

Instituto tecnológico de Massachussets
Departamento de ingeniería eléctrica e informática

6.345 Reconocimiento del habla
Primavera 2003

Publicado: 21/03/03

Entregar: 09/04/03

Trabajo 7

Modelos gráficos para su tiempo de ocio

Introducción

Este trabajo le ofrecerá algunas ideas prácticas sobre el uso de las redes bayesianas (BN), abarcando un poco más y no centrándose sólo en el reconocimiento de voz. La parte I consta de ejercicios para realizar en papel; la parte II es la sección de “prácticas” de este trabajo, en el que utilizará el juego de herramientas de los modelos gráficos (GMTK) para realizar experimentos de reconocimiento de voz. Las partes I y II son independientes, pero probablemente le será más útil entrenarse con los ejercicios de la parte I como “calentamiento” para la parte II.

Como en anteriores trabajos, las siguientes tareas (señaladas con **T**) deberían acabarse durante la sesión de prácticas. Las respuestas a las preguntas (marcadas con **P**) deberían entregarse dentro de su correspondiente plazo.

NOTA: Para la parte práctica de este trabajo, será conveniente que trabaje en el laboratorio, donde podrá obtener ayuda directa de un monitor de prácticas si lo necesita. Esta es la primera vez que se va a ofrecer esta práctica, así que puede que haya aspectos mejorables en el futuro. Póngase en contacto con nosotros si tiene alguna duda, y no posponga la práctica para el final. Observe más adelante que antes de continuar, debe confirmar sus respuestas a la **P12** con un monitor de prácticas.

Al final de esta fotocopia se enumeran algunas referencias que pueden serle útiles.

Parte I: Ejercicios de calentamiento

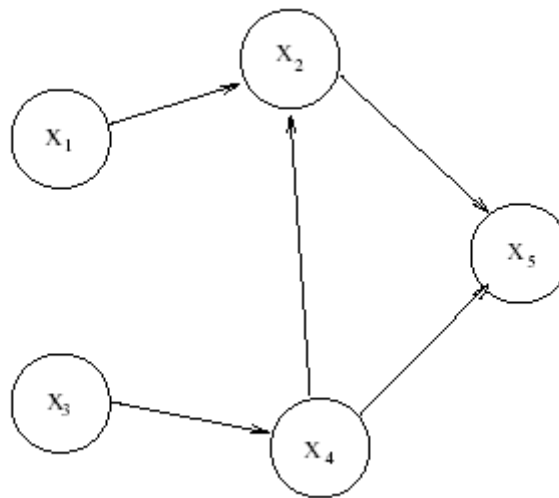
Notación: En esta fotocopia utilizamos una convención, según la cual las letras mayúsculas representan a las variables aleatorias, mientras que las minúsculas correspondientes representan

los valores de esas variables. Utilizamos $p(\cdot)$ como abreviatura para todos los tipos de funciones de probabilidad (funciones de masa de probabilidad (PMF) para variables discretas y

densidades (PDF) para variables continuas), y la convención según la cual los argumentos en minúscula de la función, identifican a las variables a las que ésta se refiere. Ej., $p(x, y)$ es la función PMF $P_{X,Y}(x, y)$, si X e Y son discretas, y es la PDF $f_{X,Y}(x, y)$ si son continuas; y $p(x|y)$ es $P_{X|Y}(x|y)$ si X es discreta, y $f_{X|Y}(x|y)$ si es continua.

Una red bayesiana simple

Las preguntas **P1–P3** se refieren a la siguiente red bayesiana:



P1: Escriba la forma factorizada de la probabilidad de unión $p(x_1, x_2, x_3, x_4, x_5)$ representada por este grafo. (Nota: Esto debería llevarle aproximadamente 5 segundos).

P2: Sea la notación $A \perp B$ la indicación de que la variable A es independiente de la variable B , y $A \perp B | C$ indica que A es independiente de B dada la variable C . Para cada una de las siguientes independencias, declare si esto queda o no implicado en el grafo anterior, y explique brevemente por qué se utiliza el algoritmo de la pelota de Bayes. Si la independencia *no* está implícita en el grafo, basta con dar una explicación de la trayectoria que la pelota podría seguir.

- (a) $X_1 \perp X_4$
- (b) $X_1 \perp X_4 | X_5$
- (c) $X_3 \perp X_5$
- (d) $X_3 \perp X_5 | X_2$

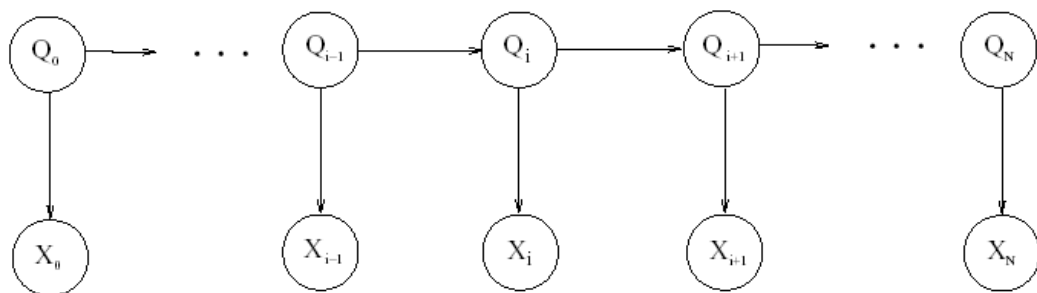
P3: Suponga que X_1, X_2, \dots, X_5 son variables binarias.

- (a) ¿Cuántos parámetros son necesarios para especificar totalmente la distribución de unión $p(x_1, \dots, x_5)$?
- (b) ¿Cuántos parámetros serían necesarios si existiese un flujo desde X_1 hasta X_5 ?
- (c) ¿Cuántos parámetros serían necesarios si no tuviésemos información sobre las relaciones de independencia entre las variables? Además, dibuje *dos* redes bayesianas para las variables X_1, \dots, X_5 , que representasen tal situación, y haga la factorización de $p(x_1, \dots, x_5)$ que está representada por cada una de sus redes bayesianas (BN).

P4: (Esta pregunta no está relacionada con la figura). Por definición, el grafo de una BN no debe tener ciclos dirigidos. Podemos comenzar a ver por qué esto es necesario, teniendo en cuenta qué ocurriría si eliminásemos esta restricción, y observando algunos grafos sin ciclos. Facilite un ejemplo de un grafo que contenga un ciclo dirigido, y muestre *matemáticamente* que tratar de interpretarlo mediante la semántica de una red bayesiana de factorización de probabilidad, da como resultado una contradicción.

Una red bayesiana dinámica

Las preguntas **P5–P8** hacen referencia a la red bayesiana dinámica (DBN) de abajo y a las tablas 1-3.



Suponga que en cada tramo, Q_i es discreta con cardinalidad 4, X_i es continua y unidimensional, y $p(x_i|q_i)$ es una mezcla de hasta M_{q_i} gaussianas. Ej.,

$$p(x_i|q_i) = \sum_{m=1}^{M_{q_i}} w_{m,q_i} \mathcal{N}(x_i; \mu_{m,q_i}, \sigma_{m,q_i}^2) \tag{1}$$

P5: Suponga que esta DBN se “descomponen” en tres tramos (ej., $N = 2$). Escriba la expresión factorizada para $p(x_0, x_1, x_2, q_0, q_1, q_2)$ representada por este grafo.

P6: Esta DBN representa un modelo oculto de Markov. Para cada una de las tablas de probabilidad condicional en las tablas 1-3, trace el diagrama de transición de estado para el modelo HMM al que corresponde.

P7: Para una de las tablas anteriores, existe una DBN más sencilla que podría representar la misma distribución. ¿Cuál es la tabla, y que aspecto tendría la DBN simplificada en este caso?

P8: Suponga que quisiésemos implementar una DBN utilizando un paquete de software en el que las variables continuas pudiesen tener sólo distribuciones gaussianas (ej., las mezclas de gaussianas no son posibles).

Describa cómo, aún así, podemos representar la misma distribución de unión sobre

$X_1, \dots, X_N, Q_1, \dots, Q_N$ como nuestra DBN original, añadiendo una variable más al grafo (y todas las dependencias necesarias).

Trace la DBN modificada resultante, y especifique la cardinalidad de la nueva variable, su función de probabilidad local y las funciones de probabilidad local de cualquiera de sus hijos, en función de las cantidades ya definidas.

| $q_{i-1} \backslash q_i$ | 0 | 1 | 2 | 3 |
|--------------------------|-----|-----|-----|-----|
| 0 | 0.2 | 0.4 | 0.1 | 0.3 |
| 1 | 0.2 | 0.4 | 0.1 | 0.3 |
| 2 | 0.2 | 0.4 | 0.1 | 0.3 |
| 3 | 0.2 | 0.4 | 0.1 | 0.3 |

Tabla 1: $p(q_i | q_{i-1})$, versión 1.

| $q_{i-1} \backslash q_i$ | 0 | 1 | 2 | 3 |
|--------------------------|-----|-----|-----|-----|
| 0 | 0.9 | 0.1 | 0 | 0 |
| 1 | 0 | 0.9 | 0.1 | 0 |
| 2 | 0 | 0 | 0.9 | 0.1 |
| 3 | 0 | 0 | 0 | 1 |

Tabla 2: $P(q_i | q_{i-1})$, versión 2.

| $q_{i-1} \backslash q_i$ | 0 | 1 | 2 | 3 |
|--------------------------|-----|-----|-----|-----|
| 0 | 0.5 | 0 | 0 | 0.5 |
| 1 | 0.2 | 0 | 0.4 | 0.4 |
| 2 | 0 | 0.3 | 0 | 0.7 |
| 3 | 0 | 0 | 1 | 0 |

Tabla 3: $P(q_i | q_{i-1})$, versión 3.

Parte II: Utilización del juego de herramientas GMTK para el reconocimiento de voz en Aurora

Tarea

En esta parte, realizará experimentos de reconocimiento de voz utilizando el juego de herramientas para modelos gráficos (GMTK)¹ sobre un pequeño subconjunto del corpus Aurora 2.0, que contiene cadenas de dígitos conectados con varias cantidades y tipos de ruido añadido (ruido subterráneo, murmullo, etc.) Aprenderá algunos de los características del juego de herramientas y lo utilizará para representar, entrenar y poner a prueba a los reconocedores.

Nota: En esta fotocopia, los comandos que usted debería escribir están impresos como líneas separadas con la fuente `typewriter`, con un % para indicar la ventana de símbolo del sistema de UNIX.

Cuando un comando se extiende más allá del ancho de la página, se imprime en líneas múltiples, aunque usted lo debiera escribir como uno.

Cómo prepararse

¹ J. Bilmes y G. Zweig, "The Graphical Models Toolkit: An Open Source Software System for Speech and Time Series Processing". *IEEE International Conference on Acoustics, Speech, and Signal Processing*, junio 2002, Orlando, Florida.

Para prepararse para esta práctica, escriba lo siguiente en la ventana de símbolo del sistema de UNIX:

```
% start_lab7.cmd
% cd lab7
```

`start_lab7.cmd` creará un nuevo subdirectorio bajo su directorio local llamado `lab7`, y lo rellenará con los archivos necesarios para esta práctica. Estos archivos contienen datos, modelos y especificaciones del parámetro utilizados durante la práctica. Las descripciones de los archivos se encuentran principalmente como comentarios dentro de los mismos archivos. En la primera parte de la práctica, observaremos detalladamente algunos de estos archivos para conocer un poco sobre cómo el juego de herramientas GMTK representa a las estructuras de DBN y a los parámetros.

Lo esencial del juego de herramientas GMTK

Comenzaremos con algunos antecedentes breves para que usted se familiarice con las características del juego GMTK que le harán falta (y quizás con un par que no necesitará, pero que es conveniente que conozca). Mientras vemos estas características, entrenaremos y probaremos un reconocedor de base. Para ello, usted realizará algunas modificaciones en el reconocedor, y ejecutará experimentos con distintas configuraciones.

En GMTK, la estructura de un modelo (ej., las variables y dependencias) está definida en un *archivo de estructura*. El archivo de estructura también contiene ciertos atributos de cada variable (ocultos frente a observados, discretos frente a continuos, etc.), junto con un nombre y tipo para la función de probabilidad local de cada variable (gaussiana, tabla de probabilidad, determinístico, etc.). Luego técnicamente, proporciona en realidad, más que la simple estructura del modelo. Los parámetros para cada probabilidad local se especifican en *archivos de parámetro* individuales. Nos centraremos en cada tipo de archivo por separado. Los dos principales programas del juego GMTK que es necesario que conozca, son `gmtkViterbi` y `gmtkEMtrain`. `gmtkViterbi` incluye una estructura, parámetros y *archivos de observación*, facilitando los valores de todas las variables observadas, computa los entornos de máxima probabilidad de las variables ocultas e imprime los valores de las variables de interés (la cual, en reconocimiento de voz, es normalmente la palabra). `gmtkEMtrain` incluye una estructura, entornos del parámetro inicial y archivos de observación que contienen comentarios del entrenamiento, y ejecuta el algoritmo de EM para hallar los entornos de probabilidad máxima de los parámetros. En la mayor parte de esta práctica, usted no tendrá que ejecutar directamente `gmtkEMtrain`, sino un script envoltorio que utiliza `gmtkEMtrain` para llevar a cabo el entrenamiento en paralelo en múltiples máquinas.

Archivos de estructura

Las estructuras de DBN que utilizaremos para entrenar y decodificar un reconocedor Aurora de base, están definidas en `STRUCTURES/[training,decoding].str`. Este reconocedor de base está basado en modelos HMM, con la variable fono como estado del HMM.

Lea los archivos de estructura para familiarizarse con la sintaxis de especificación de GMTK y las estructuras concretas con la que estamos trabajando. La estructura de

decodificación es la misma que la que se mostró en clase (observe las transparencias del tema en la página web del curso). La estructura del entrenamiento es un poco distinta, ya que debe tener en cuenta el hecho de que durante el entrenamiento conocemos la cadena de la palabra (pero no los tramos que van con cada palabra). Observe primero el archivo de estructura de decodificación, ya que introduce la mayoría de la sintaxis.

P9: Trace la estructura de la DBN de entrenamiento, incluyendo todos los tramos/variables que están en el archivo de estructura. En su diagrama, utilice nodos sombreados para las variables observadas, y flechas punteadas para indicar los padres cambiantes. Es posible que le sea más fácil comenzar con la estructura de decodificación de las transparencias del tema y modificarla tal y como requiera. Observe que la variable de observación es multidimensional, pero en esta estructura sigue siendo tratada como una variable simple; simplemente parece ser una variable vector (en este caso, de valores MFCC). Indíquela en el grafo como un nodo simple. Señale también, para cada variable de cada tramo, (ya se encuentre cerca a su nodo en el grafo o en una lista separada), si ésta es discreta o continua, y de ser discreta, si es o no determinística.

(Por cierto, esto es sólo un ejercicio de lectura del archivo. Sin embargo, le será más fácil después referirse a su dibujo que al archivo).

P10: Si esto es un modelo, ¿por qué entonces este modelo no tiene sólo dos variables, un estado y una observación? ¿Podríamos haber implementado la misma funcionalidad utilizando un modelo con sólo estas dos variables? Si es así, ¿cómo?

Archivos del parámetro

En GMTK, las variables continuas pueden tener distribuciones gaussianas o de mezclas de gaussianas. La probabilidad de una variable discreta se facilita a través de tablas de probabilidad condicional (CPT). Por lo general, una CPT es simplemente una tabla (multidimensional) de valores de probabilidad, una fila por combinación de valores padres.

Los parámetros de gaussianas y las tablas de probabilidad pueden entrenarse mediante EM y se almacenan en un *archivo de parámetros entrenables*.

Para nuestro reconocedor de base, el archivo inicial del parámetro entrenable que contiene los parámetros que utilizará para inicializar el entrenamiento de EM, se encuentra en `PARAMS/flat_start.gmp`. Examine este archivo, que también está documentado con comentarios. En este caso, estamos inicializando todas las gaussianas a $\mathcal{N}(0, 10)$, y todas las tablas de probabilidad a probabilidades uniformes.

Para el caso especial de las variables *determinísticas*, ej., aquellas que son sólo funciones de sus padres, se puede especificar una “CPT determinística”, que simplemente hace asociaciones desde una combinación de valores padres hasta un valor simple utilizando un árbol de decisión. (Por supuesto, uno siempre puede utilizar una CPT no determinística para una variable determinística, pero la tabla sería casi todo ceros). En un árbol de decisión como este, cada uno de los puntos de las ramas pregunta el valor de uno de los padres de la variable, y según eso, desciende por la rama adecuada. Cada trayectoria del árbol de decisión corresponde a alguna combinación de valores de las variables del padre. Los nodos hoja del árbol, ej., las “respuestas” facilitan los valores de la variable hijo para las correspondientes trayectorias que descienden del árbol. Los “parámetros” para las variables determinísticas, ej., las

especificaciones de sus CPT determinísticas y los árboles de decisión que utilizan, se facilitan en un archivo individual. Los parámetros fijos para el reconocedor de base se encuentran en `PARAMS/masterFile.template`. De nuevo, la sintaxis del archivo se proporciona a través de comentarios en el mismo archivo. (Se denomina “archivo maestro” porque presenta de hecho más capacidades, aunque nosotros no las utilizaremos; además, se llama “plantilla” sólo porque se utilizará para generar archivos múltiples durante el entrenamiento paralelo).

Listas de archivos, archivos de rasgos, formas de onda

En estos experimentos, utilizará un grupo de entrenamiento de 100 enunciados, compuestos de dos subgrupos de 50 enunciados cada uno. Uno de los subgrupos consta de grabaciones limpias y el otro presenta ruido subterráneo añadido a una proporción señal a ruido (SNR) de 5 dB. Los grupos de pruebas presentan también 50 enunciados en cada nivel de ruido.

Los archivos `train.filelist`, `test.clean.filelist` y `test.SNR5.filelist` contienen listas de enunciados en los que se almacenan las observaciones acústicas (en este caso, los coeficientes MFCC más sus derivadas y segundas derivadas). El archivo `train.wavlist` posee una lista de archivos de la forma de onda en el grupo de entrenamiento. Si tiene curiosidad, puede escuchar unos cuantos de esta clase para captar un sentido del tipo de discurso que estamos tratando junto con la diferencia entre los niveles de ruido. Puede reproducir un archivo de forma de onda ejecutando:

```
% waveform_play -normalize <filename>
```

Entrenamiento del reconocedor de base

T1: Realice 4 iteraciones del algoritmo de EM en la estructura de base ejecutando:

```
% emtrain_parallel header.emtrain.baseline
```

`emtrain_parallel` ejecuta cada iteración en múltiples máquinas en paralelo utilizando el servicio de envío del host del grupo SLS. El algoritmo EM se paraleliza dividiendo los pasos “E” y “M”; el paso “E” se realiza dividiendo el grupo de entrenamiento en un número de partes más pequeñas y computando la estadística de las partes en distintas máquinas en paralelo, mientras el paso “M” reúne toda la estadística (en una sola máquina) y lleva a cabo la maximización.

El argumento a `emtrain_parallel` es un archivo cabecera en el que especificamos varios parámetros requeridos para el entrenamiento, como las ubicaciones de los diversos archivos, el número de iteraciones y demás.

Como se indicó en el archivo cabecera, los parámetros finales se escribirán en `PARAMS/learned_params.baseline.gmp`.

Cuando se ejecute un trabajo paralelo en el envío del host, puede obtener alguna información sobre el estado de todos los hosts, junto con sus actuales trabajos, ejecutando `phd -status`. La vuelta de entrenamiento debería durar de uno a unos cuantos minutos, dependiendo de cómo es estén ejecutando al mismo tiempo muchos otros trabajos.

T2: Utilice el reconocedor de base para decodificar los enunciados en los dos grupos de prueba ejecutando:

```
% gmtkViterbi -of1 test.clean.filelist -strF STRUCTURES/decoding.str
-inputMaster PARAMS/masterFile.decode -nf1 42 -fmt1 ascii
-inputTrainable PARAMS/learned_params.baseline.gmp -beam 500
-printWordVar word -varMap word_map -trans wordTransition | tee
OUT/clean.baseline.out
```

```
% gmtkViterbi -of1 test.SNR5.filelist -strF STRUCTURES/decoding.str
-inputMaster PARAMS/masterFile.decode -nf1 42 -fmt1 ascii
-inputTrainable PARAMS/learned_params.baseline.gmp -beam 500
-printWordVar word -varMap word_map -trans wordTransition | tee
OUT/SNR5.baseline.out
```

Estos comandos también están guardados en el archivo `decode commands.baseline`, así que no tendrá que escribir todo lo anterior a mano. Para recopilar más información sobre los argumentos de la línea de comando, escriba `gmtkViterbi` sin ningún argumento.

La decodificación se realiza localmente (ej., en su máquina y no en paralelo), y cada vuelta de la decodificación debería tardar aproximadamente un minuto.

La salida irá a la pantalla y a los archivos `OUT/[clean,SNR5].baseline.out`.

Aquí tiene un ejemplo de qué aspecto tendría la salida para dos enunciados:

```
Example prob: -31068.1 : 166 frames
sil (0-32)
five (33-62)
nine (63-99)
seven (100-136)
sp (137-165)
Example prob: -35347 : 187 frames
sil (0-20)
one (21-42)
six (43-84)
sil (85-103)
three (104-135)
oh (136-158)
sil (159-186)
```

La primera línea facilita la probabilidad logarítmica total de las observaciones acústicas dado el modelo, así como el número de tramos de cada enunciado. Las líneas restantes muestran la cadena de la palabra decodificada, junto con los números de tramo correspondientes a cada palabra entre paréntesis. En el primer enunciado de arriba, la primera palabra decodificada es silencio, y se extiende desde el tramo 0 hasta el 32; la segunda es “*five*”(cinco), desde el tramo 33 hasta el 62; y así sucesivamente. Mida las tasas de error de estos grupos ejecutando:

```
% score_aurora_SubstnsDel OUT/clean.baseline.out test.clean.filelist
word_list
```

```
% score_aurora_SubstnsDel OUT/SNR5.baseline.out test.SNR5.filelist
word_list
```

Esto imprimirá información sobre las tasas de sustitución, inserción y eliminación, la tasa de error total por palabra, y los cálculos detallados de cada tipo de error.

T3: Entrene también reconocedores individuales para las dos condiciones de ruido, ej., entrene un reconocedor con sólo 50 enunciados limpios y uno con 50 enunciados con ruido:

```
% emtrain_parallel header.emtrain.clean
```

```
% emtrain_parallel header.emtrain.SNR5
```

Los parámetros entrenados se guardarán en `PARAMS/learned_params.clean.gmp` y `PARAMS/learned_params.SNR5.gmp` respectivamente.

T4: Como antes, pruebe cada uno de los reconocedores en los grupos de prueba limpios y en los de SNR5. Los comandos que tendría que ejecutar para esto están en el archivo `decode_commands.separate`.

P11: Tabule las tasas de error por palabra que midió en las tareas **T2** y **T4**. ¿Funciona mejor el reconocedor cuando se decodifican los enunciados con la misma SNR que cuando fue entrenado, o cuando es entrenado con el grupo de entrenamiento mixto? ¿Era inevitable el resultado? ¿Por qué o por qué no? (Observe que esta pregunta no tiene nada que ver concretamente con los modelos gráficos).

Adición de una variable de ruido

Querríamos ahora modificar la estructura de base, de modo que se pudiese modelar el hecho de que cada enunciado fuese limpio o ruidoso, utilizando distintos modelos de mezcla de gaussianas para cada condición. Nos gustaría hacer esto añadiendo una variable binaria llamada `noise`, que tomará el valor 0 para los enunciados limpios, y 1 para los enunciados con ruido.

La observación dependerá ahora *tanto* del estado del fono como del valor del ruido, ej., tendrá una mezcla de gaussianas distinta para cada combinación del estado del fono y del valor del ruido. Durante el entrenamiento, la variable ruido será observada (dado que sabemos los enunciados de entrenamiento que pertenecen a cada condición de ruido); se ocultará durante la fase de pruebas. Esto puede considerarse una forma de clasificación implícita del ruido, ya que el valor de la variable ruido durante la decodificación nos indicará a qué condición de ruido, según el reconocedor, corresponde el enunciado. Supondremos que este nuevo reconocedor será probado con datos de prueba con la misma distribución de enunciados limpios frente a enunciados con ruido como grupo de entrenamiento.

Un nuevo archivo de parámetros entrenable, `PARAMS/flat_start.noise.gmp`, se ha facilitado para su utilización con este nuevo modelo; dado que existen ahora dos mezclas de gaussianas individuales por estado de fono, este archivo de los parámetros

presenta el doble de mezclas de gaussianas, componentes de gaussianas, medias, etc. como `PARAMS/flat_start.gmp`.

El archivo `train.noise.filelist` contiene una lista de archivos con “observaciones” del nivel de ruido para cada una de los enunciados de entrenamiento. Para un enunciado limpio, su archivo de observación simplemente contiene un “0” por cada tramo; para un enunciado con ruido, contiene un “1” para cada tramo. (Teóricamente, no deberíamos necesitar un valor separado para cada tramo, ya que sabemos que la condición de ruido es la misma por todo el enunciado, pero el GMTK requiere que todos los tramos tengan el mismo número de observaciones).

P12: Dibuje un nuevo par de estructuras de entrenamiento y decodificación que representen este nuevo modelo, aumentando los grafos de entrenamiento de base/decodificación con la variable ruido y cualquier dependencia adicional necesaria. Especifique también cuáles son las funciones de probabilidad local para la variable ruido y para cualquier otra variable cuyas probabilidades locales sean distintas en la nueva estructura.

(Hemos dicho ya, por ejemplo, que la variable de observación dependerá del estado del fono y del ruido). Para cualquier variable determinística, puede especificar la probabilidad local como una regla o función (ej., “ $v = \text{el valor del padre}$ ”, o “ $v = 0$ si el padre es 1, y si no”).

Es conveniente que confirme la respuesta a esta pregunta con un monitor de prácticas antes de continuar, ya que lo próximo será la implementación de este modelo.

T5: Desearíamos modificar ahora los archivos necesarios para implementar este nuevo modelo. Usted ha proporcionado archivos parciales que tendrá que completar. No se desanime si este paso le lleva mucho tiempo. Sea cuidadoso y paciente, y si se queda atascado, no dude en pedir ayuda.

En primer lugar, eche un vistazo a `STRUCTURES/training.noise.str`. La variable de observación ha sido ya modificada adecuadamente para el nuevo modelo, y se ha incluido el esqueleto de una nueva variable llamada “ruido”. Será necesario que rellene el resto de la especificación para la variable ruido en cada tramo. Puede nombrar cualquier función de probabilidad local del modo que desee, siempre que sea coherente. Luego, será necesario que modifique el archivo inicial de parámetros entrenables `PARAMS/flat_start.noise.gmp` y/o los parámetros no entrenables en `PARAMS/masterFile.noise.template`. Ambos archivos incluyen ya las modificaciones necesarias para la variable de observación (nuevas gaussianas y un árbol de decisión para asociar el estado del fono y el ruido con la correspondiente mezcla de gaussiana). Necesitará añadir cualquier otra CPT nueva a estos archivos en los lugares apropiados. Recuerde que debe utilizar los mismos nombres para las nuevas CPT, como especificó en el archivo de estructura. Además, no olvide incrementar el número de tablas CPT cuando añada una nueva.

Finalmente, tendrá que hacer lo mismo con los archivos de decodificación `STRUCTURES/decoding.noise.str` y `PARAMS/masterFile.noise.decode`. Cuando crea que haya terminado de hacer los cambios necesarios, sería conveniente que se asegurara de que el modelo se ejecuta. Para probar el modelo de entrenamiento, ejecute:

```
% convert_masterfile.pl < PARAMS/masterFile.noise.template >
PARAMS/masterFile.noise

% gmtkEMtrain -of1 train.local.filelist -nf1 42 -ni1 0 -fmt1 ascii
-of2 train.noise.local.filelist -nf2 0 -ni2 1 -fmt2 ascii
-inputMaster PARAMS/masterFile.noise -inputTrainable
PARAMS/flat_start.noise.gmp -binInputTrainable false -strFile
STRUCTURES/training.noise.str -maxE 1 -trrng 0:1 -baseCaseT 100000
```

También puede encontrar estos comandos en `train command`, y no tendrá que escribirlos. Con esto ejecutará `gmtkEMtrain` en dos enunciados para una iteración. La última línea de salida debería incluir “*PROGRAM ENDED SUCCESSFULLY WITH STATUS 0*” (programa finalizado con éxito con estado 0). Si hay un error, vuelva a atrás y fije sus archivos de parámetro y estructura y repita. Para probar el modelo de decodificación, ejecute:

```
% gmtkViterbi -of1 test.clean.filelist -strF
STRUCTURES/decoding.noise.str -inputMaster
PARAMS/masterFile.noise.decode -nf1 42 -fmt1 ascii
-inputTrainable PARAMS/flat_start.noise.gmp -beam 500
-printWordVar word -varMap word_map -trans wordTransition
-baseCaseT 100000 -dcdrng 0:1
```

(O, de forma equivalente, `decode command`). Con esto se ejecutará `gmtkViterbi` en dos enunciados, utilizando los parámetros iniciales. La finalización con éxito queda de nuevo indicada en la frase de arriba.

T6: Una vez que su modelo se ejecuta, entrénelo con 100 enunciados de entrenamiento como antes. Se ha facilitado un nuevo archivo de cabecera que utiliza los archivos de parámetro y estructura. Para entrenar al reconocedor, ejecute:

```
% emtrain_parallel header.emtrain.noise
```

Pruebe también el funcionamiento del reconocedor con dos grupos de prueba, utilizando los comandos de `commands.noise`. Puntúe las salidas utilizando:

```
% score_aurora_SubstnsDel OUT/clean.noise.out test.clean.filelist
word_list

% score_aurora_SubstnsDel OUT/SNR5.noise.out test.SNR5.filelist
word_list
```

P13: ¿Cuáles son las tasas de error por palabra en ambos grupos? ¿De qué modo se comparan estos con sus anteriores resultados? Trate de explicar cualquier diferencia entre el funcionamiento de este modelo y aquel de los modelos entrenados en cada condición de ruido por separado. (Pista: Piense en cómo se diferencia este modelo del hecho de tener un modelo *completo* individual para cada condición).

T7: Para probar el buen funcionamiento del nuevo reconocedor al clasificar los enunciados de prueba como limpios o con ruido, ejecute los comandos `decode commands.noise classify`. La salida para cada enunciado será ahora el valor de la variable ruido durante cada palabra (que debería ser la misma para todas las palabras del enunciado). La salida se guardará en `OUT/[clean,SNR5].noise class.out`.

P14: ¿Cuántos errores de clasificación de ruido comete su modelo en cada condición?

P15: ¿Podríamos haber implementado el mismo modelo utilizando un reconocedor basado en modelos HMM? Si es así, ¿cómo? Si no, ¿por qué no? (Por “mismo modelo”, queremos decir aquel que siempre se comportaría exactamente igual; en concreto, debería dar las mismas salidas de decodificación).

Adición de una variable auxiliar oculta

Ahora, nos gustaría averiguar qué ocurriría si añadiéramos una variable binaria al reconocedor de base como antes, pero sin “decir” al reconocedor que esta variable adicional supuestamente representaba el nivel de ruido. Es decir, querríamos mantenerla oculta durante el entrenamiento y las pruebas, y permitir al algoritmo EM entrenar a los parámetros a su modo.

T8: Repita la **T5**, esta vez para un modelo que implemente esta nueva variable auxiliar. Rellene los archivos de estructura de decodificación y entrenamiento

```
STRUCTURES/[training,decoding].aux var.str,  
el archivo de parámetros entrenables PARAMS/flat start.aux var.gmp y el  
archivo de parámetros no entrenables PARAMS/masterFile.aux  
var.[template,decode]. Observe que la estructura de decodificación y los  
parámetros pueden ser exactamente iguales que en la T5. De hecho, si lo desea puede  
simplemente copiar su STRUCTURES/decoding.noise.str y  
PARAMS/masterFile.noise.decode, cambiando el nombre de la variable de  
“ruido” a “aux”, y asegurándose de que los nombres de la tabla CPT son coherentes en  
el entrenamiento y en la codificación. No es necesario que entregue un diagrama de la  
nueva estructura de entrenamiento, pero puede resultarle útil dibujar una antes de editar  
el archivo de estructura.
```

Puede probar de nuevo que los nuevos modelos se ejecutan antes del entrenamiento, utilizando `train command.aux var` y `decode command.aux var`.

T9: Entrene el modelo utilizando :

```
% emtrain_parallel header.emtrain.aux_var
```

Decodifique con el modelo entrenado utilizando `decode commands.aux var` y mida las tasas de error utilizando:

```
% score_aurora_SubstnsDel OUT/clean.aux_var.out test.clean.filelist  
word_list
```

```
% score_aurora_SubstnsDel OUT/SNR5.aux_var.out test.SNR5.filelist  
word_list
```

Observe también los valores de la variable auxiliar para cada enunciado utilizando `decode commands.aux classify`. Las salidas estarán en `OUT/[clean,SNR5].aux class.out`. El formato será el mismo, excepto que esta vez los valores de la salida serán 0 o 1, en vez de “limpio” o “con ruido”.

P16: ¿Es posible que esta nueva variable auxiliar haya aprendido a asociarse con el nivel de ruido? ¿A qué corresponde al valor auxiliar de “0”, y a qué corresponde el “1”? ¿Qué piensa usted que ocurriría si ejecutásemos el mismo experimento, pero utilizando sólo discurso limpio como datos de entrenamiento (ej., ¿cómo podrían obtenerse los valores de la variable auxiliar en ese caso)?

P18: (Puntos extra)

(a) Si observa las medias de gaussianas en `flat start.aux var.gmp`, se dará cuenta de que no son uniformes (como eran las que se utilizaban para el modelo de base y para el modelo con la variable de ruido observada).

Este archivo de los parámetros se inicializaba con valores aleatorios y no con uniformes. ¿Puede explicar por qué es esto necesario en este caso?

Es posible que quiera intentar entrenar este reconocedor con una inicialización uniforme (modificando `flat start.aux var.gmp` para utilizar las medias de gaussianas desde `flat start.noise.gmp`), y ver qué ocurre.

(b) Basándose en su respuesta a la parte (a), ¿puede explicar cualquier diferencia entre las tasas de error por palabra que encontró en **T9**, utilizando la variable auxiliar oculta, y en **T6**, utilizando la variable de ruido?

Referencias bibliográficas

Las siguientes lecturas pueden serle útiles. Existen enlaces a éstas en la página web del curso.

[1] J. Bilmes, “Graphical Models and Automatic Speech Recognition”, en *Mathematical Foundations of Speech and Language Processing, Institute of Mathematical Analysis Volumes in Mathematics Series*, Springer-Verlag, 2003.

Este artículo ofrece una visión global de los modelos gráficos en general, además de su uso en reconocimiento de voz. Las secciones más relevantes, para nuestro fin, son las secciones 1, 2.1-2.5 (aunque puede saltarse los detalles de la inferencia en la sección 2.5), 3.4, 3.9 y 4-6.

[2] G. Zweig, *Speech Recognition with Dynamic Bayesian Networks*. Tesis doctoral, U.C. Berkeley, 1998.

Esta tesis desarrolla el uso de redes DBN para el reconocimiento de voz. El capítulo 6 es el capítulo principal que trata cómo pueden configurarse las redes DBN para el reconocimiento de voz.

[3] J. Bilmes, “What HMMs can do.” UWEETR-2002-0003 (Informe técnico de la universidad de Washington), febr 2002.

Este artículo trata con mayor profundidad el por qué los modelos más allá de los HMM son necesarios para el reconocimiento de voz.

Muestra que los modelos HMM son extremadamente potentes en la teoría, pero que otros modelos pueden ser mejor bajo las limitaciones del mundo real de los datos finitos y la computación. Esta lectura no es obligatoria, pero podría resultarle interesante.