UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Sistema de comprensión de habla multilingüe

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Sergio Laguna Bello

Tutores: Lluis Felip Hurtado, Emilio Sanchis

Curso: 2013/14

Resumen

En este proyecto se realiza la integración de diversas tecnologías para la construcción de un prototipo de un sistema de comprensión del habla multilingüe. Además, se ha desarrollado el módulo de identificación de idioma. Este prototipo es accesible mediante una aplicación web basada en HTML5 y desarrollada con el *framework* web Django.

El prototipo se divide en diferentes módulos: identificación del idioma, reconocimiento automático del habla, traducción y comprensión del habla.

Palabras clave: comprensión del habla, reconocimiento del idioma, Django, aplicación web

Abstract

In this project the integration of diverse technologies for the development of a prototype of multilingual spoken language understanding is performed. Moreover, we have developed the language identification module. This prototype is accessible through a web-based application developed with HTML5 and Django web framework.

This prototype is divided into different modules: language identification, automatic speech recognition, translation and speech understanding.

Keywords: spoken language understanding, language identification, Django, web app

Índice general

Capítulo 1

Introducción

1.1. Comprensión del habla

Bajo el nombre de "Ingenierías del lenguaje" se agrupan un conjunto de técnicas desarrolladas en los últimos años para generar interfaces y herramientas que permitan interaccionar con los computadores mediante el lenguaje, oral o escrito. El análisis de grandes volúmenes de documentos no estructurados, la recuperación y extracción de información, el reconocimiento y comprensión de habla, la identificación de idioma o locutor o los sistemas de diálogo hablado son algunos ejemplos enmarcados en esta área.

Un importante aspecto para la mayoría de las aplicaciones es el poder dotarlas de multilingüismo. Esta posibilidad empezó a hacerse más viable a partir de los años noventa, cuando los sistemas de traducción automática empezaron a dar sus frutos. La ampliación de los sistemas para permitir la interacción multilingüe se suele hacer siguiendo dos esquemas. El primero consiste en mantener el sistema desarrollado en un idioma y traducir la interacción o el *query* del usuario, realizado en otro idioma, al propio del sistema. El otro esquema consiste en adaptar los modelos y el sistema completo a un nuevo idioma, lo cual es deseable hacerlo con el menor esfuerzo posible. Tanto para un caso como para el otro se requiere disponer de un sistema automático de traducción.

Los sistemas de traducción automática que más éxito han tenido se basan en métodos estadísticos. Es decir, a partir de corpus paralelos de frases en dos idiomas se aprende un modelo estadístico basado en la frecuencia de concurrencias de secuencias de palabras en ambos idiomas. Existen hoy en día diversos traductores web de dominios no restringidos, como Google Translator, y también existen herramientas para el aprendizaje específico de traductores en dominios restringidos, siempre que se disponga de un corpus

bilingüe paralelo. Este es el caso de MOSES [1], que es una herramienta de libre acceso que permite desarrollar este tipo de traductores.

Una aplicación que integra muchas de estas tecnologías son los sistemas de diálogo hablado. Este tipo de sistemas empezaron a desarrollarse mucho a partir de la década de los 2000, cuando los resultados en reconocimiento del habla continua fueron lo suficientemente buenos como para abordar tareas donde una moderada tasa de errores del reconocimiento no impedían conseguir los objetivos. Algunos de los principales sistemas se desarrollaron bajo el proyecto americano DARPA Communicator [2], mientras en Europa también se iniciaron proyectos como ARISE [3]. La mayoría de estos sistemas consistían en tareas de acceso a información sobre algún servicio, como trenes o aviones. En la figura 1.1 se presenta un esquema de la arquitectura de un sistema de diálogo hablado.

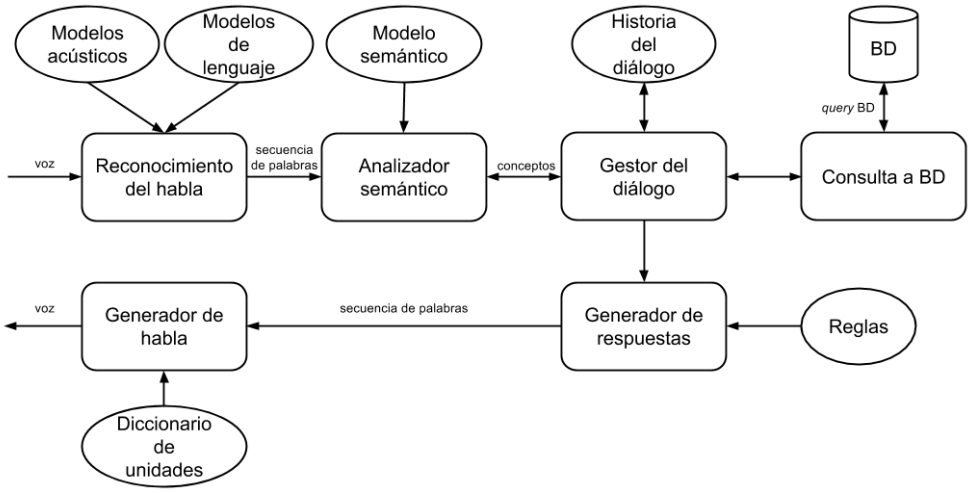

Figura 1.1: Arquitectura de sistema de diálogo hablado

Los distintos módulos que componen el sistema son:

- Reconocedor del habla: Los modelos que se utilizan son los modelos acústico-fonéticos (que representan las características acústicas de los fonemas correspondientes a una lengua) y los modelos de lenguaje (que representan el vocabulario y las concatenaciones y estructuras sintácticas de la lengua).
- Analizador semántico o módulo de comprensión: Es una de las partes principales del sistema de diálogo. En este módulo se realiza la interpre-

tación de las frases de entrada generando a la salida una representación semántica de la frase.

- Gestor de diálogo: Este módulo es el encargado de tomar las acciones necesarias para desarrollar el diálogo con el usuario. A partir de la historia del diálogo, la última intervención del usuario y, algunas veces, el resultado de la consulta al sistema de información, este módulo es el encargado de generar el turno de diálogo del sistema. Normalmente genera una representación codificada de la información que hay que transmitir al usuario y se lo envía al módulo generador de respuestas.

- Generador de respuestas: Este módulo se encarga simplemente de construir una frase en el idioma correspondiente a partir de la información recibida del gestor de diálogo. Normalmente la generación de respuestas se realiza a partir de un conjunto de plantillas que se completan con algunos datos concretos.

- Generador de habla: Es un clásico sistema de síntesis de voz.

Los sistemas de diálogo desarrollados actualmente se caracterizan por ser de dominios semánticos restringidos, es decir, están diseñados para tareas en las que la talla del vocabulario no es muy grande (menos de cinco mil palabras), y el conjunto de conceptos con los que construir la representación semántica también está muy acotado. En cuanto al tipo de interacción con el usuario, para que sean sistemas amigables, es necesario que acepten "iniciativa mixta", es decir, que el usuario pueda realizar turnos de diálogo con una cierta libertad. De esta forma los sistemas se alejan de los prototipos iniciales en el que el modelo de interacción era un estricto esquema de pregunta-respuesta, totalmente dirigido por el sistema.

El trabajo realizado en este proyecto se ha centrado en el módulo de comprensión [4] en el marco del proyecto TIMPANO [5]. Se ha trabajado sobre la interpretación semántica de las frases a partir de entradas que contienen múltiples hipótesis. Además, se le ha dotado de mecanismos para poder trabajar con otros idiomas diferentes del propio del sistema [6]. Para ello, ha sido necesario desarrollar un modelo de traducción del idioma del usuario (se ha diseñado para el francés e inglés) al del sistema (que es el castellano). También se ha estudiado el problema de la detección automática del idioma, es decir, a partir del audio del usuario poder detectar en qué idioma ha hablado y automáticamente configurar el traductor para ese idioma. Todo ello ha llevado a la implementación de un prototipo de comprensión multilingüe.

1.2. Objetivos

El objetivo principal de este proyecto es la implementación de un prototipo de comprensión multilingüe enfocado a una tarea de consulta de horarios de trenes a partir del proyecto DIHANA [7]. Este objetivo lo podemos dividir dos objetivos más concretos:

1. Plantear una arquitectura de comprensión del habla que acepte diferentes idiomas de entrada. En concreto, es necesario introducir un módulo de identificación de idioma al sistema.

2. Desarrollar una aplicación web que contenga el sistema anterior. Esta aplicación debe ser capaz de recoger la consulta del usuario.

La memoria se ha dividido de la siguiente forma:

- En el capítulo 2 se describe la arquitectura del sistema de comprensión del habla. Partimos de una arquitectura clásica hasta llegar a una arquitectura capaz de comprender múltiples idiomas.

- Dada su importancia, en el capítulo 3 vemos como podemos determinar el idioma del usuario a partir de su consulta en audio.

- En el capítulo 4 se listan las diferentes tecnologías utilizadas en el desarrollo de nuestro prototipo.

- En el capítulo 5 se describe la arquitectura de la aplicación web desarrollada.

- En el capítulo 6 se detalla el proceso del sistema desde que se recoge el audio hasta que se muestran los resultados.

- En el capítulo 7 se detalla toda la experimentación realizada.

- Por último, en el capítulo 8 se comprueba si se han cumplido los objetivos del proyecto y se plantean diferentes mejoras o ampliaciones al trabajo realizado.

Capítulo 2

Sistema de comprensión del habla

En un sistema de comprensión del habla o *Spoken Language Understanding System* (SLU) nuestro objetivo es obtener una interpretación semántica de un fragmento de audio.

En este capítulo vamos a ver como podemos definir este sistema. Para ello, primero describiremos la arquitectura clásica. Después, veremos una arquitectura ampliada que permite reconocer un nuevo idioma sin que ello conlleve un gran trabajo [8]. Finalmente, modificaremos esta arquitectura para que admita varios idiomas de entrada, que es nuestro objetivo.

2.1. Arquitectura clásica de comprensión

Para comenzar, vamos a ver en qué debe consistir un sistema de comprensión del habla. Primero debemos tener claro que la entrada del sistema será un fragmento de audio y la salida del sistema debe ser la información semántica contenida en ese audio.

Para que el sistema sea más sencillo, lo separaremos en dos fases. En la primera fase reconoceremos el audio para poder representarlo en texto. En la segunda, partiremos del texto anterior para extraer la información relevante a la tarea.

En la primera etapa queremos pasar el audio a texto. Para obtener la transcripción de un audio necesitamos un reconocedor automático del habla o *Automatic Speech Recognizer* (ASR). El ASR realiza esta transformación a partir de información acústica, léxica y sintáctica, representada en modelos. En nuestro caso, utilizamos un ASR disponible en la web.

En la segunda etapa pasaremos de la transcripción a su contenido semánti-

co mediante el módulo de comprensión de texto (en adelante, comprendedor) [9]. Para ello será necesario aprender un modelo semántico, el cual utilizará este módulo para obtener la información semántica del texto. El aprendizaje del modelo semántico partirá de un corpus debidamente segmentado y etiquetado con un conjunto de conceptos dependientes de la tarea. Es decir, las frases que formen el corpus deben dividirse en fragmentos de forma que cada uno de ellos tenga asociado un concepto antes definido en función de la tarea.

El módulo de comprensión nos devuelve la segmentación del texto con los conceptos asociados a cada segmento. Esta segmentación será procesada para convertirla a una representación basada en marcos (*frames*), de forma que se extraiga la información relevante.

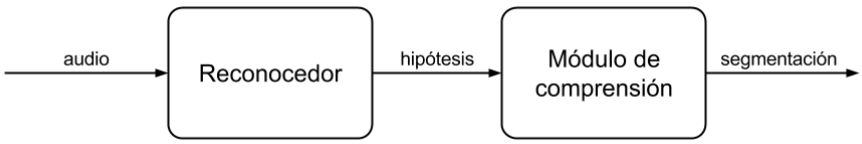

Figura 2.1: Esquema de la arquitectura clásica

Esta arquitectura desacoplada presenta determinados problemas:

- La transcripción que obtiene el reconocedor y que después utiliza el comprendedor puede contener errores, de forma que es posible que se pierda información durante la primera etapa del sistema. Esta pérdida de información puede condicionar la etapa de comprensión.
- También es una fuente de errores la fase de comprensión, por lo que se pueden acumular errores de las dos fases que forman el sistema actual.

Además, esta arquitectura está pensada para comprender un solo idioma, por lo que no nos sirve para nuestro objetivo. El siguiente paso debe consistir en mitigar los problemas anteriores y acercarnos a una arquitectura de comprensión del habla multilingüe. Por eso, en el siguiente apartado vamos a ver una arquitectura que facilita el desarrollo de un sistema de comprensión para otro idioma partiendo de un modulo de comprensión ya existente.

2.2. Comprensión de un nuevo idioma

Después de ver la anterior arquitectura, vamos a plantear una nueva que solucione los problemas encontrados en la anterior y nos acerque a nuestro

objetivo de un sistema multilingüe. Para ello, primero vamos a ver como comprender un nuevo idioma a partir de un módulo de comprensión que hemos obtenido anteriormente para otro idioma.

Si queremos comprender un segundo idioma, en un principio necesitaríamos un nuevo corpus en este idioma segmentado y etiquetado que utilizaríamos para aprender el modelo semántico que utiliza el comprendedor. Sin embargo, este proceso es muy costoso, con lo que nos planteamos si es posible reutilizar un módulo de comprensión de otro idioma.

Para poder utilizar un comprendedor que hemos obtenido para un idioma diferente necesitaremos añadir una nueva fase entre el *ASR* y el módulo de comprensión, de forma que a partir del texto que representa al audio en el idioma correspondiente lo traduzca al idioma que acepta nuestro comprendedor [10]. Por ejemplo, si ya tenemos un comprendedor del español y queremos utilizarlo para un sistema de comprensión del francés, la nueva fase se encargaría de traducir la transcripción en francés al español.

Por lo tanto, si partimos de un sistema de comprensión para un idioma y lo queremos adaptar a un nuevo idioma, realizaremos los siguientes cambios:

- En el caso de la etapa de reconocimiento, debemos cambiar el *ASR* a uno que reconozca el idioma del usuario. En nuestro caso, al utilizar un reconocedor online es tan fácil como cambiar el parámetro de idioma.

- Después de la etapa de reconocimiento, añadimos una nueva etapa de traducción donde pasaremos el texto devuelto por el *ASR* para traducirlo al idioma que acepte el comprendedor. En este caso, para traducir el resultado del reconocedor utilizamos MOSES. MOSES es un sistema de traducción automática estadística que permite entrenar modelos de traducción y utilizarlos posteriormente para traducir entre pares de idiomas.

  Para poder realizar la traducción con MOSES necesitamos un corpus paralelo con el que aprender el modelo de traducción asociado. Un corpus paralelo es un conjunto de frases en dos idiomas en el que existe una correspondencia entre frases, de forma que la frase i-ésima de un idioma es la traducción de la frase i-ésima del otro. A partir de este corpus, podemos utilizar MOSES para aprender el modelo de traducción que nos permite traducir de un idioma a otro.

Al añadir la etapa de traducción podemos comprender un idioma utilizando un comprendedor de otro idioma, de forma que no necesitamos un corpus segmentado y etiquetado por cada idioma que queramos comprender. Sin embargo, seguimos necesitando un corpus, el corpus paralelo que necesitamos

para entrenar el modelo de traducción. En este caso, el corpus se traduce al idioma nuevo mediante un conjunto de traductores web. Esto supone que los errores que cometen los traductores web afectarán al modelo de traducción resultante, aunque también obtenemos esta traducción más rápida que de forma manual. Por este motivo, podemos utilizar varios traductores web para reducir el número de errores.

Con todo esto, el sistema es capaz de comprender un idioma diferente al que acepta directamente el módulo de comprensión. Sin embargo, seguimos arrastrando los problemas comentados en el anterior apartado. No solo eso, hemos aumentado la problemática al introducir la etapa de traducción, ya que es una nueva fuente de errores donde podemos perder información. Para intentar reducir estos errores vamos a introducir los siguientes cambios:

- Para evitar la pérdida de información en la fase de traducción vamos a cambiar el resultado de esta etapa para que devuelva un número n de posibles traducciones. El traductor no genera una sola hipótesis, sino que su resultado es un conjunto de hipótesis con una puntuación asociada, de forma que la traducción que estábamos obteniendo es la de mayor puntuación o confianza. Al utilizar las n traducciones con mayor confianza, tenemos más información que puede facilitar la obtención de la interpretación semántica correcta.

  Sería deseable aplicar este cambio también en la fase de reconocedor, de forma que el traductor generase las traducciones a partir de las diferentes hipótesis del *ASR*. Sin embargo, esto no resulta fácil, ya que el traductor debería conocer que no son frases diferentes, sino que son hipótesis del reconocedor para una misma frase.

- Aplicando el cambio anterior tendríamos que la salida del traductor es un conjunto de posibles traducciones. Por lo tanto, tendremos que cambiar la fase de comprensión, que hasta ahora esperaba una única traducción, para que tenga en cuenta todas las posibles traducciones.

Para implementar el punto anterior, que el comprendedor recoja un conjunto de traducciones, necesitaremos introducir una nueva etapa en el sistema. En esta etapa, situada entre la traducción y el modulo de comprensión, partiremos de las n traducciones de la etapa anterior para representarlas en un grafo dirigido [11]. Este grafo consta de un conjunto de nodos y arcos, donde cada arco representa una palabra y cada posible camino entre el nodo inicial y el nodo final es una de las traducciones. Para generar este grafo se sigue un proceso dividido en dos fases: el alineamiento de las hipótesis mediante un algoritmo *Multiple Sequence Alignment* (*MSA*) [12] para conseguir

la matriz de alineamiento y la utilización de esta matriz para generar el grafo de palabras.

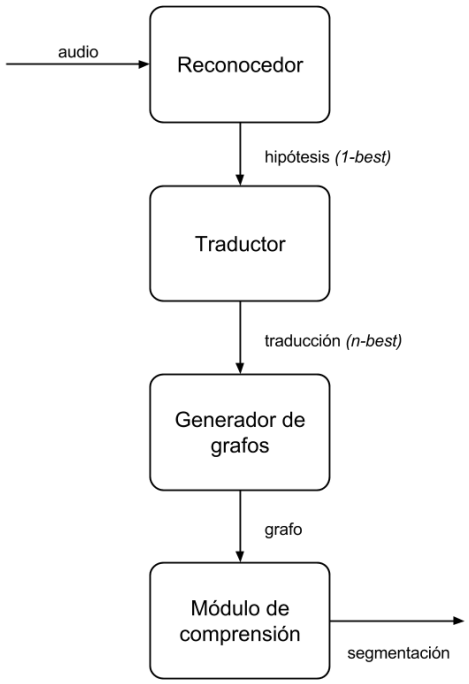

Figura 2.2: Esquema de la arquitectura ampliada

Hasta aquí ya tenemos una arquitectura de comprensión del habla que se puede adaptar fácilmente a diferentes idiomas y sin excesivos problemas de pérdida de información. Aún así, el sistema planteado no acepta varios idiomas, sino que solo acepta un único idioma que debe ser conocido. Por ello, necesitamos cambiar la arquitectura del sistema para que pueda detectar el idioma utilizado y adaptar las diferentes etapas a ese idioma. En el siguiente apartado veremos los cambios necesarios para que el sistema de comprensión admita varios idiomas de entrada.

2.3. Comprensión multilingüe

En este apartado vamos a describir las modificaciones necesarias para que el sistema de comprensión admita varios idiomas de entrada. Para ello debemos ser capaces de conocer el idioma que está utilizando el locutor.

Si seguimos la arquitectura vista en el apartado anterior (2.2), estamos dando por conocido el idioma que utiliza el usuario. Sin embargo, en nuestro sistema aceptamos más de un idioma y no conocemos cual de ellos utilizará el usuario. Sí podemos suponer que el usuario utilizará uno del conjunto de idiomas aceptados (en este caso, español, inglés y francés). Con esta información ya sabemos que debemos tener un *ASR* para cada uno de los idiomas (como utilizamos un reconocedor web, no hay problema) y un modelo de traducción para cada idioma (al ser el idioma del comprendedor el español solo necesitamos dos modelos de traducción: inglés-español y francés-español).

Teniendo los reconocedores para cada idioma y los modelos de traducción necesarios, solo falta saber a cual de los posibles idiomas pertenece el audio. Por eso, el cambio más importante en la arquitectura será una nueva etapa anterior al reconocedor del habla. En esta etapa, el objetivo es identificar el idioma del audio de entrada del sistema para poder utilizar el *ASR* correspondiente y si es necesario traducir las hipótesis del reconocedor y con que modelo de traducción. La detección del idioma se realizará mediante un proceso de decodificación acústico-fonética. Este proceso se verá en mayor detalle en el capítulo 3.

En estos momentos, la arquitectura se corresponde con un sistema de comprensión del habla multilingüe. El sistema es capaz de recoger un audio en varios idiomas conocer *a priori* cual se va a utilizar y comprenderlo. El sistema queda de la siguiente forma:

1. Detectamos el idioma al que pertenece el audio.
2. A partir del idioma detectado, pasamos el audio al reconocedor correspondiente.
3. Si es necesario, traducimos las hipótesis del reconocedor al idioma que acepta el comprendedor.
4. Construimos un grafo de palabras que represente las hipótesis de la fase anterior.
5. El comprendedor utiliza el grafo de palabras para obtener la mejor secuencia de conceptos que represente el conjunto de hipótesis.

El sistema planteado no está exento de problemas. Dado que en la etapa de reconocimiento utilizamos un *ASR* para cada idioma, hay una gran dependencia entre esta etapa y la detección del idioma. Por lo tanto, si la detección del idioma falla, las siguientes etapas se verán afectadas.

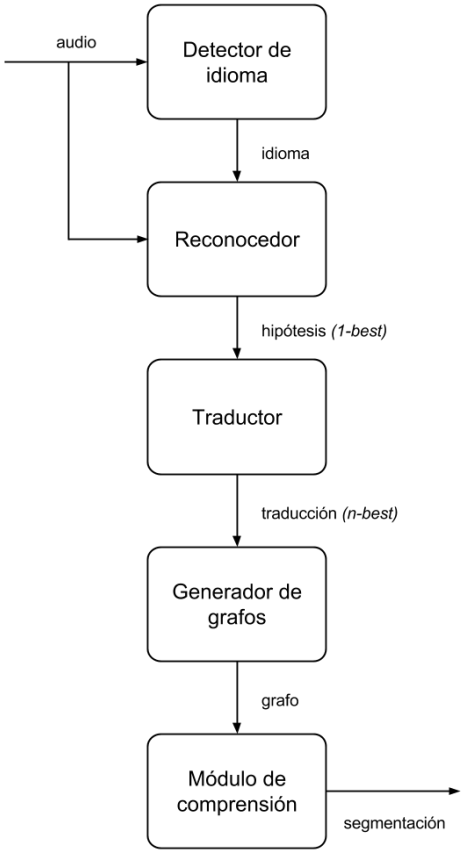

Figura 2.3: Esquema de la arquitectura multilingüe

Capítulo 3

Identificación del idioma

En este capítulo vamos a ver como realizar la identificación del idioma. Este proceso debe tomar como entrada el audio del usuario y dar como salida el idioma correspondiente a ese audio entre los diferentes idiomas que acepta el sistema (español, inglés y francés). Para afrontar este problema vamos a ver como podemos convertir la señal acústica a una representación en texto y utilizar esta representación como punto de partida de la clasificación del audio en un idioma.

En nuestro caso, hemos optado por realizar una decodificación acústico-fonética al audio de entrada para obtener la representación en fonemas de este audio y partir de estos fonemas para decidir el idioma al que pertenece, planteando esta segunda parte como un problema de clasificación. A continuación veremos con más detalle ambas partes.

3.1. Decodificación acústico-fonética

En la primera parte de la etapa de identificación del idioma queremos saber los fonemas que forman el audio. Este proceso se conoce como decodificación acústico-fonética, es decir, pasar la señal acústica a una representación en fonemas. Para realizar esta decodificación utilizamos *Hidden Markov Model Toolkit* (*HTK*), un *toolkit* que nos permite construir y modificar modelos ocultos de Markov y que es principalmente utilizado en reconocimiento del habla. En este *toolkit* está incluida la herramienta `HDecode`, un reconocedor del habla, que utilizaremos para obtener la transcripción fonética de un fichero de audio.

Si queremos utilizar `HDecode` para realizar la decodificación acústico-fonética, primero necesitamos los siguientes ficheros:

- Un modelo acústico que contenga la información de los fonemas.

- Un modelo de lenguaje de fonemas que indique las probabilidades de concatenar los distintos fonemas.
- Un diccionario con los distintos fonemas y que recoja el modelo de silencio.
- Un fichero de características que utilice los coeficientes cepstrales en las frecuencias de Mel (MFCC) que represente el audio.

En nuestro caso, el modelo acústico, el modelo de lenguaje y el diccionario los vamos a obtener a partir del corpus TC-STAR [13]. Utilizaremos el subconjunto de audios en español, de forma que los resultados del proceso de decodificación acústico-fonética se dan utilizando fonemas del español.

En los siguientes apartados vamos a ver como generar los modelos y el diccionario necesarios para la decodificación acústico-fonética y como pasar del audio inicial a los fonemas que lo forman.

3.1.1. Generar un modelo acústico

Para generar un modelo acústico necesitamos un conjunto de ficheros de audio y las transcripciones de esos audios. El proceso de creación de los modelos acústicos se dividen en tres etapas. Este proceso se ha realizado siguiendo el tutorial del *HTK Book* [14].

1. Creación de lista de índices para entrenamiento y test.

   En esta etapa partimos el corpus en dos subconjuntos: entrenamiento y test. La partición quedará en un 80 % para entrenamiento y el 20 % restante para test. Realizamos esta partición para después poder comparar los distintos modelos acústicos; en producción utilizaremos un modelo acústico entrenado con todo el corpus.

2. Extracción de características

   En esta etapa, el objetivo es extraer las características de los audios utilizados para entrenamiento. La representación de estas características será mediante coeficientes cepstrales en la escala de Mel (*MFCC*). Para el calculo de estos coeficientes se sigue el siguiente proceso: transformada de Fourier, banco de filtros de Mel, logaritmo y transformada discreta del coseno.

3. Entrenar modelos acústicos

   Después de preparar los datos, podemos comenzar el aprendizaje del modelo acústico. Esta etapa se divide en cinco subetapas:

a) Primero necesitamos generar un fichero con formato MLF que contenga las frases en texto del conjunto de entrenamiento. También necesitamos un diccionario que contenga las transcripciones fonéticas de las diferentes palabras de las frases de entrenamiento.

b) Comenzamos el entrenamiento del modelo acústico partiendo de un fichero con formato MLF con la transcripción fonética de cada frase de entrenamiento que hemos obtenido con los ficheros anteriores. El entrenamiento parte con unos valores triviales (*flat start*) y realizamos sucesivas reestimaciones de los parámetros con el comando `HERest`. De esta forma ya tendremos un conjunto de HMM, uno para cada fonema, pero sin incluir los silencios. Modificamos el modelo obtenido en la última iteración para añadirlos.

c) Después de modificar el modelo, necesitamos realinear los datos. Modificamos el diccionario de pronunciaciones para añadir posibles silencios al final de cada pronunciación. Utilizamos el nuevo diccionario y el modelo acústico antes modificado para reconocer los ficheros de audio y obtener la representación en fonemas en un fichero con formato MLF. Volvemos a reestimar en varias iteraciones el modelo con el fichero MLF que acabamos de obtener.

d) El modelo que tenemos hasta este momento es de fonemas. Sin embargo, queremos un modelo acústico de trifonemas, por lo que utilizaremos el comando `HLEd` para pasar a un modelo de trifonemas y volveremos a utilizar `HERest` para reestimar los parámetros de los HMM.

e) Por último, utilizando el modelo acústico que hemos obtenido en la etapa anterior, obtendremos una serie de modelos con diferente número de mixturas de Gaussianas. Para lograrlo utilizamos el comando `HHEd` y posteriormente volvemos a reestimar los parámetros del modelo.

Una vez acabado este proceso tendremos un conjunto de modelos acústicos de trifonemas con diferente cantidad de mixturas de Gaussianas.

3.1.2. Generar un modelo de lenguaje

El modelo de lenguaje que necesita el comando `HDecode` es un modelo de trifonemas que represente las probabilidades de cada trifonema. Para generar este modelo debemos partir del Master Label File a nivel de fonemas generado durante la obtención del modelo acústico.

Primero debemos pasar las transcripciones fonéticas del MLF a un fichero donde cada transcripción ocupe una línea. Con el programa `ngram-count` se generará un modelo a partir del fichero anterior que indica las probabilidades (en forma logarítmica) de cada combinación de monofonemas, bifonemas y trifonemas.

3.1.3. Generar el diccionario

Uno de los ficheros necesarios del reconocedor es el diccionario. En este diccionario deben aparecer todos los fonemas del modelo acústico. Este fichero también sirve para indicar que determinados símbolos como `<s>` o `</s>` (utilizados para indicar el comienzo y el fin de una frase) se corresponden con el modelo de silencio (es decir, el símbolo `sil`).

```
1 -pau- sil
2 </s> sil
3 <s> sil
4 <unk> sil
5 B B
6 D D
7 ...
```

Figura 3.1: Contenido del diccionario

3.1.4. De audio a fonemas

Una vez que tengamos el modelo acústico, el modelo de lenguaje y el diccionario ya podemos realizar la decodificación acústico-fonética. Sin embargo, la decodificación no se puede realizar directamente con el audio, sino con un fichero `.mfc` que contiene la representación del audio mediante vectores de características. Este fichero se obtendrá con un extractor de características.

Una vez obtenido el fichero anterior mediante un extractor de características, podemos utilizar `HDecode` con el modelo acústico, el modelo de lenguaje y el diccionario para conseguir decodificar el audio a la secuencia de fonemas más probable.

Ya tenemos la secuencia de fonemas que representa el audio, por lo que podemos pasar a la siguiente parte del proceso de identificación del idioma.

3.2. Detección de idioma

En esta segunda parte el objetivo es ver a que idioma pertenece el audio a partir del resultado de la decodificación acústico-fonética. Este es un problema de clasificación en el que tenemos una clase por idioma (español, inglés y francés). Para abordar este problema nos planteamos diferentes aproximaciones: mínima perplejidad asociada a modelos de lenguaje y clasificador multiclase basado en máquinas de vectores soporte (SVM). A continuación vemos a detallar cada una de ellas.

3.2.1. Mínima perplejidad

En esta aproximación para asignar un idioma a una frase partiendo de los fonemas que la representa, necesitamos un conjunto de modelos de lenguaje, uno por cada idioma que queramos identificar. Estos modelos se generan de forma similar al modelo de lenguaje que hemos generado en la etapa anterior para el reconocedor, pero en este caso utilizaremos las transcripciones fonéticas en el idioma al que corresponda, es decir, para aprender el modelo del español utilizaremos las transcripciones de los audios en español.

En nuestro caso aprenderemos modelos de lenguaje para español, inglés y francés. Con estos modelos y los fonemas obtenidos en el paso anterior vamos a comprobar a cual de los posibles idiomas pertenece el audio.

Mediante el programa `ngram` podemos medir cuanto se ajusta una secuencia de fonemas a un modelo de lenguaje. `ngram` nos devuelve a partir del modelo de lenguaje y de los fonemas la perplejidad asociada al conjunto de prueba, en nuestro caso una frase. La perplejidad se define como 2 elevado a la entropía:

$$2^{H(x)} \tag{3.1}$$

Dado que la entropía se calcula mediante la siguiente fórmula:

$$H(x) = -p(x) \cdot log_2 \cdot p(x) \tag{3.2}$$

podemos redefinir la ecuación 3.1 como:

$$2^{-p(x) \cdot log_2 \cdot p(x)} \tag{3.3}$$

La entropía (y, por lo tanto, la perplejidad) mide la incertidumbre. De esta forma, cuanto mayor sea el valor mayor es la incertidumbre. Por lo tanto, utilizando este valor para la detección del idioma, podemos suponer que una cadena de fonemas pertenece al idioma del modelo con menor entropía o

perplejidad. De esta forma, la salida de fase de detección de idioma es la siguiente:

$$\hat{L} = \arg\min_{L \in \mathcal{L}} 2^{-p(x|L) \cdot log_2 \cdot p(x|L)} \tag{3.4}$$

En nuestro caso, al utilizar modelos de lenguaje de trifonemas, la probabilidad $p(x)$ de define de la siguiente forma:

$$p(x) = p(x_1 x_2 \dots x_n) \tag{3.5}$$

$$p(x) = \prod_{i \in |x|} p(x_i | x_{i-1}, x_{i-2}) \tag{3.6}$$

3.2.2. Clasificador multiclase SVM

Otra alternativa para la detección del idioma es un clasificador multiclase basado en máquinas de vectores soporte, *Support Vector Machines* (*SVM*). En este caso, utilizamos los datos de entrenamiento para construir el clasificador. Dado que este debe ser un clasificador de tres clases, necesitaremos varias máquinas de vectores soporte para implementar el clasificador. Podemos elegir entre un clasificador "uno contra uno" (*one-versus-one*) y "uno contra el resto" (*one-versus-the-rest*). La diferencia entre ambos tipos suele ser el número de *SVM* necesarios, ya que mientras un clasificador "uno contra uno" necesita $n \cdot (n-1)/2$ para n clases, un clasificador "uno contra el resto" solo necesita n *SVM*. En nuestro caso, como tenemos tres idiomas, para ambos tipos de clasificador se necesitan tres *SVM*.

El principal problema que tenemos si queremos utilizar esta aproximación es que la entrada del clasificador no puede ser una cadena de fonemas. Por este motivo, necesitaríamos un paso intermedio como la utilización de un vectorizador que convierta la representación en fonemas a un vector. Una posibilidad es utilizar la métrica *tf.idf* (*Term Frecuency - Inverse Document Frecuency*) que expresa la relevancia de una palabra según su frecuencia en un conjunto de documentos.

$$tf.idf(t, d, D) = tf(t, d) \cdot idf(t, D) \tag{3.7}$$

Dado que en nuestro caso una palabra se corresponde con un fonema, utilizar esta métrica directamente no sería conveniente, ya que la frecuencia de los fonemas en las distintas frases será relativamente alta. Por este motivo, en lugar de fonemas, podemos utilizar n-gramas de fonemas como términos.

Capítulo 4

Tecnologías

El sistema de reconocimiento del habla multilingüe que se ha visto en el capítulo anterior va a ser el que se implemente en este trabajo. Para ello, se va a desarrollar una aplicación web que permita al usuario realizar una consulta en audio que será la entrada de nuestro sistema. Esto hace necesario plantearse que tecnologías se utilizarán en la implementación de esta aplicación, siendo algunas de ellas obligadas por ser una aplicación web.

4.1. HTML

El lenguaje de marcado de hipertexto o *HyperText Markup Language* (*HTML*) es un lenguaje estándar para elaborar páginas web. Este lenguaje es uno de los estándares que desarrolla el *World Wide Web Consortium* (*W3C*), organización centrada en los estándares web. *HTML* es el encargado de dar una estructura determinada a una página web. Permite añadir textos, imágenes, formularios o enlaces al documento web entre otros elementos.

Aunque la última versión de *HTML* es HTML4, publicada en 1999, la definición de la nueva versión, HTML5, está finalizando (está previsto que se publique la versión final durante este año) y su utilización en la web cada vez es mayor. Esta versión introduce numerosas características nuevas, como, por ejemplo, la posibilidad de reproducir vídeos y audios sin utilizar plugins externos.

Vamos a aprovechar algunas características que nos ofrece HTML5 en el desarrollo de nuestra aplicación web. En concreto, utilizaremos el elemento `<audio>` y la *API* Web Audio [15] para grabar la consulta del usuario.

4.2. CSS

Hojas de estilos en cascada o *Cascading Style Sheets* (*CSS*) es un lenguaje que permite modificar el estilo de la web. De igual forma que *HTML*, *CSS* es un lenguaje estándar a cargo del *W3C*. Si el lenguaje *HTML* se encarga de la estructura y los datos, con *CSS* podemos definir el aspecto de la web. Con CSS podemos modificar desde el tamaño de la fuente de un texto hasta cambiar completamente el aspecto de un formulario.

Utilizando CSS conseguimos que la interfaz de la aplicación sea más amigable para el usuario, pero esto conlleva mucho tiempo para definir los estilos. Por ello, para que la modificación de la interfaz sea más rápida y sencilla, vamos a utilizar *Bootstrap*, un *framework* para el diseño de sitios web

Bootstrap fue diseñado en *Twitter* y actualmente es un proyecto de código abierto. Esta herramienta proporciona un conjunto de hojas de estilo que se pueden utilizar directamente, aunque también es posible modificarlas. En nuestro caso, utilizaremos los estilos predefinidos, que nos permitirán conseguir una mejor interfaz sin emplear demasiado tiempo en su implementación.

4.3. JavaScript

JavaScript es un lenguaje interpretado enfocado a la web. Es utilizado habitualmente en el lado del cliente (el navegador web del usuario) por webs dinámicas para, por ejemplo, realizar la validación de formularios antes de enviar la solicitud al servidor.

En el caso de nuestra aplicación web, utilizamos JavaScript para grabar el audio. Aunque, como ya se ha comentado en el apartado 4.1, utilizamos *HTML*5 para acceder al micrófono, realmente el encargado de grabar el audio es un plugin JavaScript llamado Recorder.js [16].

No solo utilizamos JavaScript para la grabación del audio, sino que *Bootstrap* emplea JavaScript para determinados elementos, requiriendo de la librería de JavaScript jQuery, una librería de JavaScript que facilita el desarrollo de aplicaciones web dinámicas.

La versión de *Bootstrap* que estamos utilizando requiere una versión de jQuery igual o superior a la versión 1.9.0. Nosotros vamos a utilizar la versión 2.1.1, ya que aunque la rama 2.x pierde la compatibilidad con las versiones de Internet Explorer anteriores a la 9, nuestra aplicación no puede grabar audio en Internet Explorer debido a la falta de soporte de la *API* Web Audio en este navegador.

4.4. Django

Dado que la aplicación web va a ser implementada utilizando el lenguaje Python, necesitaremos un *framework* que nos permita construir aplicaciones web en este lenguaje de programación.

Django [17] es un *framework* de código abierto basado en Python para el desarrollo de aplicaciones web. El desarrollo de este *framework* comenzó en 2003 cuando los desarrolladores web del periódico *Lawrence Journal-World* (Kansas, EEUU) comenzaron a utilizar Python para construir aplicaciones. Fue en 2005 cuando se liberó como código abierto [18].

Django es un *framework* que sigue, en cierto modo, la arquitectura *Model-View-Controller* (*MVC*). Esta arquitectura intenta separar los datos de la aplicación y la interfaz que determina como se presentan los datos, de forma que la modificación tanto de la lógica de la aplicación como de la interfaz de usuario sea más sencillo. A continuación vamos a ver en detalle cada uno de los componentes de esta arquitectura:

- Modelo
  El modelo es la representación de la información que almacena la aplicación. Se encarga también de las peticiones de acceso y actualización de la información que puede realizar el controlador.

- Vista
  La vista se encarga de la interfaz de usuario, donde mostrará el contenido del modelo.

- Controlador
  El controlador se encarga de enlazar el modelo y la vista. Tras una acción del usuario, se encarga de obtener o modificar la información contenida en el modelo.

Sin embargo, Django no implementa esta arquitectura plenamente, sino que difiere en algunos aspectos. El primero de ellos es que la vista de Django define que datos se mostrarán al usuario, a diferencia de la arquitectura *MVC*, donde la vista define como se mostrarán esos datos. El componente que en Django describe la interfaz de usuario es el *template* o plantilla. Habitualmente, una plantilla contiene código HTML y variables, que formarán una página web mostrando los datos que esas variables contengan.

La segunda diferencia es la falta de un controlador como tal en Django, aunque podemos decir que las funciones del controlador las asume el propio *framework*.

4.5. Gunicorn

Durante el desarrollo de nuestra aplicación web podemos utilizar el servidor incluido en Django, `runserver`. Este servidor está pensado para entornos de desarrollo, por lo tanto, cuando lleguemos a la fase de producción debemos cambiar el servidor que se encargará de nuestra aplicación.

Para sustituir el servidor de desarrollo de Django vamos a utilizar *Gunicorn* [19], un servidor HTTP que implementa *Python Web Server Gateway Interface*, definido en la *Python Enhancement Proposal* 3333 [20].

Aunque *Gunicorn* es capaz de servir nuestra aplicación, no puede sustituir al servidor de desarrollo de Django completamente, ya que *Gunicorn* solo se encarga del contenido dinámico dejando a un lado los ficheros estáticos. Por ello es necesario añadir un servidor que manejará el contenido estático de la aplicación web.

4.6. Nginx

Como hemos comentado en el apartado anterior, el servidor web que vamos a utilizar es *Gunicorn*. Sin embargo, desde la web de *Gunicorn* se recomienda que el servidor se encuentre detrás de un proxy. Uno de los motivos para esta recomendación es que sin un proxy que se encargue de los "clientes lentos", *Gunicorn* es vulnerable a ataques de denegación de servicio (*DoS*) [21].

Uno de los posibles servidores proxy a utilizar y, además, recomendado por *Gunicorn* es Nginx [22]. Es un servidor proxy con baja utilización de recursos. También nos servirá para solucionar el problema con los ficheros estáticos, ya que es capaz de servirlos.

Capítulo 5

Arquitectura del prototipo

Dado que nos encontramos desarrollando una aplicación web, nos situamos en un esquema cliente-servidor, donde el cliente será el navegador web que utiliza el usuario para acceder a la aplicación y el servidor el que se encarga de procesar la petición y, posteriormente, devolver al cliente una respuesta.

Como hemos indicado en los apartados 4.5 y 4.6, en el lado del servidor tendremos dos servidores web: Nginx, que hace de servidor proxy y sirve el contenido estático y *Gunicorn*, que se encarga de la aplicación web. De esta forma la arquitectura queda de la siguiente forma:

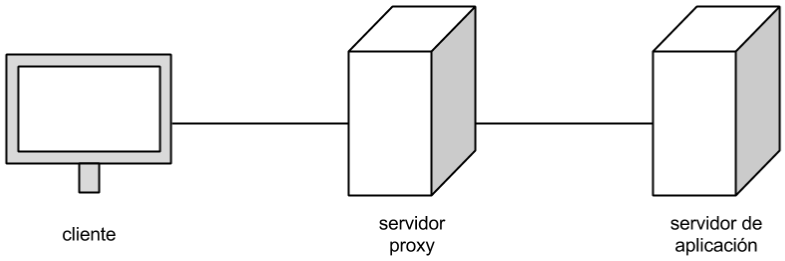

Figura 5.1: Arquitectura cliente-servidor

También debemos notar que la utilización de Django para el desarrollo de nuestra aplicación nos ha obligado a adoptar la arquitectura *MVC*. Habitualmente, la implementación de esta arquitectura sigue el siguiente flujo:

1. El usuario interacciona con la interfaz y el controlador recibe un evento con la acción realizada por el usuario y la solicitud asociada a esa acción.

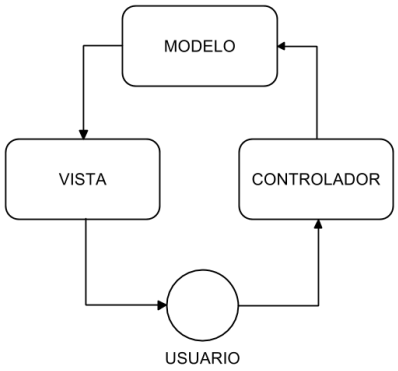

Figura 5.2: Arquitectura Modelo-Vista-Controlador

2. El controlador accede al modelo, para actualizarlo con nueva información o para acceder a la información que ya contiene.

3. El modelo pasa la información necesaria a la vista correspondiente. En algunas implementaciones la vista no puede acceder directamente a la información del modelo; el controlador hace de intermediario entre la vista y el modelo.

4. La vista, con la información del modelo, genera la nueva interfaz de usuario.

Pero, como ya hemos dicho en el apartado 4.4, Django no implementa la arquitectura *MVC* completamente. La arquitectura implementada por Django queda de la siguiente forma:

1. El usuario interacciona con la interfaz generando una solicitud a una URL determinada. Es el propio *framework* el encargado de redirigir esa solicitud a la vista correspondiente.

2. Desde la vista se accede al modelo para actualizar o recuperar la información necesaria.

3. La vista pasa al *template* correspondiente la información que debe mostrar al usuario.

4. El *template* se completa con la información y se actualiza la interfaz.

Cuando el usuario interacciona con la aplicación, la solicitud que se genera la recoge Django y siguiendo el fichero `urls.py` del proyecto y la aplicación dirige esta solicitud a la vista que corresponda.

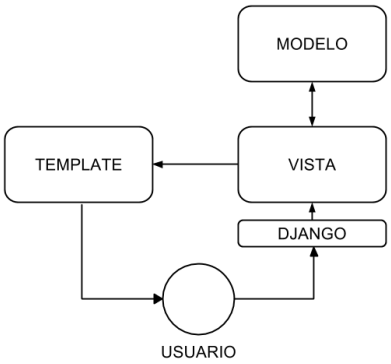

Figura 5.3: Arquitectura utilizada por Django

En el momento que en la vista ya están preparados los datos que deben mostrarse al usuario, se debe pasar al *template* correspondiente estos datos en un diccionario, de forma que cuando se genere el fichero, en este caso, HTML se sustituyan las variables por su valor en el diccionario.

Capítulo 6

Proceso

En los siguientes subapartados se detalla todo el proceso que tiene lugar cuando el usuario interacciona con la aplicación web.

Figura 6.1: Captura de la web

6.1. Grabar y subir un audio

El usuario puede elegir entre subir un fichero de audio que contenga la consulta o grabarla desde la misma web utilizando un micrófono. Cuando se opta por grabar el audio se utiliza el plugin Recorder.js. Este plugin está im-

plementado en JavaScript y permite grabar y exportar audio en páginas web. También utilizaremos algunos componentes de HTML5, como `getUserMedia`, que permite recoger audio del micrófono o vídeo de la cámara. Esto permite que el plugin tenga acceso al micrófono.

Para grabar un audio, el usuario debe pulsar primero el botón "Grabar". Una vez pulsado, el navegador mostrará un mensaje que advierte al usuario que la página quiere acceder al micrófono. Para continuar, debe aceptar este mensaje.

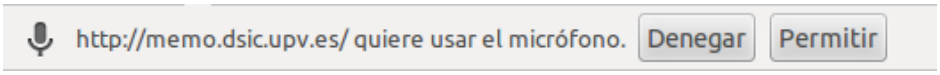

Figura 6.2: Mensaje de Google Chrome para permitir el acceso al micrófono

Una vez aceptado, comenzará la grabación del audio que se detendrá cuando se pulse el botón "Parar" (solo se puede pulsar este botón durante la grabación).

Si el usuario opta por subir un fichero, tendrá que seleccionarlo pulsando el botón "Seleccionar fichero" y pulsar el botón "Subir"(solo se puede pulsar este botón si hay un fichero seleccionado).

Cuando el usuario finaliza la grabación del audio o selecciona un fichero, se lanzará una petición POST al servidor, que contendrá el audio, la opción de idioma (Autodetectar, Español, Francés o Inglés) y mediante que método se ha obtenido el audio (grabación o fichero).

6.2. Detección del idioma

Una vez que el audio esté en el servidor, y antes de detectar el idioma, realizaremos una serie de comprobaciones. Primero comprobaremos que es un formato admitido (en este caso solo aceptaremos ficheros de formato WAV) y que el tamaño no es superior al máximo permitido (el máximo en este caso es de cinco *megabytes*). El tamaño máximo se ha establecido para que el proceso de comprensión no se alargue en el tiempo. En el caso del formato de audio, solo aceptamos WAV porque es una restricción de `HDecode`.

Si no se cumplen las dos condiciones se alertará al usuario con un mensaje, que dependerá según el método escogido para subir el audio.

Si el usuario ha indicado que el idioma debe detectarse a partir del propio audio, tenemos que continuar y obtener la decodificación acústico-fonética. Obtendremos esta decodificación con `HDecode`, que a partir de un fichero de audio y un modelo acústico nos devuelve la decodificación en fonemas del audio.

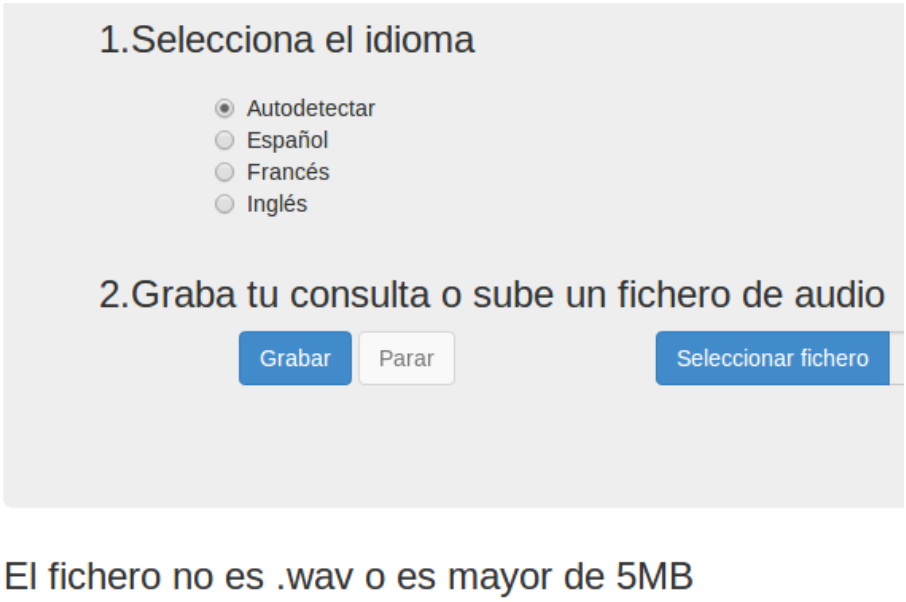

Figura 6.3: Mensaje de error del fichero

Aunque `HDecode` espera un fichero WAV, este también tiene que cumplir dos requisitos más: que sean de un solo canal o mono y que la frecuencia de muestreo sea de 16 000 Hz. Por eso, debemos convertir el audio original a otro que cumpla esos requisitos. Para ello, se utiliza el programa *Sound eXchange* (*SoX*).

Con este audio obtenemos la decodificación acústico-fonética que utilizaremos para detectar el idioma. Para detectar el idioma utilizaremos tres modelos de lenguaje, uno por cada idioma, que modelan la probabilidad de los fonemas en cada uno de los idiomas. Estos modelos de lenguaje son de trigramas de fonemas.

El idioma del audio será aquel cuyo modelo de lenguaje tenga una menor perplejidad asociada. La perplejidad de cada modelo se obtendrá mediante *SRI Language Modeling Toolkit* (*SRILM*) y, en concreto, con el programa `ngram`. A partir de las perplejidades obtenidas mediante este programa, asignaremos el idioma al audio.

6.3. Transcripción del audio

La parte de detección de idioma es fundamental para obtener la transcripción a texto del audio. Para la fase de reconocimiento del habla, utilizaremos un servicio de Google que a partir de un audio y el idioma nos devuelve una

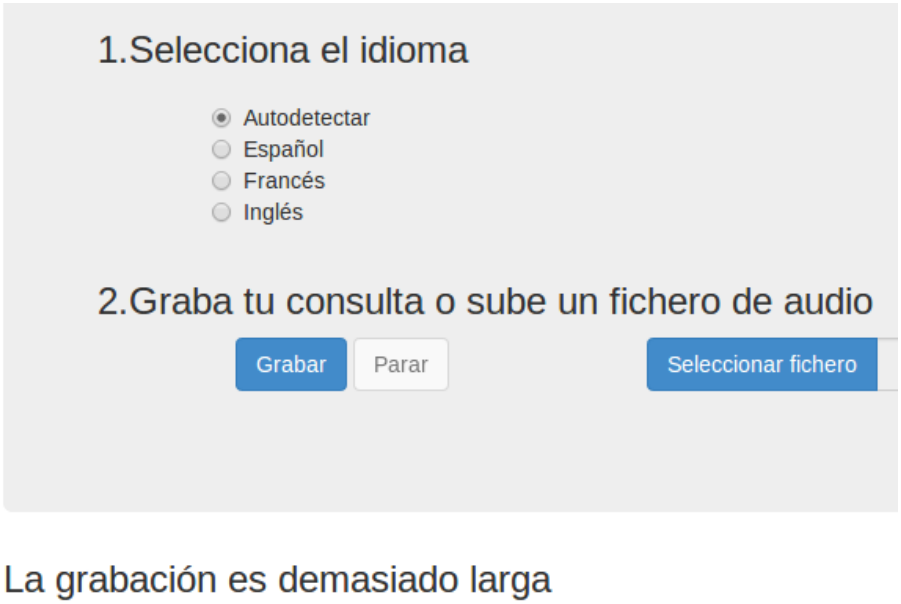

Figura 6.4: Mensaje de error de la grabación

serie de posibles transcripciones ordenadas en orden decreciente de probabilidad.

Hay que tener en cuenta que este servicio solo permite 50 peticiones al día de forma gratuita. Además, es necesario registrarse como desarrollador y obtener una clave para que Google controle el número de peticiones diarias.

Este control es reciente, ya que hasta hace poco estaba disponible la versión 1 de la *Application Programming Interface* (*API*), que no limitaba el número de peticiones. Al quitar esta posibilidad, es necesario utilizar esta nueva versión (versión 2) con todo lo que ello conlleva.

Según esta API, para obtener la lista de posibles transcripciones, debemos realizar un POST con el audio a esta URL:

```
https://www.google.com/speech-api/v2/recognize?key=clave&xjerr=
  1&client=chromium&lang=idioma&maxresults=max&pfilter=False
```

siendo *clave* la obtenida en Google Developers Console para poder utilizar la *API* de reconocimiento de voz, *idioma* el detectado en el paso anterior y *max* el número máximo de hipótesis que se obtendrán.

También debe tenerse en cuenta que Google espera el audio en formato FLAC, de forma que se utilizará de nuevo el programa SoX para convertir el audio original.

6.4. Traducción

Una vez tengamos las posibles transcripciones del audio, necesitamos traducir al español (si el audio es en inglés o francés) ya que el reconocedor solo admite este idioma.

Para traducir las transcripciones utilizaremos MOSES. MOSES es un sistema de traducción estadística que a partir de un modelo entrenado de un par de idiomas es capaz de obtener la traducción más probable del texto. Los modelos utilizados por MOSES se obtienen a partir de textos traducidos.

Utilizando MOSES se puede obtener la traducción más probable o un número máximo de traducciones. En este último caso cada traducción va acompañada de una puntuación, de forma que una mayor puntuación conlleva una mayor confianza en la traducción.

6.4.1. Aprender un modelo de traducción

Para que MOSES pueda traducir de un idioma a otro primero necesita aprender un modelo de traducción. Para poder generar este modelo necesitamos un corpus paralelo, es decir, un conjunto de frases en los dos idiomas involucrados en la traducción.

Conseguir el modelo de traducción es muy sencillo, solo tenemos que utilizar un script en Perl incluido en MOSES (`train-model.perl`) indicando los corpus que debe utilizar y la dirección de la traducción. Sin embargo, los ficheros con las frases deben cumplir algunos requisitos:

- Una frase por línea sin líneas en blanco.
- Las frases de más de 100 palabras deben ser eliminadas. También serán eliminadas sus traducciones.
- Todas las frases estarán en minúscula.

El resultado de este script son tres ficheros que forman el modelo de traducción que después utilizará MOSES. Estos ficheros son: `moses.ini`, fichero de configuración que contiene los parámetros necesarios, `phrase-table.gz`, fichero con la tabla de traducción y un modelo de lenguaje del idioma destino.

Es en la tabla de traducción donde aparecen las posibles traducciones y los datos necesarios para el calculo de la mejor traducción que se han obtenido a partir del corpus paralelo. En la tabla se recogen las frases del corpus indicando la frase en el idioma de origen y en el idioma de destino, además de información como el alineamiento de las frases. La estructura de la tabla de traducción es la siguiente:

```
source ||| target ||| scores ||| alignment ||| counts
```

siendo `source` la palabra o palabras a traducir, `target` la posible traducción de `source` y `alignment` el alineamiento de las palabras.

Cada línea representa una frase del corpus, por lo que un ejemplo del contenido de la tabla es el siguiente:

```
next week ||| la próxima semana ||| 0.485714 0.212745 0.160377
        0.00127271 2.718 ||| 0-0 0-1 1-2 ||| 35 106 17
```

Según esta entrada de la tabla, una posible traducción de "*next week*" es "la próxima semana". El alineamiento nos dice que la traducción de "*next*" es "la próxima" y la traducción de "*week*" es "semana".

6.5. Comprensión

Con las transcripciones en español, bien obtenidas directamente de Google o traducidas con MOSES si el idioma original es inglés o francés, llegamos a la fase de comprensión.

El comprendedor espera las posibles hipótesis en forma de grafo, por lo tanto, se debe convertir el fichero de transcripciones en texto plano a otro que las represente en forma de grafo. En el caso del español se utilizan todas las hipótesis devueltas por Google (máximo de veinte) y en inglés y francés todas las posibles traducciones obtenidas por MOSES (máximo de diez).

En este momento el comprendedor, a partir del fichero que representa el grafo, se encarga de separar el texto en diferentes segmentos y asignar a cada uno de ellos una categoría (las categorías aparecen en la tabla 6.1).

6.5.1. Segmentación más probable

El comprendedor debe ser capaz de calcular la mejor secuencia de conceptos a partir del grafo que contiene todas las posibles transcripciones del audio. Para realizar este proceso, la comprensión se divide en dos etapas.

En la primera etapa, el módulo de comprensión parte del grafo de palabras que representa las posibles traducciones para generar un grafo de conceptos. Este segundo grafo tendrá el mismo número de nodos que el grafo de palabras. Cada arco representará un concepto asociado con un conjunto de palabras; el arco saldrá del nodo desde el que comienza la secuencia de palabras y llegará al nodo en el que termina la secuencia de palabras asociada al concepto.

En la segunda etapa de comprensión, se utilizarán tanto el grafo de las traducciones como el de conceptos que el comprendedor ha generado para

Categorías	
<afirmacion>	m_salida
ciudad	nada_?
ciudad_destino	nada
ciudad_origen	<negacion>
clase_billete	<no_entendido>
coletilla	nombre_atributo
consulta	not
cortesia	numero_relativo_orden
<duracion>	<precio>
fecha	precio
<hora>	<servicios>
hora	servicios
<hora_llegada>	<tipo_tren>
<hora_salida>	tipo_tren
m_llegada	tipo_viaje

Tabla 6.1: Categorías del reconocedor

obtener la secuencia de conceptos. Para ello, se recorren ambos grafos y se calcula el camino de mayor probabilidad, de forma que la salida es la secuencia más probable de conceptos con sus palabras asociadas.

La salida que proporciona el módulo de comprensión está determinada por la siguiente ecuación:

$$\hat{C} = \arg\max_{C \in \mathcal{C}^*} p(C|A) \tag{6.1}$$

siendo C una secuencia de conceptos y A el audio de entrada del sistema.

En este caso, al tratarse de un sistema de comprensión multilingüe tenemos que el audio está en un idioma diferente al utilizado en el comprendedor y necesitamos traducirlo al idioma del comprendedor. Si definimos como W_s la transcripción en palabras del audio en el idioma original y W_t la traducción al idioma del comprendedor, podemos redefinir la ecuación 6.1 de la siguiente forma:

$$\hat{C} = \arg\max_{C \in \mathcal{C}^*} \max_{W_s, W_t} p(C, W_s, W_t|A) \tag{6.2}$$

Aplicando la regla de Bayes y realizando una serie de suposiciones podemos llegar a la siguiente ecuación:

$$\hat{C} = \arg\max_{C \in \mathcal{C}^*} \max_{W_t} p(W_t|A) \cdot p(W_t|C) \cdot p(C) \tag{6.3}$$

Será esta ecuación la que determine la salida de nuestro sistema de comprensión.

6.6. Representación basada en marcos

Una vez el comprendedor ha obtenido la segmentación del texto, queda un último paso antes de devolver al usuario el resultado del sistema de comprensión. Este paso consiste en pasar la segmentación a una representación basada en marcos o *frames*.

Tras esto, solo queda mostrar al usuario el resultado obtenido. El servidor envía como respuesta un documento HTML que contiene el idioma detectado, las hipótesis que Google ha devuelto, la traducción con mejor puntuación (si procede) y la segmentación devuelta por el reconocedor.

Capítulo 7

Experimentación

En este capítulo se van a describir una serie de experimentos realizados con el objetivo de mejorar el sistema en diversos aspectos. Primero veremos si es posible reducir el tiempo que necesita nuestro sistema para mostrar los resultados, ya que el tiempo total no puede ser excesivo. Después intentaremos mejorar la etapa de detección de idioma, porque un error en esta fase condicionará en gran medida el resultado final.

7.1. Parametrización DAF

Durante la comprobación del correcto funcionamiento del sistema, hemos visto como el tiempo total desde que se sube el fichero de audio hasta que se muestran por pantalla los resultados es alto.

Para ver cual es la etapa crítica del sistema, hemos utilizado una frase grabada y hemos medido el tiempo en cada etapa. Indicamos a continuación el tiempo dividido según etapas mediante una gráfica (figura 7.1) y una tabla (7.1).

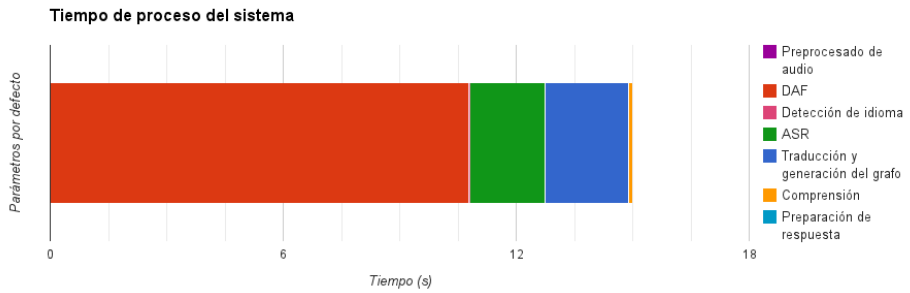

Figura 7.1: Tiempo de proceso del sistema

Etapa	Tiempo (s)
Preprocesado de audio	0,004128
Decodificación acústico-fonética	10,775271
Detección de idioma	0,029139
Reconocedor del habla	1,932643
Traducción y generación del grafo	2,148666
Comprensión	0,122848
Preparación de la respuesta	0,000075

Tabla 7.1: Tiempo de proceso del sistema

Como podemos ver, gran parte del tiempo de proceso se emplea en la decodificación acústico-fonética, por lo que si queremos reducir el tiempo total del sistema, es necesario centrarnos en esta etapa.

Para este experimento utilizamos un subconjunto del corpus DIHANA para el inglés y el francés y un subconjunto del corpus ALBAYCÍN para el español. El número total de audios es el reflejado en la tabla 7.2.

Idioma	Número de audios
Español	1400
Inglés	1336
Francés	508

Tabla 7.2: Número de audios utilizados

Al realizar la decodificación acústico-fonética se indican tres parámetros numéricos: la anchura de la poda (*pruning beam width*), el factor de escala del modelo del lenguaje (*LM scale factor*) y la penalización por inserción de palabra (*word insertion penalty*).

Estos tres parámetros influyen tanto en el resultado de la decodificación como en el tiempo necesario, por lo que nos planteamos que configuración es mejor. Nuestro objetivo es reducir el tiempo total, pero es posible que al hacerlo el resultado de la decodificación sea peor y, por lo tanto, influya en la detección de idioma. Por este motivo, intentaremos llegar a un equilibrio entre el tiempo y el número de aciertos en la detección de idioma.

Los valores más adecuados de la decodificación dependen en gran medida de la tarea, por lo que se han probado diferentes configuraciones. Durante la realización de este experimento se ha podido comprobar como la penalización por inserción de palabra no tenía efectos significativos tanto en tiempo como en precisión, lógico teniendo en cuenta que la duración de las palabras (en nuestro caso, fonemas) son similares.

Para los resultados que se muestran, se han variado los parámetros de la siguiente forma: en el caso de la anchura de la poda se han variado los valores en un rango de 300 a 900 con incrementos de 150 y para el factor de escala del modelo de lenguaje se ha probado con los valores 0, 9 y 18. Los resultados se pueden ver en las tablas 7.3, 7.4 y 7.5 y en las figuras 7.2 y 7.3

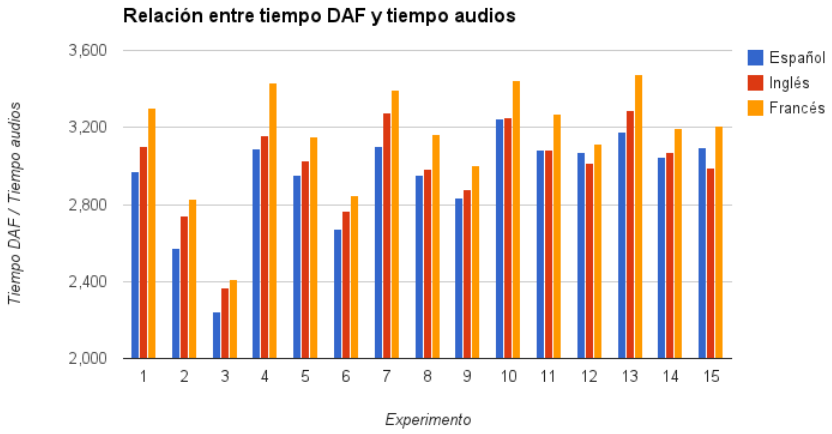

Figura 7.2: Tiempos de decodificación acústico-fonética

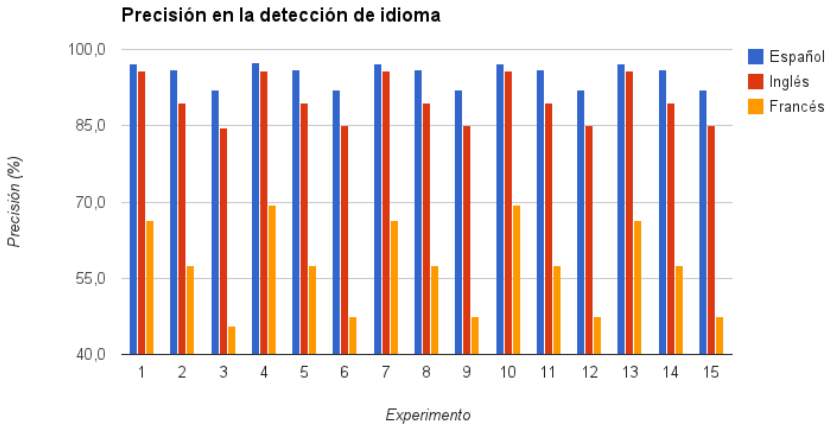

Figura 7.3: Precisión en la detección de idioma

A partir de estos resultados podemos extraer varias conclusiones:

- Los mejores resultados de detección de idioma se dan en el idioma

Exp.	*Pruning beam width*	*LM scale factor*	Tiempo DAF / Tiempo audios	Precisión (%)
1	300	0	2,971	97,1
2	300	9	2,572	96,0
3	300	18	**2,245**	92,1
4	450	0	3,089	**97,5**
5	450	9	2,950	96,0
6	450	18	2,672	92,1
7	600	0	3,103	97,1
8	600	9	2,953	96,0
9	600	18	2,837	92,1
10	750	0	3,247	97,1
11	750	9	3,086	96,0
12	750	18	3,073	92,1
13	900	0	3,179	97,1
14	900	9	3,049	96,0
15	900	18	3,095	92,1

Tabla 7.3: Resultados para el idioma español

español; esto se debe a que la decodificación se realiza con fonemas de este idioma. El francés es el idioma con peores resultados.

- La anchura de la poda influye en los resultados de la siguiente forma: cuanto mayor es este parámetro el tiempo se incrementa ligeramente sin que ello suponga grandes cambios en la precisión. Esto se puede deber a que el parámetro de poda es demasiado alto como para podar.

- El factor de escala del modelo de lenguaje hace que el tiempo necesario para realizar la decodificación acústico-fonética disminuya a costa de la precisión. Por ejemplo, podemos comparar los resultados de los experimentos 1 y 3 del español. Al pasar el factor de escala de 0 a 18, el tiempo se reduce un 24,45 % mientras la precisión desciende cinco puntos.

Como el objetivo es llegar a un compromiso entre el tiempo de decodificación y la precisión de esta, se han escogido como parámetros los correspondientes al experimento 2, ya que el tiempo medio de decodificación es de los más bajos sin que suponga que la precisión se reduzca en exceso.

Exp.	*Pruning beam width*	*LM scale factor*	Tiempo DAF / Tiempo audios	Precisión (%)
1	300	0	3,099	**95,8**
2	300	9	2,741	89,5
3	300	18	**2,367**	84,6
4	450	0	3,159	**95,8**
5	450	9	3,029	89,5
6	450	18	2,766	85,0
7	600	0	3,277	**95,8**
8	600	9	2,986	89,5
9	600	18	2,876	85,0
10	750	0	3,254	**95,8**
11	750	9	3,081	89,5
12	750	18	3,013	85,0
13	900	0	3,286	**95,8**
14	900	9	3,071	89,5
15	900	18	2,992	85,0

Tabla 7.4: Resultados para el idioma inglés

7.2. Cambio del modelo acústico

Hasta ahora, el modelo acústico que hemos utilizado presenta un pequeño inconveniente: ha sido entrenado con el corpus TC-STAR español, por lo que estamos obteniendo transcripciones fonéticas del inglés y del francés a partir de fonemas castellanos. Esto puede suponer una peor detección de ambos idiomas, por lo que nos planteamos la posibilidad de entrenar un nuevo modelo acústico con fonemas universales.

Para que fuera posible entrenar este modelo acústico universal necesitaríamos, además de los fonemas, la forma de obtener la representación en estos fonemas de los audios con los que entrenamos este modelo. Como, en nuestro caso, solo disponemos de un programa para el español capaz de pasar una transcripción textual a otra fonética, no podemos implementar esta aproximación. Sin embargo, si tenemos a nuestro alcance un diccionario para el inglés de palabras y su correspondencia a fonemas de las palabras contenidas en el corpus DIHANA, por lo que podríamos entrenar un único modelo acústico para el español y el inglés.

Por ello, lo que hemos hecho es generar un nuevo modelo acústico a partir de los audios en español e inglés del corpus DIHANA y comparar los resultados en la detección de idioma con el modelo acústico ya entrenado con el corpus TC-STAR.

Exp.	*Pruning beam width*	*LM scale factor*	Tiempo DAF / Tiempo audios	Precisión (%)
1	300	0	3,301	66,3
2	300	9	2,826	57,4
3	300	18	**2,412**	45,5
4	450	0	3,431	**69,3**
5	450	9	3,154	57,4
6	450	18	2,848	47,5
7	600	0	3,394	66,3
8	600	9	3,162	57,4
9	600	18	3,000	47,5
10	750	0	3,446	**69,3**
11	750	9	3,270	57,4
12	750	18	3,113	47,5
13	900	0	3,478	66,3
14	900	9	3,198	57,4
15	900	18	3,209	47,5

Tabla 7.5: Resultados para el idioma francés

Para obtener este segundo modelo acústico es necesario el conjunto de audios y sus correspondientes transcripciones fonéticas. En el caso del español, se parte de la transcripción en palabras de cada audio y se utiliza el programa `ort2fon`, que devuelve la transcripción fonética de un texto. Para el inglés, como hemos comentado anteriormente, partiremos de un diccionario en el que aparecen las transcripciones fonéticas de todas las palabras del corpus. De esta forma, los fonemas utilizados en este nuevo modelo serán la unión del conjunto de fonemas del castellano y del inglés.

El conjunto de audios disponibles se ha dividido en una partición de entrenamiento con el 80 % de los audios y una partición de prueba con el 20 % restante. Con esta partición de test mediremos la precisión de los modelos y podremos comparar ambos. Aunque para entrenar el modelo no utilizaremos los audios en francés de DIHANA al no disponer de las transcripciones fonéticas, separaremos igualmente el corpus en este idioma en entrenamiento y prueba para después entrenar el modelo de lenguaje correspondiente.

Una vez obtenido el nuevo modelo acústico, necesitamos obtener el modelo de lenguaje que después se utilizará para realizar la decodificación acústico-fonética. Para ello, utilizamos SRILM con los textos de entrenamiento de inglés y español transcritos fonéticamente y tendremos un modelo de trigramas.

Por último, entrenamos los modelos de lenguaje que después se utilizarán

para detectar el idioma. Los modelos los entrenamos a partir de los resultados de la decodificación acústico-fonética de la partición de entrenamiento de los tres idiomas.

En este momento ya podemos detectar el idioma de un audio con el nuevo modelo acústico, por lo que nos disponemos a comparar los resultados de los dos modelos acústicos que tenemos, de forma que, utilizando la partición de test, vamos a medir la precisión de la identificación del idioma utilizando la decodificación acústico-fonética generada con cada modelo acústico.

Los resultados los podemos observar en las tablas 7.6 y 7.7.

		Idioma detectado		
		Español	Francés	Inglés
Idioma audios	Español	**79 %**	13 %	8 %
	Francés	25 %	**67 %**	8 %
	Inglés	25 %	8 %	**66 %**

Tabla 7.6: Detección de idioma con modelo TC-STAR

		Idioma detectado		
		Español	Francés	Inglés
Idioma audios	Español	**99,4 %**	0,3 %	0,3 %
	Francés	5 %	**82 %**	13 %
	Inglés	3 %	4,1 %	**92,9 %**

Tabla 7.7: Detección de idioma con modelo DIHANA

A partir de estos resultados podríamos deducir que la detección de idioma es mejor con el modelo acústico nuevo, sin embargo, en la práctica no se cumple. Tras probar el sistema con audios nuevos podemos ver como, por ejemplo, la detección del español es peor con el modelo nuevo.

Para confirmar con datos este comportamiento, hemos escogido dos audiolibros de dominio público (uno para cada idioma) para utilizarlos como conjunto de test con los modelos anteriores. Para cada uno de los audiolibros, se han obtenido fragmentos de diez segundos de duración. Los resultados indican que en ambos casos la detección de idioma es incorrecta en gran medida. Pero un dato que llama la atención es que en ambos modelos, la perplejidad asociada al modelo de lenguaje del español es mucho mayor que el francés o el inglés (llegando a ser la perplejidad del español 2 ordenes de magnitud superior a la del inglés o francés).

7.3. Detección de idioma con SVM

En esta sección vamos a ver la posibilidad de modificar la etapa de detección de idioma de nuestro sistema para utilizar clasificadores basados en Máquinas de Soporte Vectorial o *SVM* como ya hemos explicado en el apartado 3.2.2.

Para implementar los clasificadores vamos a emplear un *toolkit* de aprendizaje automático para Python llamado *scikit-learn*. Este *toolkit* contiene, entre otros, diferentes algoritmos de clasificación. Nosotros nos centraremos en las diferentes implementaciones de clasificadores basados en *SVM*. Los clasificadores están representados en las clases `SVC`, `NuSVC` y `LinearSVC`. Las dos primeras siguen una estrategia de "uno contra uno", mientras la última emplea una estrategia "uno contra el resto". Aunque las dos primeras son similares, no utilizan los mismos parámetros y difieren en su formulación matemática.

En un principio vamos a utilizar únicamente los clasificadores `SVC` con un *kernel* lineal y `LinearSVC` con el resto de parámetros por defecto. Con estos clasificadores vamos a ver el resultado de clasificar el 20 % de test de DIHANA entrenando las *SVM* con el 80 % restante. Tal y como se ha comentado en el apartado 3.2.2, necesitamos vectorizar los datos de entrada para poder utilizar un *SVM* y para ello utilizaremos la métrica *tf.idf*. Utilizar esta métrica es muy sencillo con *scikit-learn*, ya que tiene implementado el vectorizador `TfidfVectorizer`.

Los resultados de la detección de idioma con los diferentes clasificadores utilizando el vectorizador antes mencionado con 4-gramas de caracteres son los siguientes:

		Idioma detectado		
		Español	Francés	Inglés
Idioma audios	Español	**99,9 %**	0 %	0,1 %
	Francés	5 %	**87 %**	8 %
	Inglés	2,6 %	1,5 %	**95,9 %**

Tabla 7.8: Detección de idioma con `LinearSVC`

Los resultados son similares con ambos tipos de clasificadores. La diferencia más importante es el tiempo de entrenamiento: mientras que el `SVC` necesita casi seis segundos, el `LinearSVC` entrena en dos décimas de segundo.

Ahora, igual que en el apartado anterior (7.2), vamos a utilizar como conjunto de test fragmentos de audiolibros en los tres idiomas para comprobar si la detección de idioma es correcta.

		Idioma detectado		
		Español	Francés	Inglés
Idioma audios	Español	**99,9 %**	0 %	0,1 %
	Francés	4 %	**86 %**	10 %
	Inglés	3 %	1,1 %	**95,9 %**

Tabla 7.9: Detección de idioma con `SVC` con *kernel* lineal

		Idioma detectado		
		Español	Francés	Inglés
Idioma audios	Español	**11 %**	23 %	66 %
	Francés	1 %	**89,7 %**	9,3 %
	Inglés	0 %	19,3 %	**80,7 %**

Tabla 7.10: Detección de idioma con `LinearSVC`

		Idioma detectado		
		Español	Francés	Inglés
Idioma audios	Español	**1 %**	15,7 %	83,3 %
	Francés	0,7 %	**88 %**	11,3 %
	Inglés	0 %	21 %	**79 %**

Tabla 7.11: Detección de idioma con `SVC` con *kernel* lineal

Como podemos ver en las tablas 7.10 y 7.11, aunque la detección del francés e inglés es buena, la detección del español es muy baja, por lo que debemos replantearnos este posible método de clasificación de idioma.

Nos planteamos si la detección del idioma puede ser mejor si intercambiamos los corpus de entrenamiento y de test, es decir, si utilizamos los audiolibros para entrenar el clasificador y el corpus DIHANA para comprobar el rendimiento. Los resultados que obtenemos con los dos tipos de clasificadores son los siguientes:

		Idioma detectado		
		Español	Francés	Inglés
Idioma audios	Español	**58,6 %**	35,3 %	6,1 %
	Francés	1,2 %	**68,5 %**	30,3 %
	Inglés	6 %	30,2 %	**63,8 %**

Tabla 7.12: Detección de idioma con `LinearSVC`

Como observamos en las tablas 7.12 y 7.13, utilizando los audiolibros como datos de entrenamiento de los clasificadores la detección del idioma

		Idioma detectado		
		Español	Francés	Inglés
Idioma audios	Español	**64 %**	18,2 %	17,8 %
	Francés	0,8 %	**57,7 %**	41,5 %
	Inglés	2,2 %	18 %	**79,8 %**

Tabla 7.13: Detección de idioma con `SVC` con *kernel* lineal

es, por lo general, buena para los tres idiomas. Partiendo de estos resultados, puede ser buena idea modificar los parámetros que actualmente están definidos por defecto y ver cuanto pueden mejorar. El *toolkit* que estamos empleando integra la clase `GridSearchCV`, que permite probar un conjunto de parámetros en un clasificador e indicar el mejor según una puntuación (por ejemplo, precisión o *recall*) obtenida mediante validación cruzada.

Vamos a utilizar `GridSearchCV` para comprobar si podemos mejorar el resultado del clasificador modificando sus parámetros. Partiremos con un clasificador `SVC`, indicando diferentes parámetros (incluyendo distintos *kernels*) y escogeremos aquel con mayor *recall*. Los resultados, divididos según el tipo de *kernel*, son los siguientes:

	Recall
$C = 1$	99,89
$C = 10$	99,78
$C = 100$	99,78
$C = 1000$	99,78
$C = 10000$	99,78

Tabla 7.14: *Recall* según parámetros con *kernel* lineal

	$gamma = 0,1$	$gamma = 1$	$gamma = 10$	$gamma = 100$
$C = 1$	99,56	99,78	87,33	85,89
$C = 10$	99,78	99,78	87,33	85,89
$C = 100$	99,78	99,78	87,33	85,89
$C = 1000$	99,78	99,78	87,33	85,89
$C = 10000$	99,78	99,78	87,33	85,89

Tabla 7.15: *Recall* según parámetros con *kernel* radial

		$gamma = 0,1$	$gamma = 1$	$gamma = 10$
$C = 1$	$coef0 = -10$	93,89	90,22	87,22
	$coef0 = 0$	99,56	99,89	98,33
	$coef0 = 10$	93,89	93,78	90,56
$C = 10$	$coef0 = -10$	93,89	90,22	87,33
	$coef0 = 0$	99,89	99,89	94,11
	$coef0 = 10$	93,89	93,78	90,56
$C = 100$	$coef0 = -10$	93,89	90,22	87,33
	$coef0 = 0$	99,78	99,89	89,67
	$coef0 = 10$	93,89	93,78	90,56
$C = 1000$	$coef0 = -10$	93,89	90,22	87,33
	$coef0 = 0$	99,78	99,89	88,67
	$coef0 = 10$	93,89	93,78	90,56
$C = 10000$	$coef0 = -10$	93,89	90,22	87,33
	$coef0 = 0$	99,78	99,89	88,56
	$coef0 = 10$	93,89	93,78	90,56

Tabla 7.16: *Recall* según parámetros con *kernel sigmoid*

		$grado = 1$	$grado = 2$	$grado = 3$
$coef0 = -100$	$gamma = 1$	99,89	0	99,78
	$gamma = 10$	99,78	0	99,89
	$gamma = 100$	99,78	0	99,89
$coef0 = -10$	$gamma = 1$	99,89	0	99,89
	$gamma = 10$	99,78	0	99,78
	$gamma = 100$	99,78	0	99,78
$coef0 = 0$	$gamma = 1$	99,89	99,78	99,67
	$gamma = 10$	99,78	99,78	99,67
	$gamma = 100$	99,78	99,78	99,67
$coef0 = 10$	$gamma = 1$	99,89	99,78	99,78
	$gamma = 10$	99,78	99,78	99,78
	$gamma = 100$	99,78	99,78	99,67
$coef0 = 100$	$gamma = 1$	99,89	99,78	99,78
	$gamma = 10$	99,78	99,78	99,78
	$gamma = 100$	99,78	99,78	99,78

Tabla 7.17: *Recall* según parámetros con *kernel* polinómico

Con los datos de las tablas anteriores comprobamos como los mejores resultados de *recall* se dan, entre otros, con la configuración por defecto del *kernel* lineal. Por este motivo, vamos a comparar el resultado del SVC con

esta configuración con los resultados de aplicar la aproximación que, hasta ahora, estamos utilizando. En la siguiente tabla comparamos los resultados en la identificación de idioma separados por idiomas para ambas aproximaciones:

	Mínima perplejidad	SVC con kernel lineal
Español	97,1 %	99,9 %
Francés	66,3 %	86 %
Inglés	95,8 %	95,9 %

Tabla 7.18: Precisión en identificación de idioma según clasificador

Según se puede apreciar en la tabla 7.18, con los parámetros iniciales de la decodificación acústico-fonética el clasifcador basado en *SVM* ofrece mejores resultados, sobretodo para el francés. Quedaría probar si esto se cumple en la realidad o no, como ocurría con la experimentación anterior.

Capítulo 8

Conclusiones y trabajo futuro

8.1. Conclusiones

En este punto, vamos a revisar los objetivos que nos hemos planteado al comienzo de este trabajo (apartado 1.2) y comprobar si los hemos cumplido.

En cuanto al primer objetivo, plantear una arquitectura de comprensión multilingüe, durante el capítulo 2, partiendo de una arquitectura clásica y pasando por una arquitectura ampliada, llegamos a esta arquitectura que posteriormente hemos implementado. Con respecto a la introducción de un módulo de identificación de idioma, es en el capítulo 3 donde vemos dos posibles aproximaciones para el desarrollo de este módulo.

En el caso del segundo objetivo, desarrollar una aplicación web que contenga el sistema de comprensión y capaz de recoger la consulta del usuario, hemos cumplido el objetivo desarrollando una aplicación, que utilizando tecnologías recientes como HTML5 es capaz de grabar la consulta del usuario, enviar esa consulta al sistema de comprensión y mostrar por pantalla los resultados de este sistema.

También vamos a extraer conclusiones de la experimentación realizada y detallada en el capítulo anterior (capítulo 7).

En la primera experimentación hemos visto como la etapa más lenta del sistema es la decodificación acústico-fonética que realizamos para identificar el idioma. Para intentar reducir los tiempos, planteamos la posibilidad de modificar algunos parámetros de la decodificación como la anchura de la poda *pruning beam width*) y el factor de escala del modelo del lenguaje (*LM scale factor*. A partir de los resultados, comprobamos que la modificación de estos parámetros puede significar una reducción de tiempos de hasta un 30 %. Sin embargo, reducir el tiempo de decodificación puede afectar negativamente a la identificación del idioma, por lo que es necesario escoger un conjunto de

parámetros que reduzca el tiempo de proceso sin que la identificación de idioma se vea excesivamente perjudicada.

En la segunda experimentación, nos planteamos la posibilidad de cambiar el modelo acústico utilizado en la decodificación por otro modelo universal. Sin los recursos necesarios para llevar a cabo esta posibilidad nos planteamos la posibilidad de generar un modelo acústico para español, inglés y francés. Al no disponer de las transcripciones fonéticas de texto en francés pero sí para el español y el inglés, nos planteamos la posibilidad de aprender un modelo acústico de español e inglés. Tras comparar los resultados de la fase de identificación de idioma con el modelo hasta ahora utilizado (TC-STAR) y el nuevo (DIHANA), parece que el modelo de inglés y español da mejores resultados. Sin embargo, en la práctica no se cumple, ya que el español se confunde con el inglés. En este caso, el cambio de modelo no mejora los resultados anteriores, aunque es cierto que nos hemos alejado del primer planteamiento que es un modelo universal.

Por último, en la tercera experimentación, contemplamos la posibilidad de cambiar la fase de identificación del idioma siguiendo la aproximación descrita en el apartado 3.2.2. Después de probar numerosas configuraciones, hemos visto como el clasificador basado en *SVM* ofrece, aparentemente, mejores resultados en la identificación de idioma que la aproximación con modelos de lenguaje.

8.2. Trabajo futuro

Dado que el proyecto aquí descrito consiste en el desarrollo de un prototipo de comprensión del habla multilingüe, podemos decir que queda trabajo por delante. No solo continuar el desarrollo del sistema, si no que también podemos plantear diferentes mejoras al prototipo actual.

- Es deseable reducir el tiempo de la fase de identificación del idioma. Aunque durante la experimentación hemos conseguido reducir el tiempo de la decodificación acústico-fonética, no hemos conseguido reducir el tiempo a menos del doble del tiempo real del audio.

- También sería adecuado mejorar el nivel de acierto en la fase de identificación del idioma. El nivel de acierto del idioma español es correcto, pero en el caso del inglés y el francés es relativamente bajo. Teniendo en cuenta que del idioma identificado dependen las etapas de reconocimiento y traducción, es muy importante que la identificación del idioma sea correcta.

Bibliografía

[1] Philipp Koehn, Hieu Hoang, Alexandra Birch, et al. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, pages 177–180. Association for Computational Linguistics, 2007.

[2] Marilyn A Walker, Alexander I Rudnicky, John S Aberdeen, et al. Darpa communicator evaluation: progress from 2000 to 2001. In *INTERSPEECH*. Citeseer, 2002.

[3] Lori Lamel, Sophie Rosset, J-L Gauvain, and Samir Bennacef. The limsi arise system for train travel information. In *Acoustics, Speech, and Signal Processing, 1999. Proceedings., 1999 IEEE International Conference on*, volume 1, pages 501–504. IEEE, 1999.

[4] Gokhan Tur and Renato De Mori. *Spoken language understanding: Systems for extracting semantic information from speech*. John Wiley & Sons, 2011.

[5] Emilio Sanchis, Alfonso Ortega, M Inés Torres, and Javier Ferreiros. Timpano: Technology for complex human-machine conversational interaction with dynamic learning. *Procesamiento del Lenguaje Natural*, 51:227–230, 2013.

[6] F Garcıa, LF Hurtado, E Segarra, et al. Combining multiple translation systems for spoken language understanding portability. 2012.

[7] José-Miguel Benedı, Eduardo Lleida, Amparo Varona, et al. Design and acquisition of a telephone spontaneous speech dialogue corpus in spanish: Dihana. In *Fifth International Conference on Language Resources and Evaluation (LREC)*, pages 1636–1639, 2006.

[8] Marcos Calvo, Fernando Garcıa, Lluıs-F Hurtado, et al. Exploiting multiple hypotheses for multilingual spoken language understanding. *CoNLL-2013*, pages 193–201, 2013.

[9] Stefan Hahn, Marco Dinarelli, Christian Raymond, et al. Comparing stochastic approaches to spoken language understanding in multiple languages. *Audio, Speech, and Language Processing, IEEE Transactions on*, 19(6):1569–1583, 2011.

[10] Fabrice Lefevre, François Mairesse, and Steve Young. Cross-lingual spoken language understanding from unaligned data using discriminative classification models and machine translation. In *INTERSPEECH*, pages 78–81, 2010.

[11] Marcos Calvo, Lluís-F Hurtado, Fernando García, and Emilio Sanchis. A multilingual slu system based on semantic decoding of graphs of words. In *Advances in Speech and Language Technologies for Iberian Languages*, pages 158–167. Springer, 2012.

[12] Mark A Larkin, Gordon Blackshields, NP Brown, et al. Clustal w and clustal x version 2.0. *Bioinformatics*, 23(21):2947–2948, 2007.

[13] Harald Höge. Project proposal TC-STAR-make speech to speech translation real. In *in Proc. of the LREC'02, Las*, 2002.

[14] Steve Young, Gunnar Evermann, Mark Gales, et al. *The HTK book*, volume 2. Entropic Cambridge Research Laboratory Cambridge, 1997. Versión digital en `http://htk.eng.cam.ac.uk/prot-docs/htk_book.shtml`.

[15] World Wide Web Consortium. Web Audio API. `http://www.w3.org/TR/webaudio/`, 2013. [29/07/14].

[16] Matt Diamond. Recorder.js. `https://github.com/mattdiamond/Recorderjs`, 2013. [29/07/14].

[17] Django Software Foundation. Django (versión 1.6). `https://djangoproject.com/`, 2013. [26/06/14].

[18] Jacob Kaplan-Moss and Adrian Holovaty. *The Definitive Guide to Django Web Development Done Right*. Apress, 2007. Verisón digital en `http://www.djangobook.com`.

[19] Benoit Chesneau. Gunicorn (versión 18). `http://gunicorn.org/`, 2013. [02/07/14].

[20] Python Software Foundation. Python Web Server Gateway Interface v1.0.1. `http://legacy.python.org/dev/peps/pep-3333/`, 2010. [03/07/14].

[21] Gunicorn Documentation. Deploying gunicorn. `http://docs.gunicorn.org/en/18.0/deploy.html`. [21/07/14].

[22] Nginx Inc. Nginx (versión 1.1). `http://nginx.org/`, 2012. [02/07/14].