Pascual Andrés Carrasco Gómez

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

MIARFID Máster en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

Sistema de Reconocimiento Automático del Habla

Toolkit: Kaldi

Introducción

En este trabajo se ha utilizado el toolkit Kaldi para modelar un sistema de reconocimiento automático del habla (RAH) utilizando el corpus de prácticas (Centralita) proporcionado por el profesor de la asignatura, Dr. Fernando García Granada.

El toolkit Kaldi lo podemos obtener a través del siguiente enlace:

https://github.com/kaldi-asr/kaldi

Este toolkit ha sido creado por Daniel Povey (dpovey@gmail.com) y dispone de una documentación detallada y un conjunto de demos (con scripts) para comprender el funcionamiento del toolkit. La documentación la encontramos en el siguiente enlace:

http://kaldi-asr.org/doc/

En la documentación a parte de la explicación del *core* del toolkit, encontramos una documentación detallada de la instalación del mismo.

El conjunto de demos (con scripts) se encuentran en el directorio “kaldi/egs” y son de gran utilidad para tomar un primer contacto con el toolkit, de hecho es relativamente fácil adaptar una demo al corpus con el que queramos trabajar, esto implica utilizar el 80% de los scripts de la demo con la desventaja de que no conocemos realmente el funcionamiento del toolkit.

El objeto de aprendizaje de este trabajo ha sido crear el sistema de RAH para el corpus de la Centralita creando los scripts de Kaldi desde cero, lo cual no ha sido una tarea trivial debido a la comprensión del gran número de ficheros de configuración necesarios para obtener el sistema de RAH.

Corpus

El corpus (Centralita) está formado por 150 frases de entrenamiento (train) y de 50 frases para evaluar el sistema (test). Son frases generadas por un gramática, por lo tanto estamos hablando de un lenguaje acotado, que representan la interacción de una persona con una centralita telefónica. El corpus es monolocutor, ya que en el sistema solo interviene la voz de una persona. Utilizar más de un locutor para entrenar y evaluar el sistema de RAH con Kaldi es muy trivial y se han realizado anotaciones en la descripción de los scripts implementados para dicho caso. El reconocimiento se realiza sobre conjuntos de palabras conectadas, por lo tanto es un ejemplo completo para jugar con el toolkit.

Estructura inicial

En el trabajo se adjunta el directorio “Centralita/” que contiene la siguiente estructura:

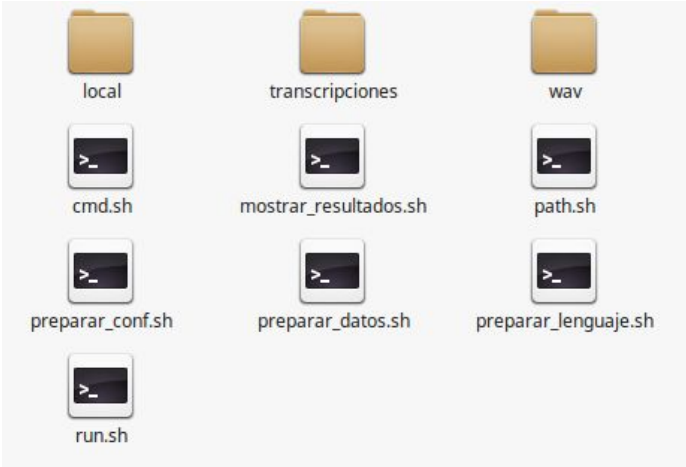

Imagen 1: Estructura inicial para la ejecución en Kaldi (Centralita).

El directorio “Centralita/” se tiene que copiar en el directorio “egs” de la instalación de Kaldi para su correcta ejecución. Es importante tener en cuenta que la ejecución obtiene el modelo de lenguaje (ML) con el toolkit SRILM, el cual debe estar instalado y configurado adecuadamente (instalación mediante el toolkit con el script “kaldi/tools/install_srilm.sh”).
A continuación comentamos los directorios y scripts implementados:

Directorios

- wav: Contiene todos los .wav con los que va a trabajar el sistema (train + test).
- transcripciones: Contiene las frases correspondientes a los wavs (enumeradas) tanto para la parte de train como para la parte de test.
- local: Contiene el conjunto de programas (en python2.7) necesarios para generar el conjunto de ficheros de configuración necesarios para el sistema de RAH en Kaldi. Estos scripts se describirán en un apartado posterior de la memoria del trabajo.

Scripts

- preparar_datos.sh: Se encarga de dividir el corpus de .wavs en train y test. También se encarga de preparar la configuración acústica de datos para poder obtener los modelos y realizar el reconocimiento.
- preparar_lenguaje.sh: Se encarga de definir la parte léxica y el conjunto de fonemas con los que va a trabajar el reconocedor.
- preparar_conf.sh: Genera los ficheros de configuración necesarios para el toolkit.
- cmd.sh: Exporta las variables necesarias para realizar la ejecución.
- path.sh: Exporta los directorios y rutas (path) de la instalación de Kaldi. En este script también se tiene que indicar el directorio donde se encuentren todos los .wavs con los que va a trabajar el sistema y sus correspondientes .txt, donde los .txt definen los fonemas de cada palabra de cada frase que se pronuncia en cada .wav. Este directorio lo crea el script “preparar_datos.sh”.
- run.sh: Es el script que lanza la ejecución del sistema de RAH para la Centralita, generando el modelo acústico, el modelo de lenguaje mediante los datos de entrenamiento (train) y la evaluación resultante del sistema mediante los datos de test. Es el encargado de lanzar todos los scripts comentados en esta sección.

- mostrar_resultados.sh: Muestra los resultados (WER) obtenidos.

Los programas python2.7 situados en "Centralita/local" que se han implementado para generar los ficheros de configuración necesarios para realizar el reconocimiento con el toolkit Kaldi son ejecutados mediante los scritps "preparar_datos.sh" y "preparar_lenguaje.sh" cuya estructura se muestra en la siguiente tabla:

Scripts	**Programas que ejecuta + Ficheros Conf**
preparar_datos.sh	txt_corpus.py wav.scp.py text.py utt2spk.py corpus.py spk2gender
preparar_lenguaje.sh	lexicon.py nonsilence_phones.py silence_phones.txt optional_silence.txt

Tabla 1: Ficheros de configuración generados y programas python2.7 ejecutados por los scripts.

- txt_corpus.py: Genera los .txt de cada .wav tanto del train como del test a partir de las frases del directorio de transcripciones. Hay tantos .txt como .wav donde cada .txt contiene cada palabra de la transcripción de ese .wav con su correspondiente transcripción fonética. Directorios de salida: "Centralita/data/{train,test}"
  Ejemplo de fichero generado:
  training001.txt → páseme con el jefe de el señor marzal por favor

```
páseme        p a s e m e
con     k o n
el      e l
jefe    x e f e
de      d e
el      e l
señor   s e h o r
marzal  m a r z a l
por     p o r
favor   f a b o r
```

- wav.scp: Genera los ficheros wav.scp en "Centralita/data/{train,test}". Los ficheros wav.scp contienen en cada línea un id del .wav y la ruta (path) donde se encuentra ubicado el .wav.
  Ejemplo de fichero generado: "Centralita/data/train/wav.scp"

```
training001 /home/pascu/kaldi/kaldi/egs/Centralita/data/train/training001.wav
training002 /home/pascu/kaldi/kaldi/egs/Centralita/data/train/training002.wav
training003 /home/pascu/kaldi/kaldi/egs/Centralita/data/train/training003.wav
training004 /home/pascu/kaldi/kaldi/egs/Centralita/data/train/training004.wav
training005 /home/pascu/kaldi/kaldi/egs/Centralita/data/train/training005.wav
…
training149 /home/pascu/kaldi/kaldi/egs/Centralita/data/train/training149.wav
training150 /home/pascu/kaldi/kaldi/egs/Centralita/data/train/training150.wav
```

- text.py: Genera los ficheros text en “Centralita/data/{train,test}”. Los ficheros text contienen por cada fila el id del .wav y su transcripción.
  Ejemplo de fichero generado: “Centralita/data/train/text”

```
training001 páseme con el jefe de el señor marzal por favor
training002 desearía hablar con el uno nueve siete cinco
training003 me gustaría hablar con el jefe de montoliu
training004 quiero hablar con el jefe de granada por favor
training005 me pasaría con la extensión cuatro ocho uno
…
training149 quería hablar con el número nueve nueve ocho tres por favor
training150 quería hablar con el secretario de antonio amengual
```

- spk2gender: Este fichero de configuración lo genera directamente el script “preparar_datos.sh” en “Centralita/data/{train,test}”. Los ficheros spk2gender contienen el id del locutor y su género: m = masculino; f = femenino.
  En nuestro caso solo tenemos un locutor con id “FER”, pero si tuviésemos más de un locutor se especificarían asociando su id y su género.
  Ejemplo de fichero generado: “Centralita/data/train/spk2gender”

```
FER m
```

- utt2spk.py: Genera los ficheros utt2spk en “Centralita/data/{train,test}”. Los ficheros utt2spk contiene por cada fila el id del .wav y el locutor que realiza la pronunciación. En nuestro sistema solo tenemos un locutor “FER”, pero si tuviésemos más de un locutor se especificarían asociando qué .wav pronuncia cada locutor.
  Ejemplo de fichero generado: “Centralita/data/train/utt2spk”

```
training001 FER
training002 FER
training003 FER
training004 FER
training005 FER
…
training149 FER
training150 FER
```

- corpus.py: Genera el fichero “Centralita/data/local/corpus.txt” que contiene todas las transcripciones (train + test) que aparecen en el corpus.

```
páseme con el jefe de el señor marzal por favor
desearía hablar con el uno nueve siete cinco
me gustaría hablar con el jefe de montoliu
quiero hablar con el jefe de granada por favor
me pasaría con la extensión cuatro ocho uno
…
me pasa con la jefa de jorge porcar por favor
puede pasarme con el secretario de pepe granada por favor
me pasa con el secretario de maría vilar
me pasa con la secretaria de josé lópez
quería hablar con carlos castellanos por favor
```

- lexicon.py: Genera el fichero “Centralita/data/local/dict/lexicon.txt”. Este fichero contiene todas las palabras que aparecen en el corpus.txt con su correspondiente transcripción fonética.Se añaden dos palabras adicionales, el silencio y una palabra que representa las palabras desconocidas por el modelo (unknown).

```
!SIL sil
<UNK> spn
aibar a i b a r
garcía g a r z i a
secretaria s e k r e t a r i a
llopis y o p i s
varó b a r o
cuatro k u a t r o
quiero k i e r o
prat p r a t
la l a
...
```

- nonsilence_phones.py:
  Genera el fichero “Centralita/data/local/dict/nonsilence_phones.txt” que contiene todos los fonemas que tiene el corpus.txt.

```
p
a
s
e
m
k
o
n
l
x
```

```
f
d
h
r
z
b
i
u
t
g
C
y
R
```

- silence_phones.txt: Este fichero de configuración lo genera directamente el script "preparar_lenguaje.sh" en "Centralita/data/local/dict/silence_phones.txt".

```
sil
spn
```

- optional_silence.txt: Este fichero de configuración lo genera directamente el script "preparar_lenguaje.sh" en "Centralita/data/local/dict/optional_silence.txt".

```
sil
```

En el directorio "Centralita/local" encontramos dos scripts (en bash) que también son necesarios para hacer funcionar el sistema de RAH con el toolkit Kaldi, son los siguientes:

- Syllables.perl: Este script (en Perl) convierte una palabra de entrada en su transcripción fonética.
- score.sh: Este script (en bash) es proporcionado por Kaldi y se utiliza para obtener las puntuaciones en el proceso de decodificación del reconocimiento.

Ejecución

Para realizar la ejecución simplemente hay que copiar el directorio "Centralita" en el directorio de instalación de Kaldi "kaldi/egs/Centralita". Para lanzar la ejecución que se encarga de obtener los modelos (modelo acústico + modelo de lenguaje) y realizar la evaluación con la parte de test hay que ejecutar ./run.sh.
Para obtener los resultados obtenidos (WER) se ejecuta el script mostrar_resultados.sh.

```
%WER 1.18 [ 5 / 425, 0 ins, 3 del, 2 sub ] exp/mono/decode/wer_15
%WER 2.12 [ 9 / 425, 1 ins, 1 del, 7 sub ] exp/tri1/decode/wer_13
```

Tabla 2: Resultados obtenidos en el trabajo.

Se observa que se obtienen mejores resultados utilizando mono-fonemas que tri-fonemas, esto se debe a que el corpus utilizado en el trabajo no contiene un número considerablemente grande de frases, para casos en el que el corpus es de gran dimensión los tri-fonemas (por norma general) obtienen mejores resultados que los mono-fonemas.
Nota: Se han adjuntado las salidas por consola de las ejecuciones del script “run.sh” y “mostrar_resultados.sh”.

Aspectos a tener en cuenta

Una vez realizada la ejecución, Kaldi genera internamente de forma automática un conjunto de directorios y ficheros de configuración que aparecerán en el directorio “Centralita” y que no los hemos comentado en el trabajo, estos ficheros los utiliza para obtener el sistema de RAH y no se han comentado en el trabajo porque son ficheros internos de Kaldi que el desarrollador del sistema de RAH no tiene que eliminar ni modificar al implementar sus scripts.

Consideraciones Personales

El primer contacto con el toolkit Kaldi (opinión personal) me ha resultado complejo, como ya he comentado anteriormente utilizar los scripts proporcionados en las demos y adaptarlos al corpus con el que quieres trabajar es trivial, pero realizar todos los scripts desde cero y entender qué parte del fichero de configuración generado necesita Kaldi en cada momento es un proceso laborioso y costoso. Por otra parte cuando se produce un error en ejecución la información que te proporciona para arreglar dicho error es escasa y he necesitado tiempo y paciencia para obtener una ejecución correcta.
Desde mi punto de vista el esfuerzo ha valido la pena, ya que después de realizar este trabajo conozco una alternativa al toolkit HTK y puedo ser capaz de tomar decisiones en un futuro para escoger una de las dos opciones para implementar un sistema de RAH.