

**Instituto tecnológico de Massachussetts**  
**Departamento de ingeniería eléctrica e informática**

6.345 Reconocimiento automático del habla  
Primavera 2003

Publicado: 14/03/03  
Entregar: 02/04/03

---

**Tarea 6**  
**Modelado del lenguaje**

---

## Introducción

El objetivo de esta práctica es mejorar su comprensión del modelado del lenguaje. Esta práctica le permite familiarizarse con aspectos del modelado de  $n$ -grama, utilizando herramientas del sistema de reconocimiento de voz SUMMIT. Esta práctica y la nº 9 (Reconocimiento automático del habla basado en segmentos) le ayudará a aprender los procedimientos implicados en la construcción de los diversos componentes de SUMMIT. Estas destrezas le resultarán muy útiles más tarde cuando esté ensamblando un reconocedor de base para los experimentos de su proyecto de fin de curso. Utilizaremos datos del entorno **Pegasus**, que contiene preguntas sobre información de salidas y llegadas de vuelos.

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

## Parte I : Modelado del lenguaje $N$ -grama

### Introducción

En esta parte de la práctica entrenaremos y evaluaremos varios modelos de lenguaje estadísticos  $n$ -grama. Concretamente, nos centraremos en lo siguiente:

- entrenar modelos de lenguaje  $n$ -grama de palabras.
- entrenar modelos de lenguaje  $n$ -grama de clase
- medir la perplejidad de los modelos de lenguaje en el entrenamiento y desarrollo de los datos.

### Software

Utilizaremos un juego de programas que compongan el sistema de reconocimiento SUMMIT. Usaremos la versión 3.7 del sistema que se especifica al ajustar la variable de entorno SLS\_HOME to `/usr/sls/sls/v3.7`. Observe que esto se lleva a cabo automáticamente en el archivo `.cshrc` de los cómputos de la clase.

Al final de esta fotocopia aparece un listado descriptivo de los programas utilizados en esta práctica, facilitado en el apéndice de **herramientas de SUMMIT**. Revise la descripción de los programas antes de iniciar la práctica.

## Cómo empezar

Para comenzar esta práctica, escriba el siguiente comando en la ventana de símbolo del sistema UNIX:

```
% start_lab6.cmd
```

Esto creará un nuevo subdirectorio bajo su directorio local llamado `lab6`, y lo llenará con el grupo de archivos que son necesarios para esta práctica. Estos archivos contienen datos, modelos y especificaciones del parámetro utilizados durante la práctica. Se pueden encontrar descripciones de los archivos en el apéndice de **archivos de SUMMIT** al final de esta fotocopia. Realice esta práctica dentro del directorio `lab6`.

## Modelos de lenguaje $n$ -grama

Para entrenar un modelo de lenguaje estadístico  $n$ -grama de palabras, es necesario que especifiquemos la lista de palabras del vocabulario y un conjunto de transcripciones ortográficas o de nivel de palabra de las oraciones de entrenamiento. El archivo `pegasus.vocab` contiene todas las palabras del vocabulario, clasificadas en orden alfabético. También contiene:

1. El marcador de límites de oración, `<>*`,
2. la palabra desconocida, `<unknown>*`,
3. `<pause1>` que aparece al comienzo de la oración,
4. `<pause2>` que aparece al final de la oración.

Entrenará los modelos de lenguaje utilizando oraciones de entrenamiento del archivo `pegasus.sents` y evaluando estos modelos en oraciones del grupo de desarrollo en `dev.sents`.

**T1:** Cree un archivo de léxico de palabras (`.wlex`) a partir del archivo de vocabulario `pegasus.vocab`, usando el programa `wlex_create` como se muestra a continuación:

```
% wlex_create -in pegasus.vocab -out pegasus.wlex
```

Use el programa `ngram_create` para entrenar un modelo de lenguaje **bigrama** suavizado de este modo:

```
% ngram_create -wlex pegasus.wlex -in pegasus.sents \  
-n 2 -out pegasus.bigram \  
-set_log_prob 50 50 0
```

Entrene un modelo de lenguaje **trigrama** suavizado del siguiente modo:

```
% ngram_create -wlex pegasus.wlex -in pegasus.sents \  
-n 3 -out pegasus.trigram \  
-set_log_prob 50 50 0
```

La opción `_set_log_prob` especifica parámetros para su modelo de lenguaje. El primer número es  $K$  que controla el parámetro de interpolación  $\lambda$ :

$$\lambda = \frac{c(h_i)}{c(h_i) + K} \quad (1)$$

El segundo número determina el cómputo mínimo de símbolos utilizados en el unigrama. El tercer número especifica si la palabra `<unknown>` será ignorada. Hemos preferido incluirlo en nuestro caso, de modo que la probabilidad de la palabra `<unknown>` será mayor que cero.

Puede observar todas las opciones y sus explicaciones mediante:

```
% ngram_create -help
```

## Perplejidad

Un modo cuantitativo de evaluar consiste en computar una medida de información teórica, la *perplejidad* del modelo sobre un nuevo grupo de oraciones. El programa `ngram_perplexity` puede utilizarse para esto.

**T2:** Mida la perplejidad de los modelos bigrama y trigrama en las oraciones del grupo de desarrollo (`dev.sents`). Por ejemplo, para computar la perplejidad del modelo de lenguaje trigrama de palabras, podemos decir:

```
% ngram_perplexity -ngram pegasus.trigram -in dev.sents
```

Pruebe varios modelos para el parámetro de suavizado del modelo trigrama cambiando el primer argumento de `_set_log_prob` en `ngram_create`. Halle un parámetro de suavizado que reducirá la perplejidad del grupo de desarrollo de las  $n$ -gramas de base.

**P1:** (a) Facilite un trazado de perplejidad (en `dev.sents`) frente a un parámetro de suavizado para los modelos bigrama y trigrama.

(b) ¿De qué modo se pueden comparar las mediciones de perplejidad de los modelos del trigrama con las de los modelos de bigrama? ¿Qué modelo es mejor?

(c) ¿Cómo afecta el suavizado a la perplejidad de las oraciones de desarrollo?

**T3:** Podemos también hacer caso omiso de la pausa al comienzo y fin de las oraciones como se muestra:

```
% ngram_perplexity -ngram pegasus.trigram -in dev.sents \  
-discount_pauses
```

Además, podemos evaluar la probabilidad y perplejidad de cualquier secuencia de palabras mediante `ngram_print_sentence_score`. Puede especificar cualquier secuencia de palabras en la entrada y esto computará las probabilidades basadas en la  $n$ -grama. Observe que estas computaciones se realizan con el logaritmo natural. Por ejemplo, escriba lo siguiente:

```
% ngram_print_sentence_score -ngram pegasus.bigram \  
  "<pause1>" what flights "<pause2>"
```

**P2: (a)** ¿Qué ocurre a la medición de la perplejidad cuando ignoramos las pausas al comienzo y fin de las oraciones?

**(b)** Al usar `ngram_print_sentence_score`, observe que la probabilidad logarítmica de comenzar con `<pause1>` no es exactamente cero, y que la probabilidad de finalizar con `<pause2>` tampoco es cero. Explique por qué.

**T4:** Podemos también imprimir las oraciones, reordenadas en función de su probabilidad, con las siguientes opciones:

```
% ngram_perplexity -ngram pegasus.bigram -in dev.sents \  
  -out out.sents
```

Examine el archivo `out.sents` que contiene las oraciones de `dev.sents` pero en orden de probabilidad. Además, mediante `ngramprint_sentence_score`, pruebe varias secuencias de palabras. Sería aconsejable que obtuviese una idea de los tipos de secuencias de palabras que el modelo de lenguaje (tanto bigrama como trigrama) favorece.

**P3: (a)** ¿Qué tipos de oraciones o secuencias de palabra predicen bien los modelos de lenguaje?

**(b)** ¿Qué tipos de oraciones o secuencias de palabra predicen mal?

## Modelos de lenguaje $n$ -grama de clase

En situaciones donde los datos de entrenamiento son escasos para palabras individuales, es posible que queramos asociar grupos de palabras con *clases de equivalencia* para reducir el número de las diversas palabras del vocabulario, simplificando así el número de parámetros que deben estimarse. Las clases se pueden crear manual o automáticamente. Normalmente, las palabras de una clase se relacionan entre sí de algún modo significativo. La equivalencia sintáctica y semántica son elecciones económicas. Un ejemplo de archivo de clase de palabras hecho a mano es `pegasus.rules`.

**T5:** Examine las asociaciones de las clases de palabras especificadas en el archivo `pegasus.rules`.

Para entrenar un modelo de lenguaje  $n$ -grama, necesitamos crear primero un archivo de una gramática libre de contexto (`.cfg`) desde el archivo de esta clase, con el programa `cfg_create`, como se muestra:

```
% cfg_create -rules pegasus.rules \  
  -wlex pegasus.wlex -out pegasus.cfg
```

Podemos entonces usar el programa `ngram_create` para entrenar modelos de lenguaje bigrama y trigrama de clase suavizados, como se muestra a continuación:

```
% ngram_create -cfg pegasus.cfg -in pegasus.sents \  
  -n 2 -out pegasus.class_bigram \  
  -set_log_prob 50 50 0  
% ngram_create -cfg pegasus.cfg -in pegasus.sents \  
  -n 3 -out pegasus.class_trigram \  
  -set_log_prob 50 50 0
```

- P4:** (a) ¿Cuántas clases distintas hay definidas en el archivo `pegasus.rule`?
- (b) ¿De qué forma varían los números de las únicas  $n$ -gramas para las distintas  $n$  en los modelos  $n$ -grama de la palabra creados en la tarea **T1**?
- (c) ¿De qué forma se pueden comparar esos números con aquellos de los modelos  $n$ -grama de clase en la tarea **T5**? Utilice `ngram_tool` para ver los contenidos del archivo del modelo de lenguaje.
- Por ejemplo: `% ngram_tool -in pegasus.class_bigram`

**T6:** Ahora, en vez de usar clases de palabra, echemos un vistazo a las clases de nivel de frase. Estas reglas operan en múltiples niveles. Se le facilitará un conjunto muy simple de reglas de clase frasal en `pegasus_phrase.rules` y se le pedirá que invente alguna más para fomentar más la perplejidad del conjunto de desarrollo.

Primero, creamos la gramática libre de contexto desde el grupo de reglas:

```
% cfg_create -rules pegasus_phrase.rules \
              -wlex pegasus.wlex -out pegasus_phrase.cfg
```

Examinaremos el resultado de estas reglas mediante la función `cfg_reduce_sentences`. Esta función tomará un archivo de oraciones e imprimirá como éstas han sido analizados por las reglas. Ejecute lo siguiente y examine al archivo de salida `out.sents`.

```
% cfg_reduce_sentences -cfg pegasus_phrase.cfg -sents dev.sents \
                       -out out.sents -detail 0
```

Observe que en `out.sents` algunas de las palabras han sido sustituidas por sus nombres de clase. Ejecute de nuevo la función con “2” como argumento para `-detail`. Encontrará los distintos niveles de las reglas impresas al detalle.

Utilice ahora esta gramática para entrenar un bigrama y un trigramas.

- P5:** (a) ¿Cuántas clases distintas hay definidas en el archivo `pegasus-phrase-_rules`?
- (b) ¿De qué modo se comparan los números de las únicas  $n$ -gramas de **T6** con los de **T5** y **T1**?

**T7:** Examine ahora la perplejidad de los modelos de lenguaje de clase de palabra y de clase frasal en el grupo de desarrollo, `dev.sents`.

**P6:** ¿Cómo se comparan las perplejidades de los modelos de clase de palabra con aquellos de clase frasal? ¿Qué modelos son mejores?

**P7:** `cfg_reduce_sentences` le será útil para estos ejercicios.

- (a) ¿Puede crear clases de palabras adicionales para reducir la perplejidad de `dev.sents`? Vuelva a entrenar sus modelos de lenguaje de clase de palabras en `pegasus.sents` y compute la perplejidad de las oraciones en `dev.sents`.
- (b) Haga lo mismo para las reglas de clase frasal. ¿Puede crear clases frasales adicionales para reducir la perplejidad en `dev.sents`? Vuelva a entrenar los modelos y compute la perplejidad.

## Generación aleatoria de oraciones

Otro modo de obtener una idea cualitativa para conocer el buen funcionamiento del modelo de lenguaje consiste en generar oraciones aleatoriamente utilizando la estadística en el modelo de lenguaje entrenado, y en ver si tienen (algún) sentido. El programa `ngram_create_random_sentences` puede utilizarse para esto.

**T8:** Use `ngram_create_random_sentences` para generar 10 oraciones aleatorias desde el modelo bigrama `pegasus.bigram` como se muestra a continuación:

```
% ngram_create_random_sentences -ngram pegasus.bigram -num 10
```

Haga lo mismo con el modelo trigram `pegasus.trigram`. Estudie las oraciones.

**P8:** Basado en su análisis de las oraciones generadas, ¿es mejor el modelo bigrama o el modelo trigram? ¿Por qué?

## Parte II :Interpolación de N-grama con Expectación-maximización

En esta parte, compararemos el método de interpolación simple basado en el cómputo utilizado anteriormente, con un método de interpolación eliminada que se basa en el uso de un algoritmo de expectación-maximización (EM) sobre datos presentados. Este algoritmo comienza con algún valor inicial para  $\lambda$  (normalmente 0.5), y vuelve a estimar repetidamente su valor para mejorar la probabilidad total de algunos datos presentados.

En el sistema SUMMIT, estimamos  $\lambda$  comenzando con la  $n$ -gram de menor orden hasta la  $n$ -grama deseada. Por ejemplo, suponiendo que hayamos estimado las probabilidades del unigrama, utilizamos siguiente fórmula para estimar la probabilidad del bigrama:

$$\tilde{P}(w_2|w_1) = (1 - \lambda_{w_1})P(w_2|w_1) + \lambda_{w_1}\tilde{P}(w_2) \quad (2)$$

Donde  $P(w_2|w_1)$  es la probabilidad del bigrama no interpolado, mientras  $\tilde{P}(w_2|w_1)$  es la del interpolado. Observe que esta notación es algo distinta de la que usted ha visto en clase:  $\lambda$  se utiliza para pesar el unigrama, mientras  $(1 - \lambda)$  se emplea para pesar el bigrama.

Los datos presentados se obtienen mediante un procedimiento *Deja-Uno-Fuera* en el nivel de símbolo  $n$ -grama. Para estimar la probabilidad  $n$ -grama de un símbolo en concreto, descartamos ese símbolo de la computación de probabilidad y lo utilizamos como un símbolo presentado.

**T9:** Utilice el programa `ngram-create` para entrenar un modelo de lenguaje **bigrama** suavizado de EM.

```
% ngram_create -wlex pegasus.wlex -in pegasus.sents \  
-n 2 -out pegasus.em_bigram \  
-em_log_prob 0
```

Entrene un modelo de lenguaje **trigram** suavizado de EM como se muestra a continuación:

```
% ngram_create -wlex pegasus.wlex -in pegasus.sents \  
-n 3 -out pegasus.em_trigram \  
-em_log_prob 0
```

- P9:** (a) ¿Cuál es la perplejidad de `dev.sents` utilizando el bigrama suavizado de EM?  
 (b) ¿Cuál es la perplejidad de `dev.sents` utilizando el trigramma suavizado de EM?  
 (c) ¿Cómo se compara el suavizado EM con la interpolación simple que realizamos en **T1**?

**T10:** Veremos cómo los dos métodos de suavizado se comparan cuándo disminuimos la cantidad de datos de entrenamiento para nuestro modelo de lenguaje bigrama. Para cada método, entrenaremos con el 80%, 60%, 40% y 20% de los datos y compararemos cómo la perplejidad se degrada para los dos métodos, cuando disminuimos la cantidad de datos de entrenamiento.

Existen alrededor de 5.000 oraciones en los datos de entrenamiento `pegasus.sents`. Para obtener el 80% de los datos, utilice:

```
% head -4000 pegasus.sents > pegasus.80.sents
```

Esto creará el archivo `pegasus.80.sents`, con las primeras 4.000 oraciones de los datos de entrenamiento, (o el 80% del entrenamiento completo). Siga el mismo método para crear los grupos de entrenamientos más pequeños.

- P10:** (a) Trace en el mismo grafo la perplejidad como función de la cantidad de datos de entrenamiento para las dos técnicas de suavizado.  
 (b) ¿Qué método funciona mejor cuando utilizamos menos datos?

**T11:** En esta parte, estimará el peso de interpolación EM de  $\lambda$  para el caso del bigrama. Haremos las siguientes suposiciones de simplificación:

Estamos tratando de estimar el peso de interpolación de  $\lambda$  de la palabra *FOR* donde hemos obtenido sólo las tres palabras *DALLAS*, *TODAY* y *THE*. Esto está representado en el árbol de la  $n$ -grama de la Figura 1. Suponga que los cómputos de los datos de entrenamiento son los siguientes:

$$C(FOR) = 14$$

$$C(DALLAS|FOR) = 10$$

$$C(TODAY|FOR) = 2$$

$$C(THE|FOR) = 2$$

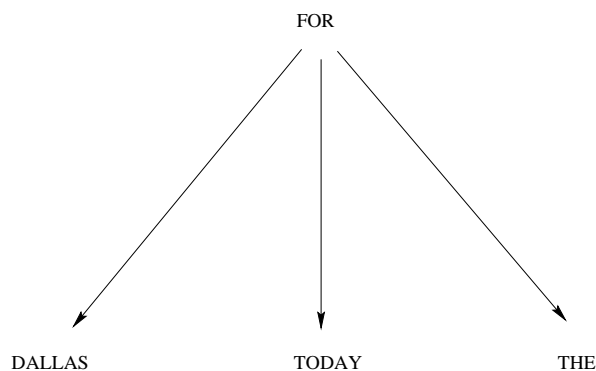


Figura1: Un árbol bigrama para la palabra “FOR”.

Además, suponga que todas las probabilidades del unigrama están ya estimadas y que el descuento no tiene un impacto sobre el valor de estos unigramas. Suponga los siguientes valores para las probabilidades del unigrama:

$$\begin{aligned}\tilde{P}(DALLAS) &= 0.05 \\ \tilde{P}(TODAY) &= 0.15 \\ \tilde{P}(THE) &= 0.70\end{aligned}$$

Bajo estas suposiciones, la ecuación de actualización del EM es:

$$\lambda_{i+1} = \frac{1}{C(W)} \sum_{\forall W_j} \frac{\lambda_i \tilde{P}(W_j)}{(1 - \lambda_i)P(W_j|W) + \lambda_i \tilde{P}(W_j)} C(W_j|W) \quad (3)$$

Donde  $P(W_j|W)$  es la probabilidad del bigrama descontado de un símbolo determinado por:

$$P(W_j|W) = \frac{C(W_j|W) - 1}{C(W) - 1} \quad (4)$$

Observe que el índice de  $\lambda$  aquí indica el número de iteración (a diferencia de las notas de la clase donde se indica el orden de la  $n$ -grama).

**P11:** Utilice la ecuación de actualización para estimar el valor de  $\lambda$  comenzando con un valor inicial de 0.5. Sería aconsejable que volviese a estimar  $\lambda$  hasta que no varíe más de 0.01. ¿ Cuántas iteraciones hacen falta? Cree una tabla de los valores de  $\lambda$  en cada paso.

**P12:** Existen dos casos donde este enfoque de EM conduce a una solución degenerada.

- Facilite un grupo de ejemplo de cálculos ( $W_j|W$ ) que conducirá al caso degenerado de  $\lambda = 1$ . Explique por qué esta respuesta está degenerada.
- Facilite un grupo de ejemplo de cálculos ( $W_j|W$ ) que conducirá al caso degenerado de  $\lambda = 0$ . Explique por qué esta respuesta está degenerada.
- ¿Qué caso, (a) o (b), es peor para el aprendizaje? ¿Por qué?
- Proponga algún límite superior para basado en los cálculos y tamaño del vocabulario.

## Apéndice: Herramientas de SUMMIT

Esta sección facilita sugerencias para la obtención de documentación en línea disponible para herramientas de SUMMIT, proporciona un índice descriptivo de las herramientas utilizadas en la práctica, y menciona algunos de los conocimientos indispensables que son necesarios para completar la práctica.

### Ayuda para el uso de la línea de comandos

Observe que para la mayoría de los programas, proveer el argumento de la línea de comandos `-help` enumerará las funciones del programa junto con los argumentos válidos de la línea de comandos. Esto es útil si se olvida de la sintaxis correcta de uso, o si desea ver las opciones que hay disponibles.

### Índice del programa y descripción

**cfg.create** Crea una gramática libre de contexto (CFG) dado un conjunto de clases/reglas (`.rules`) y un léxico de palabras (`.wlex`). La CFG se utiliza para el entrenamiento de los modelos de lenguaje de la  $n$ -grama.

**cfg.reduce\_sentences** Imprime un grupo de oraciones reducidas por una CFG, dado un conjunto de entrada de oraciones. Esto es útil para examinar las CFG.

**ngram.create** Crea un archivo de un modelo de lenguaje  $n$ -grama dado un grupo de enunciados de entrenamiento y, o un archivo `.cfg` (para una  $n$ -grama de clase), o un archivo `.wlex` (para un  $n$ -grama de palabras). La opción de la línea de comandos `-set-log-prob` se utiliza para establecer las probabilidades logarítmicas y especificar los valores de los parámetros de suavizado.

**ngram.create\_random\_sentences** Genera aleatoriamente oraciones utilizando la estadística del modelo de lenguaje  $n$ -grama especificado. Esto es práctico para la evaluación cualitativa de un modelo  $n$ -grama.

**ngram.perplexity** Computa la perplejidad de un grupo de enunciados especificado, utilizando un modelo de lenguaje  $n$ -grama determinado. Útil para obtener una medida cuantitativa del funcionamiento de un modelo  $n$ -grama.

**ngram.print\_sentence\_score** Imprime la perplejidad y probabilidad logarítmica para una secuencia de palabras determinadas y un modelo de lenguaje.

**ngram.tool** Muestra información sobre un archivo de un modelo de  $n$ -grama determinado. Además, permite la comprobación de los valores del parámetro y del ajuste de los parámetros de suavizado.

**wlex.create** Crea un archivo de un léxico de palabras (`.wlex`) desde un archivo de vocabulario básico (`.vocab`). El archivo del léxico de palabras se utiliza para construir modelos de lenguaje  $n$ -grama.

## Apéndice: Archivos SUMMIT

Esta sección contiene un índice descriptivo de los archivos utilizados en la práctica. Estos archivos contienen modelos de datos y especificaciones del parámetro necesarias para construir un reconocedor de voz.

## Índice del archivo y descripción

**pegasus.sents** Lista de enunciados del grupo de entrenamiento.

**dev.sents** Lista de enunciados del grupo de desarrollo.

**pegasus.rules** Conjunto de palabras a asociaciones de clase o reglas. Esto se utiliza para crear un archivo de CFG que es posteriormente utilizado para construir modelos de lenguaje de  $n$ -grama de clase.

**pegasus\_phrase.rules** Conjunto de frases a asociaciones de clase o reglas.

**pegasus.vocab** Lista de todas las palabras del vocabulario del dominio.