Chapter 10 Speech Recognition

10.1 Introduction

Speech recognition (SR) by machine, which translates spoken words into text has been a goal of research for more than six decades. It is also known as *automatic speech recognition* (ASR), *computer speech recognition*, or simply *speech to text* (STT). The research in speech recognition by machine involves a lot of disciplines, including signal processing, acoustics, pattern recognition, communication and information theory, linguistics, physiology, computer science and psychology. Figure 10-1 below shows a general block diagram of a task-oriented speech recognition system:

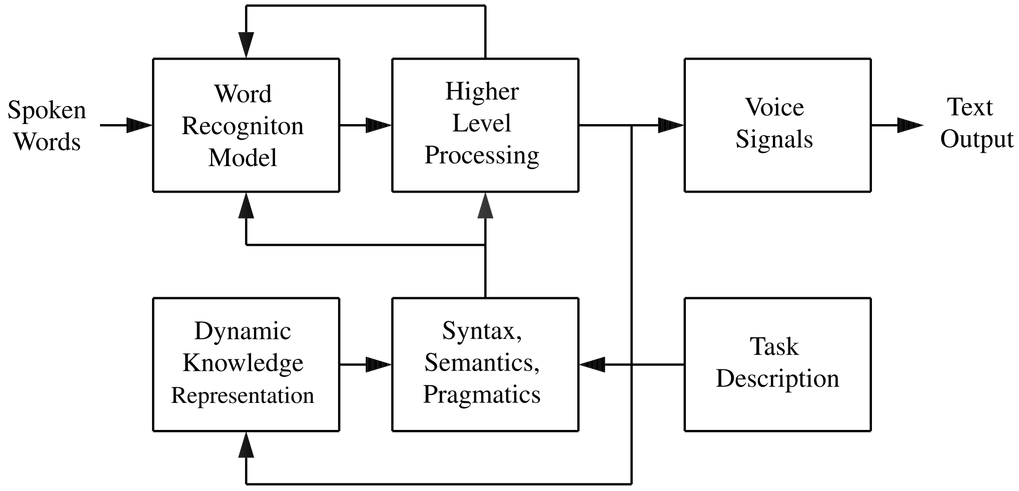


Figure 10-1 A Typical Speech Recognition System

The figure shows a general model for speech recognition, which starts with a speaker generating a speech to accomplish certain tasks. The speech is parsed into a sequence of words that makes sensible meaning according to the syntax, semantics and pragmatics of the recognition task. A higher level processor makes analysis of the spoken words and uses a dynamic knowledge representation to modify the syntax, semantics and pragmatics dynamically. The higher level processor feedback reduces the complexity of analysis and search.

Speech recognition research has been performed for about six decades. Only in recent years did it get widespread applications in the consumer products. The research in the field of speech recognition by machine started in the 1950s in large laboratories such as Bell Laboratories and RCA Laboratories, when researchers exploit the fundamental principles of how human perceives acoustic-phonetics. The speech recognition systems relied on spectral measurements. Later, researchers incorporate statistical techniques to improve upon earlier methods. In the 1960s several fundamental ideas in speech recognition emerged in the US and Japan. Significant milestones were achieved in 1970s by different groups of researchers in the world. In the 1980s, speech research shifted from template-based approaches to statistical modeling methods, especially the hidden Markov model approach. After the 1990s, the research in the field has become mature. Linguists, computer scientists, and engineers worked together to take advantage of large databases and the improvement in computing speed to create applications that had commercial usage. In recent years, mobile phones and desktop computing are inundated with speech recognition applications.

Nowadays, speech recognition (SR) mobile products are ubiquitous. There are many third party SR apps that support Android. The well known Google *Now project* allows a user to give a voice command to an Android device, which will then fetch the result for the user. It recognizes the voice and converts it into text or takes appropriate actions. Android's *Translate* app goes one step further. It lets a user dictate e-mails and text messages. It not only converts English into another spoken language such as French, but also has a *conversation mode* that will translate the French receiver's response back into English.

Some SR apps provide service to another user app through the use of *Intent*. For example, *Google Voice Search* is a Google product that allows users to use Google Search by speaking on a mobile device or a PC, rather than entering data via a keyboard. It is one of the popular recognizers available for Android and supports a wide variety of languages. The app has a very simple *Activity* to notify users for speaking, and the dialog closes promptly as soon as the user stops talking. (Google Now mentioned above is not exactly the same as Google Voice Search. Google Now is a feature of the Google Search app for Android. It shows information about what is currently happening now or will happen in the near future, such as reservations, calender events, weather, and travel information.) Figure 10-2 below shows a UI of a Google Voice Search app.

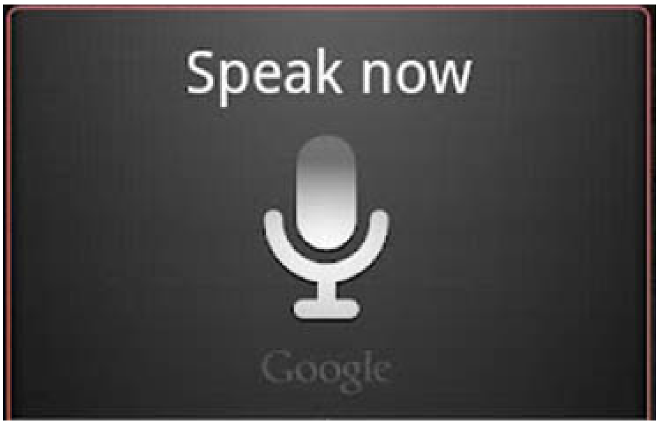


Figure 10-2 A UI of a Google Voice Search App

These applications are built upon the Android voice recognition APIs, which are fairly easy to use. The class *SpeechRecognizer* provides access to the speech recognition service, which allows access to the speech recognizer. The Android documentation recommends that we should not instantiate this class directly, but call **createSpeechRecognizer (** *Context* **)**, and the class's methods must be invoked only from the main application thread. For most SR applications, audio data are sent to remote servers, where speech recognition is performed. Consequently. as most tasks are done remotely, the API should not be used for continuous recognition, which would consume a significant amount of battery and bandwidth. For the API to work properly, a voice recognizer must have been installed in the Android device where the app runs. Normally, the app must be run in a real device. The emulator cannot perform the recognition tasks properly.

Our examples discussed below use a recognizer based on Google Voice Recognition (GVR), which uses neural network algorithms to convert human audio speech to text. GVR works for a number of major languages but we have only considered English in our applications. A neural network consists of many processors working in parallel, mimicking a virtual brain. The usage of parallel processors allows for more computing power and better operation in real-time, but what truly makes a neural network distinct is its ability to adapt and learn based on previous data. A neural network does not use one specific algorithm to achieve its task. It learns by the example of other data. In our examples, GVR uses the Internet to access its large database for voice recognition attempted by previous users. It also looks at previous Google search queries so that the voice recognition engine can guess which phrases are more commonly used than others. This way, even if the user does not speak a certain word clearly, GVR can use the context of the rest of the spoken phrase or sentence to extrapolate what the user is most likely trying to say. In general, a neural network can learn from two major categories of learning methods : supervised or self-organized. In supervised training, an external teacher provides labeled data and the desired output. Meanwhile, self-organization network takes unlabeled data and finds groups and patterns in the data by itself. GVR learns from its own database through the self-organization method.

10.2 A Simple Example

We present here a very simple example of writing an app that converts spoken words to text by making use of the speech recognition API. In this example (*SttDemo0*), a button is presented to the user. When the user clicks on the button, a Google Speech dialog appears and the user can speak. As soon as the user stops speaking, the dialog vanishes and the text of the speech is shown (Figure 10-4). The complete Java code of this app is listed below:

Program Listing 10-1 *MainActivity.java* of Project *SttDemo0*

```
package stt.sttdemo0;

import java.util.*;
import android.widget.*;
import android.os.Bundle;
import android.view.View;
import android.content.pm.*;
import android.app.Activity;
import android.content.Intent;
import android.speech.RecognizerIntent;

/**
 * A very simple SR application that starts speech recognition
 * activity through the use of RecognizerIntent.
 * Spoken words are saved as a list of words of String.
 */
public class MainActivity extends Activity
{
   private static final int REQUEST_CODE = 1989;
   private ListView words;

   @Override
   public void onCreate(Bundle savedInstanceState)
   {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_main);

     Button speechButton=(Button) findViewById(R.id.speechbutton);
     words = (ListView) findViewById(R.id.listview);

     PackageManager pm = getPackageManager();
```

```
    List<ResolveInfo> activities = pm.queryIntentActivities(
        new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH), 0);

    // Disable button in the absence of recognition service
    if ( activities.size() == 0 ) {
      speechButton.setEnabled ( false );
      speechButton.setText ( "Recognizer not present" );
    }
  }

  public void onClick( View view )
  {
    startSpeechRecognition();
  }

  // Start the speech recognition activity through an intent.
  private void startSpeechRecognition()
  {
    Intent intent = new Intent (
                     RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
    intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
                    RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
    startActivityForResult(intent, REQUEST_CODE);
  }

  // Handle the results from the speech recognition activity.
  @Override
  protected void onActivityResult(int requestCode,
                                  int result, Intent intent)
  {
    if (requestCode == REQUEST_CODE && result == RESULT_OK)
    {
      // Populate the list by words recognized by engine
      ArrayList<String> matches=intent.getStringArrayListExtra(
                  RecognizerIntent.EXTRA_RESULTS);
      words.setAdapter(new ArrayAdapter<String>(this,
                          android.R.layout.simple_list_item_1,
                  matches));
      }
      super.onActivityResult(requestCode, result, intent);
  }
}
-----------------------------------------------------------------------
```

This class, *MainActivity*, defines a *ListView* variable named *words* for displaying strings. When a user speaks, the recognition engine guesses what the speaker says and puts all the guessed sentences in *words*.

Inside the **onCreate** method, the class *RecognizerIntent* consists of constants for supporting speech recognition through starting an *Intent*. The *String* constant ACTION_RECOGNIZE_SPEECH tells to start an activity that will prompt the user for speech and send it through a speech recognizer. If an appropriate recognizer is present, the *Activity* will be a started successfully and added to the List *activities*; if no recognizer is present then no activity will be added to the List. Therefore if *activities*.**size**() is 0, we disable the speech button with the statement

speechButton.setEnabled (false);

and display a message saying *Recognizer not present* (Figure 10-3(a)). If a speech recognizer exists in the system, the speech button is displayed (Figure 10-3(b)).

When we click on the speech button, the method **startSpeechRecognition**() is called. This method will call **startActivityForResult** (*Intent*, *int*), which is a method of the class *Activity*. We use *startActivityForResult* rather than *startActivity* because it allows us to get a result back from an activity when it ends. The second integer parameter of the method is a request code that identifies the call and the result will come back through our **onActivityResult** (int, int, Intent) method. In our example, the request code is REQUEST_CODE, which is set to 1989. The method puts all the strings that the recognition engine has recognized in the *ArrayList matches*. The *String RecognizerIntent*.EXTRA_RESULTS signifies an ArrayList<String> of the recognition results when performing ACTION_RECOGNIZE_SPEECH of *RecognizerIntent*. It will be present only when RESULT_OK is returned in an activity result. In a *PendingIntent*, the lack of this extra indicates failure.

To handle the returned results, we have used an *ArrayAdaptor*, which is one of the available adaptors in Android. An *Adaptor* is a collection handler that returns an item of the collection as a view. The *ArrayAdapter* class can handle a list or an array of Java objects as input, and every Java object is mapped to one row. We have used the *android.R.layout* class of the Android OS to display the strings; the class contains all of the publicly available layouts. In particular, we have used the layout *simple_list_item_1* for the display.

In the code, **setAdapter**(*ListAdapter*) is a public method of the class *ListView*. In the example, it sets the data behind the *ListView* object *words*; its input parameter is a *ListAdapter* object responsible for maintaining the data of the list and for producing a view to represent an item of the data set.

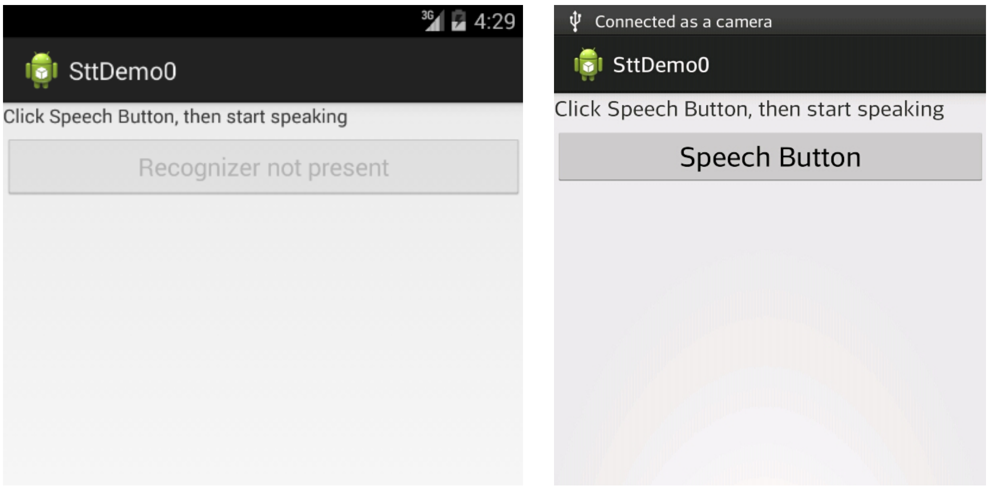
The xml layout code, *activity_main.xml* of this project is very simple. It is presented in Listing 10-2 below:

Program Listing 10-2 *activity_main.xml* of Project *SttDemo0*

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical">

   <TextView
       android:layout_width="fill_parent"
       android:layout_height="wrap_content"
       android:paddingBottom="4dip"
       android:text="Click Speech Button, then start speaking" />

   <Button android:id="@+id/speechbutton"
       android:layout_width="fill_parent"
       android:onClick="onClick"
       android:layout_height="wrap_content"
       android:text="Speech Button" />

   <ListView android:id="@+id/listview"
       android:layout_width="fill_parent"
       android:layout_height="0dip"
       android:layout_weight="1" />
</LinearLayout>
```

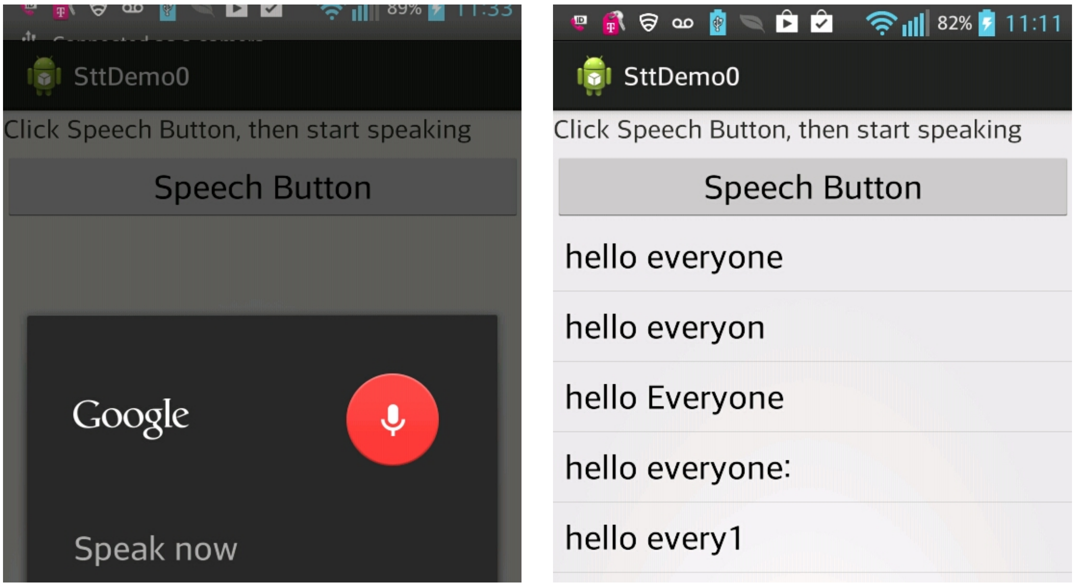
When we run the application, it presents one of the two displays shown in Figure 10-3. Figure 10-3(a) shows the UI when the app cannot find a valid recognition engine in the system. The button is disabled and will not respond to any clicks. Figure 10-3(b) shows the case when a valid recognizer is detected. The user activates the recognition engine by clicking on the button.

When the user clicks on the speech button, the app presents a dialog as shown in Figure 10-4(a). The user speaks to the phone and the engine guesses what the user says, converting all the possible guesses into text; the app saves the strings of text in a list. As soon as the user stops speaking, the dialog closes and the app presents the list of the guessed words. Figure 10-4(b) shows a sample output of the app. The user had spoken the sentence: *Hello, everyone*. The figure shows all the possible sentences that the engine thought it might have heard.


4:29
SttDemo0
Click Speech Button, then start speaking
Recognizer not present
Connected as a camera
SttDemo0
Click Speech Button, then start speaking
Speech Button

(a) (b)

Figure 10-3 UI of *SttDemo0*


SttDemo0
Click Speech Button, then start speaking
Speech Button
Google
Speak now
82% 11:11
SttDemo0
Click Speech Button, then start speaking
Speech Button

```
hello everyone
hello everyon
hello Everyone
hello everyone:
hello every1
```

(a) (b)

Figure 10-4 Input and Output of *SttDemo0*