library* versi *mobile application* dari sistem Sphinx yang dirancang oleh Carnegie Mellon University (CMU) [3]. Pada *library* ini, metode yang digunakan dalam sistem *speech recognition* adalah metode Hidden Markov Model (HMM). Proses pembelajaran unit-unit suara disebut *training*, sedangkan proses menggunakan pengetahuan yang diperoleh untuk menyimpulkan urutan yang paling mungkin dari unit dalam sinyal yang diberikan disebut *decoding*, atau secara sederhana disebut pengenalan (*recognition*). Karena terdapat dua proses tersebut maka diperlukan SPHINX *trainer* dan SPHINX *decoder*.

2.4.1 Komponen Untuk *Training*

Sphinx trainer terdiri dari kumpulan program yang setiap bagiannya memiliki tugas yang spesifik. Parameter-parameter dari model unit suara harus dipelajari dengan menggunakan kumpulan sampel sinyal suara pada *training database*. *Transcript file* adalah *file* yang berisi urutan kata-kata dan suara yang tidak diucapkan yang ditulis bersesuaian dengan sinyal suara. Dibutuhkan juga

Rancang Bangun ..., Edbert Hansel, FTI UMN, 2015

kamus (*dictionary*) yang memetakan setiap kata ke urutan unit suara. Jadi, sebagai tambahan pada sinyal suara, harus ada sekumpulan transkrip untuk *database* (dalam sebuah *file* tunggal) dan dua kamus. Satu kamus dimana kata-kata yang sah dalam bahasa dipetakan pada urutan unit suara (atau unit sub-kata) dan satu lagi dimana suara non-ucap dipetakan ke *non-speech* atau suara seperti unit suara. Bentuk yang pertama adalah kamus bahasa (*language dictionary*) dan yang kedua yaitu pengisi kamus (*filler dictionary*). Terdapat tujuh buah komponen yang diperlukan untuk proses *training* yang terdiri dari:

a. *Phonetic dictionary* (kamus fonetik) dengan *extention* .dic
b. *Phoneset file* dengan *extention* .phone
c. *Transcript file* dengan *extention* .transcription
d. *Language model* (model bahasa) dengan *extention* .DMP
e. Daftar *file* untuk *training* dengan *extention* .fileids
f. *Filler dictionary* dengan *extention* .fillers
g. *File* suara dengan format wav 8 kHz, 16 bit, mono.

2.4.2 Komponen Untuk *Decoding*

Decoder juga terdiri dari satu set program, yang telah disusun untuk memberikan *executable* tunggal yang melakukan tugas pengenalan saat diberi *input* yang tepat. *Input* yang perlu diberikan adalah model akustik yang terlatih, sebuah *file* indeks model, model bahasa, kamus bahasa, kamus pengisi, dan kumpulan sinyal akustik yang perlu dikenali. Data yang akan dikenali secara umum disebut sebagai data uji (*test data*). Terdapat lima buah komponen yang diperlukan untuk proses *decoding* yaitu:

a. *Source code decoder*
b. *Language dictionary*
c. *Filler dictionary*
d. *Language model*
e. Data uji

Rancang Bangun ..., Edbert Hansel, FTI UMN, 2015

2.4.3 Penulisan Fonetik

Penulisan fonetik pada *pocketsphinx* menggunakan sistem Arpabet. Arpabet adalah transkripsi fonetik yang dikembangkan oleh *Advanced Research Projects Agency* (ARPA) sebagai bagian dari *Speech Understanding Project* (1971-1976). Arpabet mewakili setiap fonem dari dalam bahasa Inggris Amerika umum (*General American English*) dengan urutan yang berbeda dari karakter ASCII. Dalam Arpabet, setiap fonem diwakili oleh satu atau dua huruf. Digit digunakan *indicator stress* (penekanan) dan ditempatkan pada akhir suku kata vokal yang ditekankan.

Semua kata yang terdapat dalam *language model* harus berada dalam *dictionary* dimana setiap kata diubah menjadi kumpulan fonem. Agar dapat berfungsi dengan baik dan benar, maka setiap kata harus diubah menjadi kumpulan fonem dengan mengikuti aturan seperti pada Tabel 2.1 dan Tabel 2.2. Pada Tabel 2.2 terdapat dua macam pengucapan huruf e, sehingga terdapat dua aturan fonem untuk huruf tersebut.

Tabel 2.1 Aturan Fonem Inggris [4]

Fonem	Contoh	Hasil Terjemahan
AA	odd	AA D
AE	at	AE T
AH	hut	HH AH T
AO	ought	AO T
AW	cow	K AW
AY	hide	HH AY D
B	be	B IY
CH	cheese	CH IY Z
D	dee	D IY
DH	thee	DH IY

Fonem	Contoh	Hasil Terjemahan
EH	ed	EH D
ER	hurt	HH ER T
EY	ate	EY T
F	fee	F IY
G	green	G R IY N
HH	he	HH IY
IH	it	IH T
IY	eat	IY T
JH	gee	JH IY
K	key	K IY

Rancang Bangun ..., Edbert Hansel, FTI UMN, 2015

Fonem	Contoh	Hasil Terjemahan
L	lee	L IY
M	me	M IY
N	knee	N IY
NG	ping	P IH NG
OW	oat	OW T
OY	toy	T OY
P	pee	P IY
R	read	R IY D
S	sea	S IY ZH ER
SH	she	SH IY

Fonem	Contoh	Hasil Terjemahan
T	tea	T IY
TH	theta	TH EY T AH
UH	hood	HH UH D
UW	two	T UW
V	vee	V IY
W	we	W IY
Y	yield	Y IY L D
Z	zee	Z IY
ZH	seizure	S IY ZH ER

Tabel 2.2 Aturan Fonem Indonesia [20]

Fonem Indonesia	Fonem Terjemahan
A	AH
B	B
C	CH
D	D
E	AH
E	EH
F	F
G	G
H	HH
I	IH
J	JH
K	K
L	L
M	M
N	N
O	OW

Fonem Indonesia	Fonem Terjemahan
P	P
Q	K
R	R
S	S
T	T
U	UH
V	V
W	W
Y	Y
Z	Z
AI	AH IH
KH	K HH
NG	NG
NY	NY
SY	SY

Rancang Bangun ..., Edbert Hansel, FTI UMN, 2015

2.5 *Hidden Markov Model* (HMM)

Riset mengenai pengenalan pola berkembang begitu pesat. Lahirnya metode *Hidden Markov Model* (HMM) menjadi salah satu pemicu pembaruan teknik pengolahan *speech* (ucapan / suara) yang dapat menghasilkan tingkat kesalahan seminimal mungkin dan meningkatkan ketangguhan pengklasifikasian pola pada berbagai kondisi yang kurang baik. Kelebihan dari metode HMM adalah sebagai berikut:

- Kemampuan yang handal untuk memodelkan sesuatu yang kompleks.
- Algoritma yang efisien dalam menangani sejumlah data yang sangat besar.
- Banyak dipakai di berbagai bidang ilmu, contoh: kompresi kalimat, *speech recognition*, bioinformatika, *handwriting recognition*, *gesture recognition*.

Hidden Markov Model (HMM) adalah suatu model statistik dari sebuah sistem yang diasumsikan sebuah proses Markov dengan parameter yang tidak diketahui [9]. Parameter-parameter tersembunyi (*state*) harus ditentukan dari parameter-parameter yang dapat diamati. Kemudian, parameter-parameter yang ditentukan dapat digunakan untuk analisis yang lebih jauh, misalnya untuk aplikasi *pattern recognition*. Dalam HMM *state* tidak dapat diamati secara langsung, akan tetapi yang dapat diamati adalah variabel-variabel yang terpengaruh oleh *state*. Setiap *state* memiliki distribusi probabilitas atas *token-token output* yang mungkin muncul. Oleh karena itu rangkaian *token* yang dihasilkan oleh HMM memberikan sebagian informasi tentang sekuens *state-state*.

Hidden markov model adalah sebuah *triple* (π,A,B). Dimana setiap probabilitas dalam *state transition matrix* dan *confusion matrix* adalah t*ime independent*, sehingga kedua matriks tersebut tidak berubah seiring bertambahnya waktu walaupun sistem berkembang. Dalam praktiknya, hal ini adalah asumsi yang paling tidak realistis atas *markov model* mengenai proses nyata [19].

Rancang Bangun ..., Edbert Hansel, FTI UMN, 2015

Hidden markov model adalah sebuah kumpulan terhingga atas *states*, dimana setiap *state* terhubung dengan distribusi probabilitas. Transisi antar *states* diatur oleh kumpulan probabilitas yaitu *transition probabilities*. Dalam keadaan tertentu, hasil observasi diperoleh sesuai distribusi probabilitas yang terkait. Bagi pengamat eksternal, hanya hasil observasi yang dapat dilihat, sedangkan *states* tersembunyi, maka dari itu dinamakan *hidden markov model* [14].

Untuk mendefinisikan *hidden markov model* secara lengkap, elemen yang dibutuhkan:

1. Jumlah *states* dalam model, N.
2. Jumlah simbol observasi dalam alfabet, M. Jika observasi berkelanjutan, maka M adalah tak terhingga.
3. Sebuah kumpulan dari *state transition probabilities* $\Lambda = \{a_{ij}\}$.

   $$a_{ij} = p\{q_{t+1} = j \mid q_t = 1\},\ 1 \leq i, j \leq N \qquad (2.1)$$

   dimana q_t menandakan *state* sekarang. *Transition probabilites* juga harus memenuhi *normal stochastic constraint*,

   $$a_{ij} \geq 0,\ 1 \leq i, j \leq N$$

   dan

   $$\sum_{j=1}^{N} aij = 1,\ 1 \leq i \leq N$$

4. Sebuah distribusi probabilitas pada setiap *state*, $B = \{b_j(k)\}$.

   $$b_j(k) = p\{o_t = v_k \mid q_t = j\},\ 1 \leq j \leq N,\ 1 \leq k \leq M \qquad (2.2)$$

   dimana v_k menandakan simbol observasi k^{th} dalam alfabet, dan a_t parameter vektor sekarang. Untuk memenuhi *stochastic constraint*,

   $$b_j(k) \geq 0,\ 1 \leq j \leq N,\ 1 \leq k \leq M$$

   dan

   $$\sum_{j=1}^{N} aij = 1,\ 1 \leq j \leq N$$

   Jika observasi berkelanjutan, maka harus menggunakan *continuous probability density function*.

   $$b_j(o_t) = \sum_{m=1}^{M} c_{jm} \aleph(\mu_{jm}, \Sigma_{jm}, o_t) \qquad (2.3)$$

Rancang Bangun ..., Edbert Hansel, FTI UMN, 2015

dimana, c_{jm} adalah *weighting coefficient*, μ_{jm} adalah *mean vector*, Σ_{jm} adalah *covariance* matrics. c_{jm} harus memenuhi *stochastic constraint*,

$$c_{jm} \geq 0,\ 1 \leq j \leq N,\ 1 \leq k \leq M$$

dan

$$\sum_{m=1}^{M} c_{jm} = 1,\ 1 \leq j \leq N$$

5. *Initial state distribution*, $\pi = (\pi_i)$ dimana,

$$\pi_i = p\{q_1 = i\},\ 1 \leq i \leq N \qquad (2.4)$$

Maka dari itu notasi dapat disederhanakan menjadi

$$\lambda = (\Lambda, B, \pi) \qquad (2.5)$$

untuk menyatakan sebuah *hidden markov model* dengan *discrete probability distribution*,

$$\lambda = (\Lambda, c_{jm}, \mu_{jm}, \Sigma_{jm}, \pi) \qquad (2.6)$$

untuk menyatakan *hidden markov model* dengan *continuous densities* [14].

Ada tiga masalah yang dapat dipecahkan pada sistem yang menggunakan *hidden markov model* [11]:

1. Diberikan *observation sequence* $O = O_{1...}O_n$ dan model $\lambda = (\pi,A,B)$, bagaimana mengkomputasi secara efisien $P(O|\lambda)$, peluang dari sebuah *observation sequence* O. Masalah ini dapat diselesaikan dengan *forward algorithm*.
2. Diberikan *observation sequence* $O = O_{1...}O_n$ dan model $\lambda = (\pi,A,B)$, bagaimana menentukan *state sequence* Q yang paling optimal. Masalah ini dapat dilihat bagaimana menemukan bagian tersembunyi dari *hidden markov model* [13]. Masalah ini dapat diselesaikan dengan *backward algorithm*.
3. Bagaimana menemukan model $\lambda = (\pi,A,B)$ yang mampu memaksimalkan $P(O|\lambda)$. Masalah ini dapat dilihat bagaimana melatih sebuah model agar sesuai dengan data yang diobservasi [13]. Masalah ini dapat diselesaikan dengan *forward-backward algorithm* [19].

Rancang Bangun ..., Edbert Hansel, FTI UMN, 2015

Speech recognition (pengenalan ucapan) merupakan pengenalan sebuah kata berurutan dari ucapan manusia. Otak manusia berfungsi sebagai generator teks yang menghasilkan kumpulan kata *W*. Kumpulan kata berasal dari saluran akustik yang terdiri dari artikulasi pembicara dan proses akustik lainnya yang mengkonversi kumpulan teks ke dalam suatu bentuk gelombang akustik yang dapat didengar. Saluran akustik dalam komunikasi verbal bertindak sebagai *data transducer* dan komposer.*Speech recognizer* adalah dekoder yang menampilkan proses pembalikkan dari pesan ke bentuk gelombang suara. Oleh karena itu, dekoder menunjukkan keputusan akhir maksimum dari pemeriksaan urutan kata Ŵ seperti berikut:

$$\hat{W} = \overset{argmax}{w} P(W \mid X) = \overset{argmax}{w} P(X \mid W)P(W) \qquad (2.7)$$

dimana P(X | W) adalah nilai dari pemodelan akustik dan P(W) adalah nilai dari model bahasa.

Sistem pengenalan ucapan terdiri dari beberapa komponen dasar, diantaranya:

- *Acoustic feature extraction*: berfungsi untuk mengekstraksi pengenalan suara dari gelombang suara. Biasanya termasuk analisis *cepstral* jangkapendek yang mengolah suatu fitur *vector* dari frekuensi rendah koefisien *cepstral* untuk setiap 10 ms.
- *Acoustic modeling*: berfungsi untuk menyediakan pemodelan statistik bagi observasi akustik *X*.
- *Language modeling*: berfungsi untuk menyediakan linguistik dan batasan *grammar* (tata bahasa) untuk urutan teks *W*. *Language model*(model bahasa) seringkali dibuat berdasarkan *N-gram language model*.
- *Decoding engine*: berfungsi untuk mencari urutan kata terbaik yang diberikan fitur dan model. Untuk pengenalan suara berdasarkan pemodelan HMM, maka *decoding* menggunakan *Viterbi*.

Rancang Bangun ..., Edbert Hansel, FTI UMN, 2015

Tiga masalah dasar dalam HMM yang harus dikaji ulang yaitu masalah evaluasi, masalah *decoding,* dan masalah estimasi. Masalah evaluasi adalah memperkirakan peluang P(X |Λ) dari pengamatan fitur suara urutan vektor *X* yang diberikan *Hidden Markov Model*. Masalah *decoding* yaitu untuk menemukan urutan *state* terbaik *q* yang optimal dari urutan fitur suara *X*. Karena *state* dalam HMM terhubung ke kata dan kelas kata, urutan kata dalam pengucapan suara dapat diidentifikasikan dengan menelusuri *label* kata pada urutan state q. Masalah yang ketiga yaitu estimasi yang diperlukan untuk memperkirakan parameter HMM Λ dari himpunan sampel yang sudah dilatih. Pendekatan biasa yaitu berdasarkan prinsip *Maximum Likelihood* (ML) dan himpunan parameter model Λ diestimasi sehingga *likelihood* pada pelatihan data menjadi maksimal. Beragam ML yang efisien tergantung pada algoritma yang dikembangkan dalam pengenalan suara untuk HMM, seperti *Baum-Welch* dan algoritma segmental *k-means.*

2.5.1 Perhitungan Probabilitas Maju-Mundur (*Forward-Backward*)

Algoritma rekursif yang efisien untuk perhitungan fungsi *likelihood* $f_{X|M}(X|M)$ adalah algoritma maju-mundur. Maksud dari maju-mundur adalah urutan yang dilakukan untuk mencari peluang observasi. Perhitungan pada metode maju-mundur memanfaatkan struktur yang sangat teratur dan berulang [16].

Dalam metode ini, variabel αt probabilitas maju (i) didefinisikan sebagai probabilitas gabungan dari urutan observasi parsial X = [x (0), x (1), ..., x(t)] dan *state* pada waktu *t*, dari model M:

$$\alpha_t(i) = f_{X,S|\mathcal{M}}(\boldsymbol{x}(0), \boldsymbol{x}(1), \ldots, \boldsymbol{x}(t), s(t) = i \,|\, \mathcal{M}) \tag{2.8}$$

Variabel probabilitas maju $\alpha_t(i)$ pada persamaan (2.8) dapat dinyatakan dalam bentuk rekursif pada waktu *t*-1, $\alpha_{t-1}(i)$:

Rancang Bangun ..., Edbert Hansel, FTI UMN, 2015

$$\begin{aligned}\alpha_t(i) &= f_{X,S|\mathcal{M}}\left(x(0),x(1),\ldots,x(t),s(t)=i\,|\mathcal{M}\right)\\ &= \left(\sum_{j=1}^{N} f_{X,S|\mathcal{M}}\left(x(0),x(1),\ldots,x(t-1),s(t-1)=j\,|\mathcal{M}\right)a_{ji}\right)\\ &\quad f_{X|S,\mathcal{M}}\left(x(t)\,|s(t)=i,\mathcal{M}\right)\\ &= \sum_{j=1}^{N}\left(\alpha_{t-1}(j)a_{ji}\right)f_{X|S,\mathcal{M}}\left(x(t)\,|s(t)=i,\mathcal{M}\right)\end{aligned} \tag{2.9}$$

Sama halnya dengan definisi konsep peluang maju, sebuah peluang mundur didefinisikan sebagai peluang *state* pada waktu t yang diikuti oleh urutan observasi parsial [x(t+1), x(t+2), …., x(T-1)]

$$\begin{aligned}\beta_t(i) &= f_{X,S|\mathcal{M}}\left(s(t)=i,x(t+1),x(t+2),\ldots,x(T-1)\,|\mathcal{M}\right)\\ &= \sum_{j=1}^{N} a_{ij} f_{X,S|\mathcal{M}}\left(s(t+1)=j,x(t+2),x(t+3),\ldots,x(T-1)\,|\mathcal{M}\right)\\ &\quad \times f_{X|S,\mathcal{M}}\left(x(t+1)\,|s(t+1)=j,\mathcal{M}\right)\\ &= \sum_{j=1}^{N} a_{ij}\beta_{t+1}(j) f_{X|S,\mathcal{M}}\left(x(t+1)\,|s(t+1)=j,\mathcal{M}\right)\end{aligned} \tag{2.10}$$

2.5.2 Model Re-Estimasi Baum-Welch

Masalah dalam pelatihan HMM adalah mengestimasi model parameter M = (π, A, F) untuk himpunan X yang diberikan [16]. Parameter-parameter ini adalah *state* awal pada peluang π, parameter-parameternya adalah peluang state *awal* π, peluang state transisi matriks A dan densitas kontinu (atau diskrit) observasi state *probability density function* (pdf). Parameter HMM diperkirakan dari sekumpulan contoh-contoh training {X = [x(0), ..., x (T-1)]}, dengan memaksimalkan $f_{x|M}$ (X|M), *likelihood* dari model dan data pelatihan. Metode Baum-Welch adalah iterasi dari metode *likelihood* yang dimaksimalkan berdasarkan probabilitas maju-mundur. Untuk HMM sebuah M, probabilitas

Rancang Bangun ..., Edbert Hansel, FTI UMN, 2015

posterior dari transisi pada waktu t dari *state i* ke *state j* dari model M, diberikan sebuah urutan observasi berupa X.

$$\begin{aligned}\gamma_t(i,j) &= P_{S|X,\mathcal{M}}\big(s(t)=i, s(t+1)=j \,|\boldsymbol{X}, \mathcal{M}\big) \\ &= \frac{f_{S,X|\mathcal{M}}\big(s(t)=i, s(t+1)=j, \boldsymbol{X}\,|\mathcal{M}\big)}{f_{X|\mathcal{M}}(\boldsymbol{X}\,|\mathcal{M})} \\ &= \frac{\alpha_t(i)a_{ij}f_{X|S,\mathcal{M}}\big(\boldsymbol{x}(t+1)\,|s(t+1)=j, \mathcal{M}\big)\beta_{t+1}(j)}{\sum_{i=1}^{N}\alpha_{T-1}(i)}\end{aligned} \tag{2.11}$$

dimana $f_{s,x}|M_{(s(t)}$=i, s(t+1)=j, X|M) adalah *pdf join* dari *state* s(t) dan s(t+1) dan urutan observasi X, dan fx|s (x(t+1)|s(t+1)=i) adalah *state* observasi pdf untuk *state i*. Untuk sebuah densitas observasi diskrit HMM, persamaan 2.11 diganti dengan *state* observasi *probability mass function* (pmf) diskrit Px|s (x+1)|s(t+1)=i). Probabilitas posterior dari *state i* pada waktu t dengan model M dan observasi X adalah

$$\begin{aligned}\gamma_t(i) &= P_{S|X,\mathcal{M}}\big(s(t)=i\,|\boldsymbol{X}, \mathcal{M}\big) \\ &= \frac{f_{S,X|\mathcal{M}}\big(s(t)=i, \boldsymbol{X}\,|\mathcal{M}\big)}{f_{X|\mathcal{M}}(\boldsymbol{X}\,|\mathcal{M})} \\ &= \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N}\alpha_{T-1}(j)}\end{aligned} \tag{2.12}$$

Peluang *state* transisi a_{ij} dapat ditulis sebagai berikut

$$a_{ij} = \frac{\text{Expected number of transitions from state } i \text{ to state } j}{\text{Expected number of transitions from state } i} \tag{2.13}$$

Dari persamaan (2.11) sampai (2.13) peluang *state* transisi dapat diperkirakan ulang dengan rasio

Rancang Bangun ..., Edbert Hansel, FTI UMN, 2015

$$\overline{a}_{ij} = \frac{\sum_{t=0}^{T-2} \gamma_t(i,j)}{\sum_{t=0}^{T-2} \gamma_t(i)} \tag{2.14}$$

untuk sebuah urutan observasi [x(0), …., x(T-1)] dari panjang T, transisi terakhir terjadi pada waktu T-2 yang diindikasikan sebagai batas teratas dari penjumlahan pada persamaan 2.14. Peluang inisial *state* diestimasikan sebagai berikut.

$$\overline{\pi}_i = \gamma_0(i) \tag{2.15}$$

2.5.3 Algoritma *Decoding* Viterbi

Decoding sinyal berfungsi untuk mendapatkan perkiraan posterior maksimum (MAP) dari urutan *state* yang mendasarinya [16]. Urutan *state* MAP dari model M diberikan sebuah sinyal observasi X=[x(0), …., x(T-1)] yang menghasilkan :

$$s^{MAP} = \arg\max_s f_{X,s|\mathcal{M}}(\boldsymbol{X}, \boldsymbol{s}|\mathcal{M})$$

$$= \arg\max_s \left(f_{X|s,\mathcal{M}}(X|s,\mathcal{M}) P_{s|\mathcal{M}}(s|\mathcal{M}) \right)$$

δ_t (*i*) mencatat peluang kumulatif dari jalan terbaik untuk *state i* pada waktu *t*. ψ*t* (*i*) mencatat urutan *state* terbaik untuk *state i* pada waktu *t*.

Langkah ke-1: Inisialisasi, waktu *t*=0, atau *states i*=1, …, N

$$\delta_0(i) = \pi_i f_i(x(0))$$

$$\psi_0(i) = 0$$

Langkah ke-2: Perhitungan rekursif urutan *state ML* dan peluang

Untuk waktu *t*=1, …, T-1

Untuk states *i*=1, …, N

Rancang Bangun ..., Edbert Hansel, FTI UMN, 2015

$$\delta_t(i) = \max_i [\delta_{t-1}(j) a_{ji}] f_i(x(t))$$

$$\psi_t(i) = \arg\max_i [\delta_{t-1}(j) a_{ji}]$$

Langkah ke-3: Pemutusan, mengambil *state* akhir yang paling mungkin

$$s^{MAP}(T-1) = \arg\max_i [\delta_{T-1}(i)]$$

$$Prob_{max} = \max_i [\delta_{T-1}(i)]$$

Langkah ke-4: *Backtracking* melalui urutan *state* yang paling mungkin

Untuk $t = T-2, \ldots, 0$

$$s^{MAP}(t) = \psi_{t+1} [s^{MAP}(t+1)]$$

2.6 Kerangka Berpikir

Penelitian ini dilakukan dengan membangun sebuah aplikasi *speech recognition* pada sistem operasi Android dengan menggunakan sebuah *smartphone* sebagai perangkat keras untuk menerima input berupa kata yang diucapkan oleh *user* dan memprosesnya menggunakan implementasi metode *Hidden Markov Model* (HMM) yang terdapat pada *library* Pocketsphinx. Fonem yang digunakan adalah aturan fonem Indonesia yang dikembangkan oleh Ferdiansyah [20].

Rancang Bangun ..., Edbert Hansel, FTI UMN, 2015