UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC
Departamento de Sistemas Informáticos y Computación

Departament de Sistemes Informàtics i Computació

Universitat Politècnica de València

Detección automática de segmentos acústicos e inferencia de unidades lingüísticas

Trabajo Fin de Máster

Master en Inteligencia Artificial,
Reconocimiento de Formas e Imagen Digital

Autor: Sergio Laguna Bello

Tutor: Emilio Sanchis

Curso: 2015/16

Resumen

En este proyecto se estudian diferentes tareas relacionadas con el procesamiento de lenguaje natural. En concreto, el trabajo consiste en tareas basadas en lenguaje hablado en casos donde la cantidad de recursos es limitada o inexistente para el idioma a tratar. Los problemas que se abordan son "*Query-by-Example*", que consiste en encontrar en un repositorio de audio, segmentos de semejantes al *query*, que también es audio; o "*Spoken Term Discovery*" que consiste en identificar en un audio segmentos comunes, con o sin significado lingüístico. Esto supone un nuevo enfoque en este tipo de tareas, donde es habitual disponer de una gran cantidad de recursos en el idioma correspondiente. Las tareas presentadas en este trabajo forman parte de concursos de evaluación, como el *MediaEval BenchMark*.

Palabras clave: identificación unidades acústicas, búsqueda términos, programación dinámica

Abstract

In this project different tasks related to natural language processing are studied. Specifically, the work is based on spoken language tasks where the amount of resources is limited or inexistent for the language. These tasks are "Query-by-Example", which consists in finding within an audio repository similar segments to the query; and "Spoken Term Discovery" which consists in identifying common audio segments, with or without linguistic meaning. This is a new approach in this kind of tasks, where it is usual to use a great amount of data in the appropiate language. The tasks presented in this work are part of some competitions, like MediaEval BenchMark.

Keywords: acoustic units identification, term search, dynamic programming

Índice general

Capítulo 1

Introducción

1.1. Procesamiento del habla

Dentro del área del procesamiento del habla se encuentran diversas tareas cuyo objetivo es poder extraer información a partir de un discurso hablado. Una de las principales tareas dentro de este ámbito es el reconocimiento automático del habla.

El reconocimiento automático del habla tiene como objetivo obtener una transcripción en texto del discurso de un locutor. Para ello, un sistema de reconocimiento utiliza modelos acústicos y del lenguaje. Estos modelos son, habitualmente, de tipo estadístico y se obtienen a partir de un corpus.

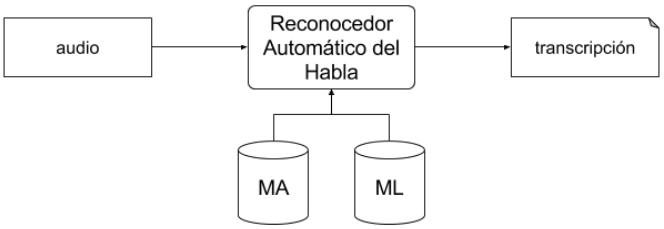

Figura 1.1: Sistema de reconocimiento automático del habla.

Además del reconocimiento del habla, otras tareas incluidas en este ámbito también son, por ejemplo, la comprensión del habla o el reconocimiento del locutor o del idioma utilizado.

Para poder realizar todas estas tareas es necesario disponer de una gran cantidad de recursos, con los que se pueda entrenar un sistema. En el caso de un reconocedor automático del habla, son necesarios numerosos fragmen-

tos de audio con la correspondiente transcripción para poder entrenar los modelos acústicos y del lenguaje.

Esta cantidad de recursos es un problema, ya que obtenerlos es costoso en tiempo y dinero (por ejemplo, obtener la transcripción de las frases). Además, existen muchos idiomas con pocos hablantes, lo que hace que obtener estos recursos sea aún más difícil.

Por estos motivos, se ha planteado en los últimos años qué podemos hacer si los recursos disponibles son escasos o, incluso, nulos.

1.2. Procesamiento con recursos limitados

Como hemos visto, es importante disponer de recursos suficientes en determinadas tareas. Sin embargo, esta situación no siempre es posible. Por ello surgen nuevas tareas capaces de procesar el lenguaje natural sin estos recursos. Para poder alcanzar este objetivo, es habitual utilizar una de las siguientes estrategias: *zero resources* o *low resources*.

- *Zero resources*
  En este caso no disponemos de ningún tipo de recurso anotado para entrenar nuestro sistema. En audio, por ejemplo, podríamos utilizar características acústicas como los coeficientes cepstrales, ya que se obtienen automáticamente.

- *Low resources*
  En este tipo de estrategia sí tenemos a nuestro alcance recursos anotados, pero no del idioma a procesar. El procedimiento es similar a tener los recursos del propio idioma, con el inconveniente de que, evidentemente, no se ajustan a los datos a tratar.

Aplicando alguna de estas dos estrategias podemos afrontar tareas en las que no es estrictamente necesario disponer de recursos anotados del idioma a procesar. Entre otras tareas, podemos encontrar dentro de este tipo que no necesitan recursos propios tareas como *Spoken Term Discovery* (descubrimiento de términos) o *Query by Example* (búsqueda mediante ejemplo). Son, en concreto, estas dos tareas las que desarrollamos en el siguiente trabajo.

1.3. Objetivos

El objetivo principal de este trabajo es la construcción de sistemas de procesamiento de lenguaje natural que permitan extraer información del discurso hablado sin recursos propios del idioma a tratar.

Más concretamente, este objetivo se divide en los dos siguientes:

- Desarrollar un sistema que a partir de un conjunto de documentos de audio sea capaz de descubrir términos o palabras.
- Construir un sistema de búsqueda de audios a partir de consultas realizadas por voz.

Para ello, la memoria se estructura de la siguiente forma:

- En el capítulo 2 se presenta la primera tarea, *Spoken Term Discovery*, explicando en que consiste e incluyendo el estado del arte.
- En el capítulo 3 se detalla la aproximación utilizada y la experimentación realizada para la tarea de *Spoken Term Discovery*, *Zero Resource Speech Challenge 2015*.
- En el capítulo 4 se presenta una nueva tarea, *Query-by-Example*, definiéndola y detallando el estado del arte de esta tares.
- En el capítulo 5, de forma similar a la tarea anterior, se detalla la participación en la tarea de *Query-by-Example*, *QUEEST*, del MediaEval 2015.
- Para finalizar, en el capítulo 6 se extraen las conclusiones del trabajo y se enumeran posibles trabajos futuros.

Además, en el anexo A se incluyen las publicaciones relacionadas con el presente trabajo y se explican brevemente otros trabajos en desarrollo.

Capítulo 2

Spoken Term Discovery

En este capítulo vamos a presentar la primera de la tareas que abordamos en el presente trabajo. Esta tarea consiste en el descubrimiento de términos en el lenguaje hablado. En inglés, se conoce esta tarea como *Spoken Term Discovery*. Primero, explicaremos brevemente en que consiste este tipo de tarea. Después, entraremos a detallar el estado del arte en este ámbito.

2.1. Definición de la tarea

El objetivo de la tarea *Spoken Term Discovery* es, como el propio nombre indica, descubrir términos a partir de un conjunto de fragmentos de audio. La dificultad de esta tarea es que el descubrimiento debe realizarse únicamente a partir de los audios, sin disponer de otros datos como transcripciones de los mismos audios. Por ello, estamos ante una tarea en la que debemos aplicar técnicas no supervisadas.

Esta tarea nos permite obtener información de un conjunto de audios sin antes haber aprendido un modelo de datos, como en otras tareas de procesamiento del lenguaje, como puede ser un reconocedor automático del habla. De esta forma, en escenarios donde no disponemos de suficientes datos como para construir los modelos, podemos ser capaces de extraer información, ya sea porque obtener los datos es conlleva un gasto inasumible o porque se trata de un idioma poco hablado y no existen suficientes datos.

Entre otras aplicaciones, el descubrimiento de términos nos permite realizar nuevas tareas:

- *Clustering* de documentos
  A partir de los términos descubiertos, podemos dividir los documentos en diferentes *clusters*, según su similitud entre estos términos [1].

- Resumen automático de documentos
  Con los términos más frecuentes, se pueden detectar qué fragmentos de un discurso son más importantes [2].

2.2. Estado del arte

En los últimos años, ha habido una aproximación para esta tarea que se ha extendido y que sirve como base para muchas de las implementaciones actuales. En 2005, Park y Glass presentan un algoritmo no supervisado que permite descubrir patrones acústicos a partir de repeticiones en diferentes audios [3].

El algoritmo que presentan es una variación de un conocido algoritmo de programación dinámica, *Dynamic Time Warping*. Este algoritmo obtiene la alineación óptima entre dos secuencias.

La fórmula recursiva del algoritmo DTW se puede observar en la ecuación 2.1.

$$M(i,j) = \begin{cases} 0 & i = j = 0 \\ +\infty & i = 0, j > 0 \\ +\infty & j = 0, i > 0 \\ \min\limits_{\forall (x,y) \in S} M(i-x, j-y) + D(A_i, B_j) & i > 0 \end{cases} \tag{2.1}$$

donde M es la matriz de programación dinámica; S es el conjunto de movimientos permitidos, representados como pares (x, y) de incrementos horizontales y verticales; A_i, B_j son los objetos que representan las posiciones i-ésima y j-ésima de sus respectivas secuencias; y D es la función que calcula la distancia o la disimilitud entre dos objetos.

El coste temporal y espacial de este algoritmo en su versión iterativa es proporcional a la longitud de los dos audios a comparar, por lo tanto el coste es $\mathcal{O}(nm)$, siendo n y m la longitud de los audios.

En nuestro caso, las secuencias a alinear son los audios, sin embargo, no se pueden comparar directamente, ya que primero es necesario convertir los audios en una representación vectorial temporal en términos de características acústicas. A partir de esta representación se puede obtener una matriz de distancias, en la que el valor $D_{i,j}$ se corresponde con la distancia entre los vectores i y j. Finalmente, con la matriz de distancias previamente calculada, podemos aplicar el algoritmo DTW y obtener el mejor alineamiento entre ambos audios.

Sin embargo, en la tarea que nos ocupa, lo habitual es disponer de audios compuestos por múltiples palabras, lo que impide aplicar el algoritmo DTW tal cual lo hemos visto. Por ello, el objetivo de la variación propuesta por Park y Glass es ser capaces que el algoritmo recoja alineamientos locales o alineamientos de subsecuencias.

Park y Glass llaman a su variación *Segmental DTW* que se basa en buscar en múltiples caminos de la matriz de distancias y poder encontrar alineamientos locales. El planteamiento es el siguiente:

1. Se divide la matriz de distancias en bandas diagonales solapadas con una anchura W.

2. Se aplica el algoritmo DTW a cada una de estas bandas para encontrar el mejor alineamiento.

3. Se acota el mejor alineamiento de cada banda obteniendo la subsecuencia de menor media cuya longitud sea, al menos, L.

4. Finalmente, nos quedamos con la subsecuencia de menor distancia como el mejor alineamiento entre ambos audios.

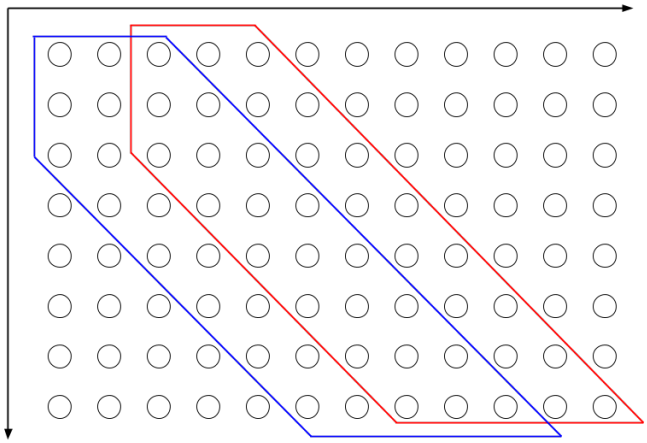

Figura 2.1: Bandas del *Segmental DTW* ($W = 2$).

A partir de esta descripción podemos observar varios inconvenientes:

- El coste computacional de este algoritmo es drásticamente mayor al coste del *Dynamic Time Warping*. Si utilizamos el algoritmo presentado

por Lin et ál. para encontrar la subsecuencia de menor media con la restricción de que esta subsecuencia debe tener una longitud mínima (L), el coste es $\mathcal{O}(n \log(L))$ [4]. Por tanto, el coste del DTW más la obtención de las subsecuencias de cada banda sería $\mathcal{O}(nm \log(L))$.

- Tenemos dos parámetros (W y L) que será necesario ajustar para obtener los mejores resultados posibles.

A pesar de estos puntos, y como hemos comentado anteriormente, gran parte de los sistemas posteriores se basan en este algoritmo.

En cuanto a la parametrización del audio que nos permita obtener la matriz de distancias en la que aplicar el algoritmo *Segmental DTW*, Zhang y Glass [5] proponen la utilización de una representación basada en posteriorgramas Gaussianos. En este caso, un posteriorgrama Gaussiano es un vector de probabilidades, representando las probabilidades a posteriori de un modelo de mixturas de gaussianas para un *frame* de audio.

Una de las ventajas que conlleva este método es la posibilidad de generalizar los datos para que sean independientes del locutor. En el articulo anteriormente citado, la aproximación planteada se divide en tres pasos:

1. A partir de los MFCCs de los audios, se aprende un modelo de mixturas de gaussianas y se obtienen los vectores de posteriorgrmas.

2. Se aplica el algoritmo de *Segmental DTW* para encontrar similitudes.

3. Las secuencias encontradas pasan un *clustering* que las agrupa.

Muscariello et ál. presentan una variación del algoritmo *Segmental DTW* que denominan *Segmental Locally Normalized DTW* [6]. Con esta variación, se valora la distancia del punto de inicio del *DTW* y la longitud de los caminos durante la minimización. El algoritmo se divide en tres pasos:

1. Identificar los posibles puntos de inicio donde puedan comenzar los patrones similares en ambos audios a partir la distancia inicial.

2. Calcular el alineamiento óptimo de los caminos que comienzan en estos puntos de inicio teniendo en cuenta la distancia normalizada por la longitud del camino.

3. Asignar un *score* como medida de la similitud de los dos fragmentos correspondientes al alineamiento obtenido anteriormente.

Para finalizar, Jansen y Van Durme [7] introducen la posibilidad de hacer más eficiente, tanto en tiempo como en memoria, la búsqueda de los patrones. Para ello, utilizan tres algoritmos que conjuntamente son capaces de obtener una matriz de similitud en tiempo $\mathcal{O}(n \log n)$. Los tres algoritmos son:

1. ***Locality Sensitive Hashing***
   El objetivo de esta fase es proyectar la representación vectorial de los audios a unos vectores aleatorios pasando de un espacio $\mathbb{R}^d$ a otro $\{0,1\}^b$. Tras esta reducción de dimensionalidad, se puede aproximar la similitud coseno a partir de la distancia de *Hamming*. Se puede implementar con una multiplicación matricial.

2. ***Point Location in Equal Balls***
   En esta fase se calcula la matriz de similitud aproximada a partir de los vectores obtenidos en el paso anterior. Este paso concentra el coste computacional del procedimiento planteado, ya que tiene un coste $\mathcal{O}(n \log n)$.

3. ***Two-Pass Repeated Trajectory Search***
   En este último paso se buscan los patrones en la matriz anteriormente obtenida. Se utiliza un algoritmo de dos pasos donde busca lineas diagonales en las que, posteriormente, aplica un simple DTW.

Capítulo 3

The Zero Resource Speech Challenge 2015

The Zero Resource Speech Challange 2015 [8] es un desafío cuyo objetivo es "el descubrimiento de unidades lingüísticas de forma no supervisada en el discurso hablado sin conocimiento previo del idioma". [1]

Este desafío cubre dos niveles de estructura lingüística: unidades de subpalabras y unidades de palabras. Esta división según el nivel da lugar a dos tareas:

- ***Unsupervised subword modeling***
  En esta tarea, el objetivo es conseguir una representación de sonidos robusta a diferentes hablantes que facilite la identificación de palabras.

- ***Spoken Term Discovery***
  En esta tarea, el objetivo es el descubrimiento de palabras de forma no supervisada, entendiendo esto como patrones de audio repetidos con cierta frecuencia en una colección de documentos de audio.

Como se indica en el mismo objetivo del desafío, se plantea que los sistemas desarrollados para las dos tareas no dispongan de información previa de los idiomas utilizados (*zero resource*). Sin embargo, también se acepta una aproximación de *low resource*, mediante la cual se puede utilizar información de otros idiomas.

En este capítulo vamos a ver como hemos afrontado la segunda tarea, que como indica el propio nombre, se trata de una tarea de *Spoken Term Discovery*, explicada en el capítulo 2.

[1] `http://www.lscp.net/persons/dupoux/bootphon/zerospeech2014/website/index.html`

3.1. Descripción de la tarea

Como hemos explicado brevemente, el objetivo de esta tarea es descubrir unidades lingüísticas, habitualmente palabras, a partir de un conjunto de documentos de audio. Dada la naturaleza de la tarea, no disponemos de transcripciones de los audios, ya que el objetivo es realizar el descubrimiento solo a partir de los audios proporcionados.

Por tanto, debemos obtener una lista de fragmentos donde se repitan determinados patrones (a ser posible, palabras), de forma que aquellos patrones similares se incluyan en un único grupo.

Primero, vamos a analizar los datos proporcionados. Para esta tarea disponemos de dos conjuntos de datos, cada uno de ellos formado por documentos de audio de diferentes idiomas:

- El conjunto de inglés proviene del corpus Buckeye [9]. Contiene fragmentos de audio de doce hablantes diferentes. Se han seleccionado por cada hablante entre 16 y 30 minutos de audio, haciendo un total de casi cinco horas.

- El conjunto de xitsonga (idioma hablado en varios países del sur de África) proviene del corpus NCHLT Speech Corpus [10]. Consiste en fragmentos de 24 hablantes, seleccionando entre 2 y 29 minutos de audio por cada uno de ellos, con un total de casi dos horas y media de audio.

Para cada uno de los corpus se dispone también de información indicando los fragmentos a analizar, considerándose el resto del audio como *non-speech*.

	Inglés	Xitsonga
Longitud audios	5 h	2 h 30 min
Fragmentos	14 137	4 058
# hablantes	12	24
Audio/hablante	16-30 min	2-29 min
Tipo audio	Casual	Leído

Tabla 3.1: Información sobre el corpus *ZeroSpeech* 2015.

A continuación, vamos a ver con detalle las métricas utilizadas en la presente tarea para analizar nuestros sistemas de *Spoken Term Discovery*.

3.1.1. Métricas

En esta tarea las métricas utilizadas tienen gran importancia. En tareas similares planteadas con anterioridad se han utilizado diferentes métricas para evaluar los sistemas propuestos. Dado que ninguna de estas métricas es capaz de medir por si misma la calidad de un sistema, en esta tarea se presentan diversas métricas. Todas ellas se engloban en tres aspectos diferentes: *matching*, *clustering* y *parsing*.

Esta división en tres tipos de métricas viene dado a partir de un posible sistema de *Spoken Term Discovery*. Primero se buscarían pares de fragmentos de audio basándose en su similitud (*matching*). Después, a partir de esos pares podemos clasificar los fragmentos, construyendo algo equivalente a un lexicón (*clustering*). Para finalizar, podemos utilizar estas clases para clasificar los posibles *tokens*.

En los siguientes apartados vamos a ver que intentan medir cada tipo y las diferentes métricas que componen cada uno de ellos. Para ello, primero se definen una serie de conjuntos. Son los siguientes:

- Primero se define el conjunto C_{disc}, que contiene todos los *clusters* descubiertos.
- F_{all} es el conjunto de fragmentos existentes en el corpus. Estos fragmentos están formados entre tres y veinte unidades fonéticas.

$$F_{all} = \{\langle i, j\rangle \in \mathbb{N} \times \mathbb{N} \mid 1 \leq i \leq j \leq n, 3 \leq j - i + 1 \leq 20\}$$

- P_{all} es el conjunto de pares de fragmentos cuya transcripción fonética es igual.

$$P_{all} = \{\langle\langle i, j\rangle, \langle k, l\rangle\rangle \in F_{all} \times F_{all} \mid T_{i,j} = T_{k,l}, [i, j] \cap [k, l] = \emptyset\}$$

- El conjunto $P_{goldclus}$ está formado por los pares de P_{all} que encuentra el sistema, aunque estén en diferentes *clusters*.

$$\begin{aligned} P_{goldclus} = \{&\langle\langle i, j\rangle, \langle k, l\rangle\rangle \in F_{all} \times F_{all} \mid \\ &\exists c_1, c_2 \in C_{disc}, \langle i, j\rangle \in c_1 \wedge \langle k, l\rangle \in c_2, \\ &T_{i,j} = T_{k,l}, [i, j] \cap [k, l] = \emptyset\} \end{aligned}$$

- Por otra parte, P_{clus} estará compuesto por los pares que **sí** estén en el mismo *cluster*.

$$P_{clus} = \{\langle\langle i, j\rangle, \langle k, l\rangle\rangle \mid \exists c \in C_{disc}, \langle i, j\rangle \in c \wedge \langle k, l\rangle \in c\}$$

- $F_{goldLex}$ es el conjunto de fragmentos correspondientes a los audios transcritos a nivel de palabra.
- B_{gold} es el conjunto de límites del corpus.
- A partir de los resultados se derivan el conjunto de fragmentos descubiertos (F_{disc}), los pares de fragmentos descubiertos (P_{disc}) y los límites de los fragmentos descubiertos (B_{disc}).

$$\begin{aligned} F_{disc} &= \{f \mid f \in c, c \in C_{disc}\} \\ P_{disc} &= \{\langle f_1, f_2\rangle \mid f_1 \neq f_2 \in c, c \in C_{disc}\} \\ B_{disc} &= \{i \mid \exists j : \langle i, j\rangle \in F_{disc} \wedge \langle j, i\rangle \in F_{disc}\} \end{aligned}$$

- Por último, el conjunto $P_{disc}*$ contiene todos los posibles pares de subsecuencias del conjunto P_{disc}.

Además, también se definen las siguientes funciones auxiliares:

- La función $ned(\langle i, j\rangle, \langle k, l\rangle)$ calcula, utilizando la distancia de Levenshtein, la diferencia entre dos fragmentos de audio.

$$ned(\langle i, j\rangle, \langle k, l\rangle) = \frac{Levenshtein(T_{i,j}, T_{k,l})}{max(j - i + 1, k - l + 1)}$$

- La función $flat(P)$ devuelve todos los fragmentos contenidos en el conjunto P.

$$flat(P) = \{p \mid \exists q : \langle p, q\rangle \in P \vee \langle q, p\rangle \in P\}$$

- $cover(P)$ devuelve la parte de los audios que está cubierta por el conjunto de fragmentos P.

$$cover(P) = \bigcup_{\langle i,j\rangle \in flat(P)} [i, j]$$

- La función $occ(t, P)$ devuelve los fragmentos del conjunto P cuya transcripción fonética sea t.

$$occ(t, P) = \{\langle i, j\rangle \in flat(P) \mid T_{i,j} = t\}$$

- Utilizando funciones previas, $w(t, P)$ calcula el ratio de fragmentos del conjunto P cuya transcripción sea t.

$$w(t, P) = \frac{|occ(t, P)|}{|flat(P)|}$$

- Por último, $types(F)$ obtiene las diferentes transcripciones fonéticas de los fragmentos de audio del conjunto F.

$$types(F) = \{T_{i,j} \mid \langle i, j\rangle \in flat(F)\}$$

Utilizando los anteriores conjuntos y funciones, se definen las métricas que a continuación se detallan.

Matching

El objetivo de las métricas de *matching* es evaluar la calidad del proceso en el que los fragmentos de audio se alinean y comparan, etapa habitual en los sistemas de *Spoken Term Discovery*. Las métricas que se encargan de evaluar este aspecto son tres: distancia de edición normalizada (*Normalized Edit Distance, NED*), cobertura (*coverage*) y *matching*.

La distancia de edición normalizada utiliza la distancia de Levenshtein para comparar el conjunto de pares de fragmentos de audio que devuelve el sistema. El objetivo es que los pares de fragmentos sean similares, y por tanto, el valor de esta métrica sea el menor posible.

$$\text{NED} = \sum_{\langle x,y\rangle \in P_{disc}} \frac{ned(x,y)}{|P_{disc}|} \tag{3.1}$$

Coverage indica el ratio entre la longitud que cubre el conjunto de pares de fragmentos devueltos y la longitud del conjunto que deberíamos encontrar. Debemos tener en cuenta que esta métrica no tiene en cuenta si ambos conjuntos son similares, únicamente compara sus dimensiones.

$$\text{coverage} = \frac{|cover(P_{disc})|}{|cover(P_{all})|} \tag{3.2}$$

La última métrica de este tipo, *matching*, compara los pares de fragmentos devueltos (incluyendo posibles subfragmentos) con el conjunto real. Esta métrica, igual que las próximas, se calcula en términos de *precision* y *recall*.

$$\text{matching}_{\text{P}} = \sum_{t \in types(P_{disc^*})} w(t, P_{disc^*}) \frac{|occ(t, P_{disc^*} \cap P_{all})|}{|occ(t, P_{disc^*})|} \tag{3.3}$$

$$\text{matching}_{\text{R}} = \sum_{t \in types(P_{all})} w(t, P_{all}) \frac{|occ(t, P_{disc^*} \cap P_{all})|}{|occ(t, P_{all})|} \tag{3.4}$$

Clustering

El segundo tipo de métricas evalúan la calidad de los *clusters* devueltos, es decir, aquellos fragmentos cuya similitud es tan alta como para deducir que representan el mismo patrón o, en este caso, la misma palabra. En este tipo de métricas se encuentran *grouping* y *type*

La primera métrica, *grouping*, comprueba si los fragmentos que devuelve el sistema se agrupan correctamente. Compara los fragmentos que forman un mismo *cluster* con aquellos cuya transcripción fonética es igual, aunque el sistema no los ubique en un mismo *cluster*.

$$\text{grouping}_{\text{P}} = \sum_{t \in types(P_{clus})} w(t, P_{clus}) \frac{|occ(t, P_{clus} \cap P_{goldclus})|}{|occ(t, P_{clus})|} \tag{3.5}$$

$$\text{grouping}_{\text{R}} = \sum_{t \in types(P_{goldclus})} w(t, P_{goldclus}) \frac{|occ(t, P_{clus} \cap P_{goldclus})|}{|occ(t, P_{goldclus})|} \tag{3.6}$$

En el caso de *type*, el objetivo es comprobar si el sistema detecta los patrones adecuados, comparando las transcripciones fonéticas de los fragmentos devueltos con las transcripciones de los fragmentos que el sistema debería encontrar.

$$\text{type}_{\text{P}} = \frac{|types(F_{disc}) \cap types(F_{goldLex})|}{|types(F_{disc})|} \tag{3.7}$$

$$\text{type}_{\text{R}} = \frac{|types(F_{disc}) \cap types(F_{goldLex})|}{|types(F_{goldLex})|} \tag{3.8}$$

Parsing

En este último tipo de métricas nos centramos en los fragmentos de audio. Para evaluarlos utilizamos dos métricas: *token* y *boundary*.

En el caso de *token* el objetivo es comprobar que los fragmentos se corresponden con palabras y no son estructuras lingüísticas de menor o mayor nivel.

$$\text{token}_{\text{P}} = \frac{|F_{disc} \cap F_{goldLex}|}{|F_{disc}|} \tag{3.9}$$

$$\text{token}_{\text{R}} = \frac{|F_{disc} \cap F_{goldLex}|}{|F_{goldLex}|} \tag{3.10}$$

Por último, *boundary* trata de evaluar la adecuación de los límites de inicio y fin de los fragmentos, es decir, compara si estos límites son los reales o los fragmentos empiezan o terminan en otros puntos.

$$\text{boundary}_\text{P} = \frac{|B_{disc} \cap B_{gold}|}{|B_{disc}|} \tag{3.11}$$

$$\text{boundary}_\text{R} = \frac{|B_{disc} \cap B_{gold}|}{|B_{gold}|} \tag{3.12}$$

Para todas las métricas anteriores donde se calcula la *precision* y el *recall* también se calcula el *F-score*, que se trata de la media armónica entre *precision* y *recall*.

$$\text{F-score} = \frac{2}{1/P + 1/R} \tag{3.13}$$

Toda la información sobre las métricas se puede ver en el artículo de la tarea [8].

3.2. Descripción del sistema

Los sistemas que planteamos siguen una estrategia de *low resource*, donde utilizamos modelos de otros idiomas. Este tipo de aproximaciones suelen ofrecer mejores resultados que *zero-resources*, como se ha visto en el apartado de estado del arte, a cambio de necesitar estos modelos de idiomas.

Las dos aproximaciones analizadas se basan en obtener a partir de los audios diferentes características. En ambos casos se utiliza el reconocedor fonético de la universidad de Brno, *PhnRec* [11], basado en redes neuronales, que nos permite obtener tanto una decodificación fonética como la probabilidad de los diferentes fonemas a lo largo del tiempo (posteriorgramas).

Para obtener esta decodificación necesitamos un sistema entrenado para un idioma en concreto. Este reconocedor ya incluye cuatro sistemas entrenados para los idiomas inglés, checo, húngaro y ruso. Los datos de entrenamiento de estos sistemas son el corpus TIMIT para el inglés [12] y el corpus SpeechDat-E en sus variantes de checo, húngaro y ruso [13].

Internamente, el reconocedor utiliza una arquitectura de Modelos Ocultos de Markov (HMM) para representar cada unidad fonética. De esta forma, la salida del reconocedor es la probabilidad a posteriori de cada uno de los tres estados de cada unidad fonética en cada *frame* del audio. En cada conjunto de unidades, existen tres adicionales que no representan fonemas reales, sino ruido o silencio.

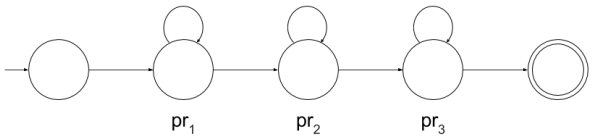

Figura 3.1: Modelo oculto de Markov por cada unidad fonética.

De esta forma, obtenemos cada 10 milisegundos un vector de probabilidades cuyo tamaño es tres veces el número de unidades fonéticas (las tres probabilidades a posteriori de cada unidad fonética). El número de unidades fonéticas de los sistemas ya incluidos se puede ver en la tabla 3.2.

Idioma	# unidades
Inglés	39
Checo	45
Húngaro	61
Ruso	52

Tabla 3.2: Número de unidades fonéticas de los modelos de *PhnRec*.

A partir de las dos posibles salidas que nos ofrece el reconocedor, planteamos dos aproximaciones. En cualquier caso, la entrada al reconocedor consiste en los audios al completo. La información proporcionada sobre los fragmentos que debemos analizar se utiliza después del reconocimiento, "cortando" la salida del reconocedor para quedarnos únicamente con los fragmentos a analizar. Realizando tras la decodifcación, el reconocedor fonético debería proporcionar unos resultados mejores (sobretodo en los *frames* iniciales y finales del fragmento) que si se cortasen los audios antes de pasar por esta etapa, ya que dispone del contexto acústico.

3.2.1. Similitud de posteriorgramas

La base de esta aproximación es obtener las probabilidades de los fonemas en función del tiempo y buscar fragmentos de audio similares utilizando estas probabilidades como características de los audios. Como se ha visto anteriormente, obtenemos un vector de probabilidades por cada *frame* de 10 milisegundos. Cada vector, al tratarse de probabilidades, debe sumar 1.

A partir de esta representación, utilizamos un algoritmo de alineamiento basado en *Dynamic Time Warping* para detectar los fragmentos que contengan términos similares. Para ello, primero obtendremos la matriz de distancias en la que representaremos la diferencia entre los diferentes *frames* de

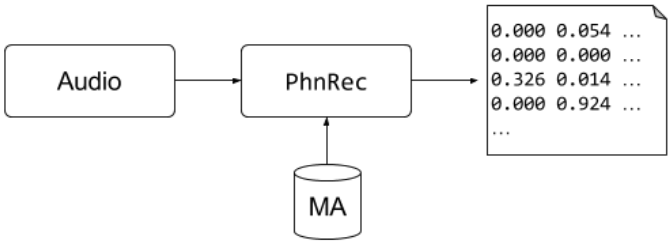

Figura 3.2: Obtención de posteriorgramas.

dos audios. En la figura 3.3 se puede observar un ejemplo de una matriz de distancias, donde el objetivo es encontrar regiones con valores de distancias bajas como las resaltadas dentro de los recuadros rojos.

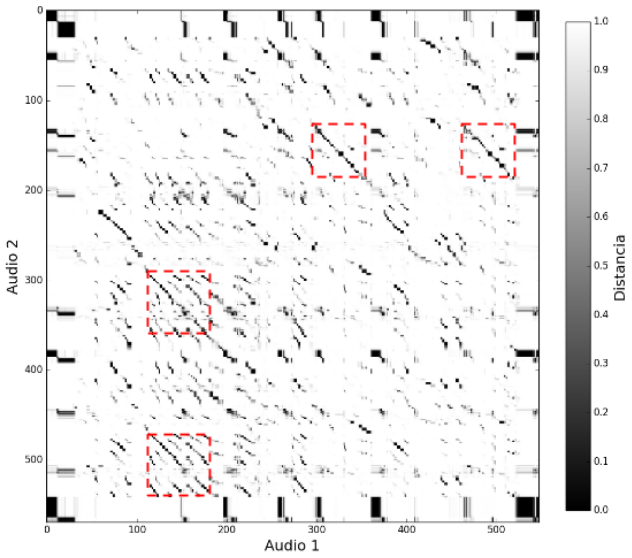

Figura 3.3: Ejemplo de comparación por posteriorgramas.

El objetivo de esta tarea es encontrar secuencias de *frames* consecutivas en un par de audios cuyas distancias sean bajas. Por ello, utilizar *Dynamic Time Warping* para encontrar la mejor alineación es una buena idea. Sin embargo, estas secuencias son solo parte de los audios, ya que representarán un término y no la frase completa de cada audio. Para encontrar estas subsecuencias, planteamos el algoritmo 3.4, donde $D(i, j)$ es la distancia entre los *frames* i

y j de los correspondientes audios y $frames_k$ hace referencia al número de *frames* de cada audio.

for $i = 0$ **to** $frames_1$ **do**
  for $j = 0$ **to** $frames_2$ **do**
    if $D(i, j) \leq \theta$ **then**
      DTW desde el punto (i,j)
    end if
  end for
end for

Figura 3.4: Búsqueda de subsecuencias

Con este algoritmo, solo comenzamos a explorar un posible camino en puntos donde la diferencia entre *frames* sea lo suficientemente baja (menor que el umbral θ). A partir de estos puntos realizamos un *Dynamic Time Warping*, pero, en este caso, si la distancia media acumulada supera el umbral anterior, entendemos que la secuencia similar ha terminado y, por lo tanto, interrumpimos el *Dynamic Time Warping*.

El resultado de este algoritmo es un conjunto de secuencias cuya distancia media no supera un determinado umbral. Dado que estamos buscando secuencias que deberían contener una palabra, la longitud de estas secuencias debería tener un valor mínimo, ya que es posible que determinadas secuencias sean muy cortas (por ejemplo, por coincidir en un único fonema). Por ello, al listado de secuencias aplicamos un filtrado, descartando todas aquellas que no superen una determinada longitud L.

Esta aproximación tiene diversos inconvenientes, como la necesidad de establecer diferentes umbrales, como por ejemplo cuál será la distancia máxima permitida entre dos fragmentos para que se considere que contienen el mismo término. Igualmente, si se establece este umbral, es posible que los fragmentos tiendan a ser de longitud reducida, lo que lleva a tener que establecer una longitud mínima para que los fragmentos devueltos sean lo suficientemente largos como para contener una palabra y no solo parte de ella.

3.2.2. Comparación de fonemas

La base de esta aproximación es la obtención de la decodificación en fonemas de los audios y buscar subsecuencias de estos fonemas entre todos los audios que forman el corpus. En este caso, partimos de una serie de fragmentos de los audios a los que el reconocedor fonético les asigna el fonema más probable, de forma similar al ejemplo de la figura 3.5.

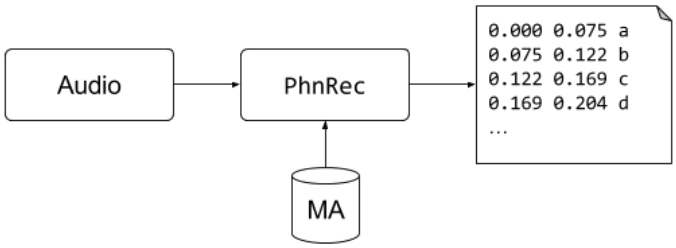

Figura 3.5: Obtención de la decodificación fonética.

Con la decodificación fonética, se buscan cadenas de fonemas iguales entre todos los pares de audios. Por ejemplo, siguiendo dos posibles decodificaciones mostradas en la figura 3.6, ambos audios tienen un fragmento cuya decodificación fonética es igual.

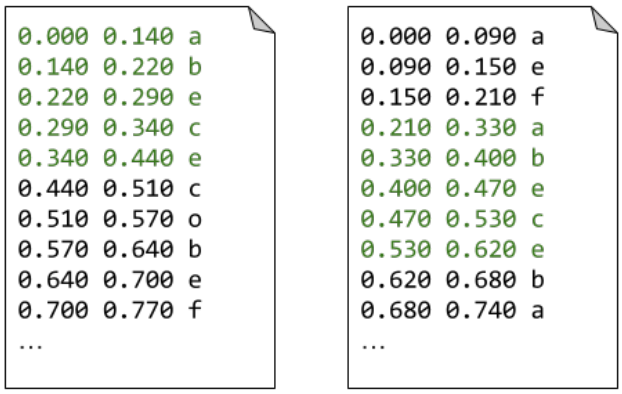

Figura 3.6: Ejemplo de secuencias fonéticas similares.

Esta búsqueda de secuencias fonéticas se puede realizar aplicando el algoritmo de *Longest common substring*, para encontrar la subcadena común más larga entre dos decodificaciones fonéticas. Utilizando una implementación que utilice la programación dinámica, el coste es $\mathcal{O}(nm)$, siendo n y m la longitud de las dos secuencias de fonemas.

Para realizar esta búsqueda de secuencias fonéticas similares, también podemos utilizar el *software Lucene* [14]. Esta librería contiene un amplio conjunto de técnicas para la búsqueda en texto. Utilizando *Lucene* no solo vamos a ser capaces de realizar búsquedas para encontrar las secuencias fonéticas exactas, sino que también nos permite realizar búsquedas de for-

ma que encontremos secuencias similares. Este segundo tipo de búsquedas es conocido en *Lucene* como *Proximity searches*.

Las *Proximity searches* permiten obtener cadenas de palabras cuyo *slop* sea no superior a un valor n. El *slop* es una especie de distancia de edición calculada a partir de las posiciones relativas de las palabras que forman la cadena de búsqueda [2].

Este valor se calcula a partir de las inserciones o intercambios en las cadenas. Si, por ejemplo, la diferencia entre dos cadenas es una palabra el *slop* es igual a 1. En el caso que la diferencia consista en un intercambio de posición de dos palabras adyacentes, el *slop* será 2.

El problema de esta aproximación es que con este tipo de búsquedas obtenemos el audio donde se encuentra una coincidencia y no las marcas temporales dentro del audio donde se detecta esta coincidencia. Por ello, el problema se convierte en un *Query by Example*, donde tenemos que encontrar un patrón a partir de una muestra. Para resolver esta cuestión, aplicaremos el algoritmo de *Subsequence Dynamic Time Warping*, similar al DTW, pero con la posibilidad de encontrar un alineamiento que comience en cualquier punto del segundo audio. Esta variante del DTW se llevará a cabo utilizando los posteriorgramas obtenidos para la primera aproximación. Este método se explicará con mayor detalle en el apartado de *Query by example*.

3.3. Experimentación y resultados

En este apartado, vamos a explicar el proceso de experimentación llevado a cabo en la tarea y, posteriormente, veremos los resultados alcanzados y los compararemos con el resto de participantes.

Antes de presentar los experimentos realizados y los resultados obtenidos, debemos indicar que el evaluador proporcionado divide el conjunto de datos en diferentes partes para facilitar el cálculo de las métricas. Además, realiza dos tipos de particiones: uno donde cada parte contiene los datos del mismo locutor y otro donde se reparten los audios de cada locutor en todas las particiones. Por ello, el resultado que se indica aquí es la media de estos valores con las particiones con los audios de cada locutor repartidos.

3.3.1. Experimentación

A continuación vamos a ver los diferentes experimentos llevados a cabo siguiendo las aproximaciones planteadas anteriormente.

[2] `http://lucene.apache.org/core/6_2_0/core/org/apache/lucene/search/PhraseQuery.html#getSlop--`

Búsqueda completa por posteriorgramas

La primera experimentación se basa en la primera aproximación presentada en el apartado anterior: utilizar los posteriorgramas obtenidos con el reconocedor fonético para pasar por un sistema de búsqueda que utiliza un algoritmo de programación dinámica como es el *Dynamic Time Warping*. De esta manera, para cada par de fragmentos de audio se busca una posible secuencia similar en ambos.

El problema principal de esta tarea está en el número de fragmentos a analizar. Como queda reflejado en la tabla 3.1, el número de fragmentos en inglés es de 14 137 y en xitsonga es un total de 4 058. Esto hace que al número de pares de fragmentos que deberíamos analizar es de casi cien millones en el caso del inglés (99 920 316 pares) y casi diez millones en el conjunto de audios en xitsonga (8 231 653 pares). Por otra parte, también es cierto que los fragmentos de audio con los que trabajamos tienen una duración reducida, por lo que, aunque se trate de una cantidad importante de fragmentos, será determinante el coste computacional de nuestro algoritmo de búsqueda.

Dado que en esta tarea el objetivo es encontrar subsecuencias similares entre diferentes fragmentos, no podemos aplicar de forma directa el algoritmo de *Dynamic Time Warping*, ya que este busca el mejor alineamiento entre dos secuencias completas. Esto hace que tengamos que modificar el algoritmo de búsqueda a cambio de un mayor coste computacional, que en el caso del DTW es $\mathcal{O}(nm)$, siendo n y m la longitud en *frames* de los dos audios. Tras unas pequeñas pruebas con un subconjunto de audios, comprobamos como el coste temporal es excesivo, por lo que decidimos explorar otras aproximaciones.

Búsqueda completa por fonemas

Intentando evitar un coste computacional elevado, planteamos esta segunda aproximación, en la que utilizamos los fonemas reconocidos por *PhnRec*, en lugar de los vectores de posteriorgramas. Esta aproximación tiene un coste temporal menor que la anterior, ya que se basa en encontrar subsecuencias comunes entre los pares, problema con un coste temporal de $\mathcal{O}(nm)$, siendo n y m la longitud de la secuencia de fonemas de cada audio.

Teniendo en cuenta que en este caso obtenemos un conjunto de pares cuyas secuencias fonéticas son iguales, podemos agrupar todos los fragmentos cuya decodificación sea igual, de forma que la salida de nuestro sistema sea un conjunto de *clusters*, donde cada uno este formado por los fragmentos de igual decodificación fonética.

Como expresamos con la anterior aproximación, es importante establecer un umbral mínimo de longitud para discriminar si se trata de una palabra.

En este caso, tratamos con secuencias de fonemas, por lo que deberíamos implantar un número mínimo de fonemas para considerar que la secuencia puede corresponder a una palabra. Para comprobar como afecta este umbral a los resultados, hemos probado diferentes valores mínimos de longitud de las secuencias de fonemas. En la tabla 3.3 se pueden observar los resultados para los conjuntos de datos de los dos idiomas según este valor.

			Matching			Grouping			Type			Token			Boundary		
#fonemas	NED	Cov	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
									Inglés								
≥3	79.7	97.7	1.8	1.3	1.5	2.1	8.2	3.4	5.2	29.8	8.9	3.7	10.8	5.5	32.1	56.6	40.9
≥4	75.0	80.5	2.8	0.5	0.9	2.6	29.9	4.8	3.3	12.8	5.3	2.4	3.0	2.7	32.2	41.5	36.3
≥5	69.1	33.8	5.2	0.2	0.3	5.7	85.9	10.2	1.8	2.0	1.9	1.5	0.4	0.6	32.7	12.3	17.9
≥6	66.5	6.1	7.3	0.0	0.1	20.7	93.8	33.3	1.6	0.2	0.4	1.3	0.0	0.1	32.6	1.6	3.0
									Tsonga								
≥3	49.2	89.9	15.7	1.8	3.2	16.4	9.4	11.9	4.0	15.6	6.3	1.9	11.9	3.3	18.8	65.0	29.2
≥4	41.0	74.2	21.1	1.0	1.8	22.0	19.1	20.5	3.6	11.6	5.5	2.0	5.7	3.0	19.6	49.7	28.1
≥5	28.9	37.5	32.4	0.5	0.9	33.0	46.0	38.4	3.3	4.8	3.9	2.4	1.9	2.1	20.9	19.8	20.3
≥6	17.3	11.9	47.2	0.2	0.4	49.1	63.0	55.0	4.0	1.6	2.3	2.9	0.6	0.9	22.7	5.2	8.4

Tabla 3.3: Resultados según número mínimo de fonemas.

A partir de la tabla 3.3 podemos ver como se comportan los sistemas dependiendo del umbral mínimo de detección de un posible patrón repetido. Cuanto mayor es el valor de n, las diferencias entre los fragmentos son menores (mejor *NED*) y los *clusters* son de mejor calidad (mejor *grouping*). Sin embargo, también hace que el resto de métricas empeoren, ya que estamos reduciendo drásticamente el número de fragmentos devueltos por el sistema (menor *coverage*).

Además, si comparamos los resultados entre ambos idiomas, vemos variaciones según las métricas. En la distancia de edición normalizada (*NED*), en tsonga los valores son mas bajos con mayor diferencia conforme se aumenta el número de fonemas. También se comporta mejor el sistema con los audios en tsonga en la precisión del *matching* o del *grouping*. Por otra parte, en inglés se obtienen mejores valores en el *recall* del *grouping* y en la precisión del *boundary*.

A partir de estos resultados hemos probado a eliminar aquellos *frames* cuya probabilidad más alta hace referencia a un posible silencio o ruido. Dado que el conjunto de audios en el idioma tsonga es más reducido que los audios en inglés, esta comparación se ha realizado únicamente en este idioma por rapidez. Los resultados obtenidos eliminando o no los silencios se muestran en la tabla 3.4.

Si observamos los resultados de la tabla 3.4, la eliminación de los silencios comporta una menor distancia de edición entre los fragmentos detectados y una mejora en la métrica *group*, por lo que los grupos de coincidencias

			Matching			Grouping			Type			Token			Boundary		
#fonemas	NED	Cov	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
Tsonga (con silencios)																	
≥3	49.2	89.9	15.7	1.8	3.2	16.4	9.4	11.9	4.0	15.6	6.3	1.9	11.9	3.3	18.8	65.0	29.2
≥4	41.0	74.2	21.1	1.0	1.8	22.0	19.1	20.5	3.6	11.6	5.5	2.0	5.7	3.0	19.6	49.7	28.1
≥5	28.9	37.5	32.4	0.5	0.9	33.0	46.0	38.4	3.3	4.8	3.9	2.4	1.9	2.1	20.9	19.8	20.3
Tsonga (sin silencios)																	
≥3	46.5	76.5	17.9	1.6	3.0	18.7	12.9	15.3	3.4	11.5	5.3	1.4	6.2	2.2	15.9	47.8	23.9
≥4	38.5	56.7	23.7	0.8	1.6	25.0	25.2	25.1	3.2	8.1	4.6	1.7	3.4	2.3	17.1	32.8	22.4
≥5	25.7	26.2	35.8	0.4	0.8	36.8	53.0	43.4	2.5	2.6	2.6	1.7	0.9	1.2	18.0	11.8	14.3

Tabla 3.4: Resultados según eliminación de silencios.

son más similares fonéticamente. Sin embargo, el resto de métricas se ven perjudicadas, destacando la cobertura.

Es posible que, dado que estamos utilizando un modelo de otro idioma, parte de los fonemas característicos del idioma evaluado no tengan correspondencia, por lo que se clasifican por el reconocedor como silencios o ruidos.

Búsqueda por fonemas aproximada

La tercera aproximación intenta profundizar en la anterior. Con la búsqueda de secuencias de fonemas iguales estamos siendo muy estrictos con la coincidencia de estos fonemas.

Por ello, introducimos una nueva aproximación utilizando *Lucene*. Este *software* nos permite realizar búsquedas de texto con la posibilidad de que la coincidencia no sea exacta, mediante las *Proximity Searches*. Con este tipo de búsquedas podemos obtener coincidencias con documentos que incluyan la secuencia que buscamos o pequeñas variaciones, basándose en una distancia de edición llamada *slop*.

Como se ha explicado anteriormente, con este tipo de búsquedas necesitamos realizar un segundo paso, ya que obtenemos los audios que contienen la secuencia de fonemas que buscamos. Para determinar las marcas temporales donde se ubican exactamente utilizamos el algoritmo *Subsequence Dynamic Time Warping*, que se detalla en el apartado correspondiente a la tarea de *Query by Example*.

En primer lugar, para comparar esta aproximación con la anterior, obtenemos los resultados con valores de *slop* de 0, equivalentes a realizar búsquedas exactas. Los resultados obtenidos se muestran en la tabla 3.5.

			Matching			Grouping			Type			Token			Boundary		
#fonemas	NED	Cov	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
								Inglés									
≥5	75.7	61.8	3.1	0.2	0.4	3.0	28.2	5.3	3.6	10.3	5.4	2.6	3.0	2.8	30.2	31.8	31.0
≥6	72.8	16.3	5.1	0.1	0.2	8.2	61.8	13.8	2.0	1.0	1.3	1.7	0.2	0.4	30.6	5.3	9.1
≥7	79.0	2.8	6.4	0.0	0.1	20.7	66.7	31.6	1.0	0.1	0.1	0.8	0.0	0.0	32.5	0.7	1.4
								Tsonga									
≥5	41.7	57.4	22.8	0.5	1.0	22.8	14.1	17.4	2.7	6.5	3.8	1.6	3.8	2.2	16.4	33.7	22.1
≥6	26.9	20.5	37.9	0.3	0.5	38.7	34.0	35.7	2.9	2.2	2.5	2.3	1.1	1.4	18.0	8.9	11.9
≥7	18.1	5.8	51.8	0.2	0.3	49.8	42.7	45.0	1.7	0.3	0.5	1.0	0.1	0.2	18.2	2.2	3.8

Tabla 3.5: Resultados según número mínimo de fonemas.

No podemos comparar las tablas 3.3 y 3.5, ya que en la aproximación anterior los fragmentos de igual decodificación fonética se sitúan en un único *cluster*. En la aproximación actual no se está realizando este proceso. Sin embargo, no es difícil implementarlo, ya que podemos unir en un mismo *cluster* todos los fragmentos obtenidos en una misma búsqueda. El resultado de este *clustering* se puede observar en la tabla 3.6.

			Matching			Grouping			Type			Token			Boundary		
#fonemas	NED	Cov	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
								Inglés									
≥5	82.3	56.7	2.2	0.2	0.4	1.7	17.1	3.1	3.4	8.1	4.8	2.6	1.9	2.2	30.0	26.7	28.3
≥6	78.4	15.0	4.0	0.1	0.1	9.2	72.0	16.1	1.8	0.8	1.1	1.7	0.2	0.3	30.4	4.6	7.9
≥7	83.7	2.5	5.2	0.0	0.1	8.3	100.0	15.4	0.5	0.0	0.1	0.4	0.0	0.0	32.1	0.6	1.2
								Tsonga									
≥5	61.6	53.6	17.6	0.6	1.2	14.2	19.4	16.4	2.2	4.9	3.0	1.4	2.0	1.6	15.8	28.3	20.3
≥6	34.8	19.4	36.4	0.3	0.5	30.3	58.4	39.7	2.5	1.8	2.1	2.1	0.6	0.9	17.3	7.3	10.3
≥7	19.3	5.6	55.6	0.2	0.3	45.5	73.7	55.4	1.3	0.2	0.4	1.1	0.1	0.1	18.3	1.8	3.3

Tabla 3.6: Resultados tras realizar *clustering*.

Si comparamos las tablas 3.3 y 3.6, podemos ver como con la actual aproximación, los valores son, en general, peores. Esto se puede deber a la necesidad de añadir una segunda etapa de búsqueda en esta aproximación, de forma que la detección de los fragmentos utilizando el algoritmo *Subsequence DTW* supone una nueva fuente de posibles errores.

En cualquier caso, el objetivo principal de esta aproximación es comprobar si introduciendo una posible variación en la secuencia de fonemas, los resultados pueden mejorar en determinadas métricas. Por ejemplo, esta variabilidad podría hacer que la calidad del *matching* sea peor, pero podría recoger un mayor número de coincidencias.

En los resultados no se ha realizado la etapa de *clustering* por la dificultad que supone unir en un mismo *cluster* diferentes decodificaciones fonéticas. Los resultados se pueden comprobar en la tabla 3.7.

				Matching			Grouping			Type			Token			Boundary		
#fons	slop	NED	Cov	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
									Inglés									
≥6	0	72.8	16.3	5.1	0.1	0.2	8.2	61.8	13.8	2.0	1.0	1.3	1.7	0.2	0.4	30.6	5.3	9.1
≥6	1	78.3	44.3	2.7	0.2	0.3	3.0	63.4	5.6	2.4	4.0	3.0	1.9	0.8	1.2	30.5	17.4	22.1
≥7	0	79.0	2.8	6.4	0.0	0.1	20.7	66.7	31.6	1.0	0.1	0.1	0.8	0.0	0.0	32.5	0.7	1.4
≥7	1	76.6	10.6	4.9	0.1	0.1	13.8	75.0	22.3	1.3	0.3	0.5	1.1	0.1	0.1	31.4	2.8	5.1
≥7	2	81.2	32.4	1.7	0.1	0.2	3.3	60.4	6.2	1.6	1.6	1.6	1.3	0.3	0.5	30.8	10.5	15.7
									Tsonga									
≥6	0	26.9	20.5	37.9	0.3	0.5	38.7	34.0	35.7	2.9	2.2	2.5	2.3	1.1	1.4	18.0	8.9	11.9
≥6	1	39.0	42.2	26.6	0.5	0.9	25.3	30.7	27.7	2.6	4.3	3.3	1.9	1.9	1.9	17.5	19.4	18.4
≥7	0	18.1	5.8	51.8	0.2	0.3	49.8	42.7	45.0	1.7	0.3	0.5	1.0	0.1	0.2	18.2	2.2	3.8
≥7	1	25.2	14.1	41.7	0.3	0.5	39.8	47.8	42.9	1.8	0.8	1.1	1.2	0.3	0.4	18.3	4.9	7.8
≥7	2	43.4	31.3	24.1	0.4	0.8	21.9	42.7	28.8	2.1	2.4	2.2	1.3	0.7	0.9	17.5	11.9	14.1

Tabla 3.7: Resultados con diferentes valores de *slop*.

A partir de esta tabla podemos ver cómo se comporta el sistema si aceptamos mayor variabilidad en las búsquedas. En términos generales, esto comporta peores resultados en la precisión del *matching* y del *grouping*. Por otra parte, se mejoran los valores del *recall* del *boundary*. Es decir, parece que los fragmentos están más ajustados en cuanto a tiempo, pero hace que la calidad de los emparejamientos sea bastante peor.

3.3.2. Resultados oficiales

A continuación se pueden observar los resultados obtenidos por los sistemas *baseline* y *topline* en la tabla 3.8.

			Matching			Grouping			Type			Token			Boundary		
	NED	Cov	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
									Inglés								
baseline	21.9	16.3	39.4	1.6	3.1	21.4	84.6	33.3	6.2	1.9	2.9	5.5	0.4	0.8	44.1	4.7	8.6
topline	0	100	98.3	18.5	31.1	99.5	100	99.7	50.3	56.2	53.1	68.2	60.8	64.3	88.4	86.7	87.5
									Tsonga								
baseline	12	16.2	69.1	0.3	0.5	52.1	77.4	62.2	3.2	1.4	2	2.6	0.5	0.8	22.3	5.6	8.9
topline	0	100	100	6.8	12.7	100	100	100	15.1	18.1	16.5	34.1	49.7	40.4	66.6	91.9	77.2

Tabla 3.8: Resultados *baseline* y *topline*.

El sistema *baseline* consiste en aplicar el sistema de Jansen y Van Durme [7], donde se aplica un proceso basado en DTW modificado para ser más eficiente. Después de este proceso, se realiza un segundo paso de *clustering*. El sistema *topline* utiliza las transcripciones fonéticas y una gramática de unigramas de fonemas [15].

Si comparamos estos resultados —especialmente el *baseline*— con los obtenidos con la segunda aproximación (*Longest Common Substring*), podemos

comprobar como en determinadas métricas conseguimos un mejor resultado que el *baseline* (ver tabla 3.9)

			Matching			Grouping			Type			Token			Boundary		
	NED	Cov	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
									Inglés								
baseline	21.9	16.3	39.4	1.6	3.1	21.4	84.6	33.3	6.2	1.9	2.9	5.5	0.4	0.8	44.1	4.7	8.6
LCS≥3	79.7	97.7	1.8	1.3	1.5	2.1	8.2	3.4	5.2	29.8	8.9	3.7	10.8	5.5	32.1	56.6	40.9
LCS≥4	75.0	80.5	2.8	0.5	0.9	2.6	29.9	4.8	3.3	12.8	5.3	2.4	3.0	2.7	32.2	41.5	36.3
									Tsonga								
baseline	12	16.2	69.1	0.3	0.5	52.1	77.4	62.2	3.2	1.4	2	2.6	0.5	0.8	22.3	5.6	8.9
LCS≥3	49.2	89.9	15.7	1.8	3.2	16.4	9.4	11.9	4.0	15.6	6.3	1.9	11.9	3.3	18.8	65.0	29.2
LCS≥4	41.0	74.2	21.1	1.0	1.8	22.0	19.1	20.5	3.6	11.6	5.5	2.0	5.7	3.0	19.6	49.7	28.1

Tabla 3.9: Comparación entre *baseline* y nuestros sistemas.

En concreto, en las métricas *type*, *token* y *boundary* conseguimos mayores valores de *recall*, obteniendo también un mejor valor de *F-score*. Es decir, aunque la precisión se ve reducida, la ganancia en *recall* es suficientemente alta como para compensar esta pérdida.

A continuación, vamos a comparar los resultados que hemos obtenido con los resultados de los otros grupos. Los resultados oficiales de la tarea se muestran en la tabla 3.10[3].

			Matching			Grouping			Type			Token			Boundary		
Sistema	NED	Cov	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
									Inglés								
Räsänen [1]	89.6	40.6				4.0	10.8	5.7	13.5	11.3	12.3	21.6	4.8	7.9	76.1	28.5	41.4
Räsänen [2]	88.0	42.2				4.3	10.6	6.0	12.7	10.8	11.6	21.6	4.7	7.8	75.7	27.4	40.3
Räsänen [3]	70.8	42.4				13.4	15.7	14.2	14.1	12.9	13.5	22.6	6.1	9.6	75.7	33.7	46.7
Lyzinski [1]	77.3	25.5	14.6	2.3	4.0				4.7	2.5	3.3	4.2	0.6	1.0	39.6	7.5	12.7
Lyzinski [2]	61.2	80.2	6.5	3.5	4.6				3.1	9.2	4.6	2.4	3.5	2.8	35.4	38.5	36.9
Lyzinski [3]	36.4	46.7	12.9	5.1	7.2				2.3	2.9	2.6	1.9	0.7	1.0	31.7	14.2	19.6
									Tsonga								
Räsänen [1]	78.4	77.7				6.2	3.2	4.3	1.7	4.1	2.4	1.8	1.8	1.8	26.2	26.3	26.3
Räsänen [2]	61.2	95.0				17.7	2.8	4.9	1.1	3.3	1.7	0.8	1.3	1.0	16.3	24.4	19.5
Räsänen [3]	63.1	94.7				10.7	3.3	5.0	2.2	6.2	3.3	2.3	3.4	2.7	29.2	39.4	33.5
Lyzinski [1]	36.1	30.2	30.6	0.6	1.2				3.0	2.7	2.8	2.0	0.9	1.2	19.4	11.2	14.2
Lyzinski [2]	43.2	89.4	21.2	3.8	6.5				4.9	18.8	7.8	2.2	12.6	0.8	18.8	64.0	29.0
Lyzinski [3]	34.1	67.6	13.3	7.4	9.5				2.6	6.0	3.6	1.5	2.3	2.0	14.8	29.5	19.7

Tabla 3.10: Resultados oficiales de la tarea.

Al comparar estos sistemas con los nuestros vemos como estos resultados son, en términos generales, mejores. Por ejemplo, destacan los valores de *matching* de los sistemas presentados por Lyzinski et ál. Aún así, por ejemplo, en el caso del tsonga, con nuestros sistemas conseguimos una mayor calidad

[3]Las métricas en blanco no están disponibles.

de los *clusters* (según la métrica *clustering*) que los sistemas de Räsänen et ál., entre otros.

Los sistemas presentados por Räsänen et ál. siguen una aproximación de *zero-resources*. Para ello, se intenta descubrir unidades silábicas, a partir de las cuales se buscan secuencias de sílabas. Los tres sistemas presentados difieren en el algoritmo empleado para la detección de estas unidades.

En el caso de los sistemas de Lyzinski et ál. se trata de dos sistemas *zero-resources* y un sistema *low-resources*. Mediante el sistema de *Segmental DTW* propuesto por Jansen y Van Durme [7] se obtiene un grafo de fragmentos que indica la similitud entre ellos y a partir del cual se obtienen los *clusters* finales. Los sistemas utilizan diferentes características y algoritmos de *clustering*. Es el tercer sistema el que sigue una aproximación *low-resources*, ya que utiliza como características los *Bottleneck Features*, obtenidos con una red neuronal entrenada con audios del corpus inglés Fisher.

En la tabla 3.11 se indica que caracteriza a cada uno de los sistemas presentados. Más información sobre estos sistemas se pueden encontrar en los respectivos artículos [16, 17].

Sistema	Característica principal
Räsänen [1]	Vseg algorithm
Räsänen [2]	Envelope Minima Detection
Räsänen [3]	Amplitude envelope-driven oscillator
Lyzinski [1]	Connected Components - PLP
Lyzinski [2]	Connected Components - FDLPS
Lyzinski [3]	FastGreed - BNF (Eng)

Tabla 3.11: Sistemas presentados a la tarea.

Capítulo 4

Query by Example

En este capítulo vamos a presentar la segunda tarea que abordamos en el presente trabajo. En este caso la tarea es conocida como *Query by Example*. De forma similar a la primera tarea, primero explicaremos brevemente en que consiste esta tarea. Posteriormente, entraremos a detallar el estado del arte en este ámbito.

4.1. Definición de la tarea

El objetivo de la tarea *Query by Example* es realizar una búsqueda en una colección de documentos de audio a partir de una consulta también en audio. Como veremos más adelante, es una tarea más o menos resuelta en su versión más simple, pero que está lejos de estar resulta en condiciones complejas y más cercanas a la realidad.

En este tipo de tarea disponemos de un conjunto de documentos de audio que serán el objetivo de la búsqueda. Para ello, partimos de una consulta, también en formato de audio con la que realizar la búsqueda. La dificultad reside en que no se dispone de otra información, como podrían ser transcripciones de los audios, por lo que debe ser una búsqueda de audio a audio.

4.2. Estado del arte

Müller, en el libro *Information Retrieval for Music and Motion* [18], presenta como el algoritmo de *Dynamic Time Warping*, ya utilizado en otras tareas del ámbito del procesamiento del lenguaje natural como el reconocimiento del habla, puede servir de base para otras tareas.

Entre otras, plantea una variante del DTW que permite encontrar el mejor alineamiento de una secuencia dentro de otra de longitud mayor. Müller

denomina a esta variante *Subsequence DTW*, ya que el algoritmo tiene como objetivo obtener la subsecuencia que mejor se ajusta a la secuencia de referencia. Para ello, permite que el alineamiento pueda comenzar en cualquier punto de la secuencia de mayor longitud. Si aplicamos este algoritmo a la tarea de *Query by Example*, nos permite buscar la subsecuencia más parecida a una *query* en cualquier documento.

Anguera y Ferrarons [19] plantean que el algoritmo propuesto por Müller anteriormente, siendo simple y efectivo, también es poco eficiente en términos de memoria, ya que requiere almacenar una matriz de tamaño NxM donde N es la longitud de la *query* y M, la longitud de la colección de documentos. También introducen la posibilidad de aplicar normalizaciones (locales o globales) en el algoritmo, de forma similar a Muscariello et ál. en el caso de una variante del *Segmental DTW* [20].

Muscariello et ál.[21] también introducen la posibilidad de mejorar el sistema de *Query by Example* añadiendo la utilización de matrices de similitud propia (*Self Similarity Matrix*). En este caso, se obtienen las matrices de similtud propia tanto de la *query* como del fragmento coincidente según el algoritmo de *DTW* (figura 4.1). Estas matrices se comparan obteniendo un *score* que representa la similitud de ambas secuencias de audio. Finalmente, se tendrán en cuenta los dos algoritmos para asignar el *score* final.

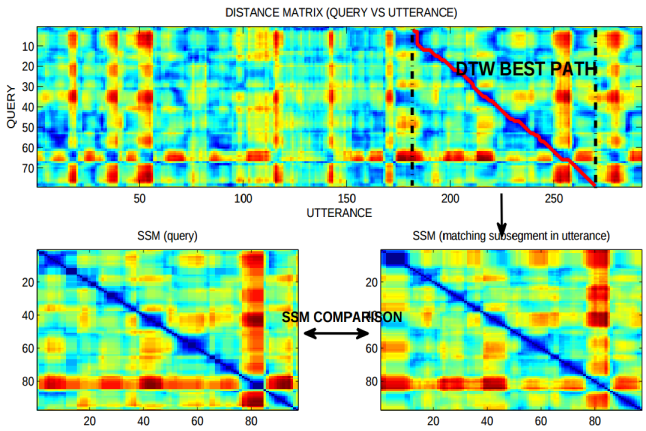

Figura 4.1: Combinación de *DTW* y *Self Similarity Matrix* [21].

Para aplicar cualquier algoritmo basado en *Dynamic Time Warping* es necesario primero obtener una representación adecuada de los fragmentos de audio que queremos comparar. Entre varias posibilidades, una que está bastante extendida es el uso de posteriorgramas, como se ha visto en la

anterior tarea. Existen diferentes tipos de posteriorgramas según como se definan estas clases.

Hazen et ál. [22] muestran como se pueden utilizar los posteriorgramas fonéticos para la tarea de *Query by Example*. En este tipo de posteriorgramas las clases se corresponden con unidades fonéticas, de forma que se los posteriorgramas están formados por un vector de probabilidades referidas a diferentes fonemas. Para obtener esta representación se utiliza un reconocedor fonético.

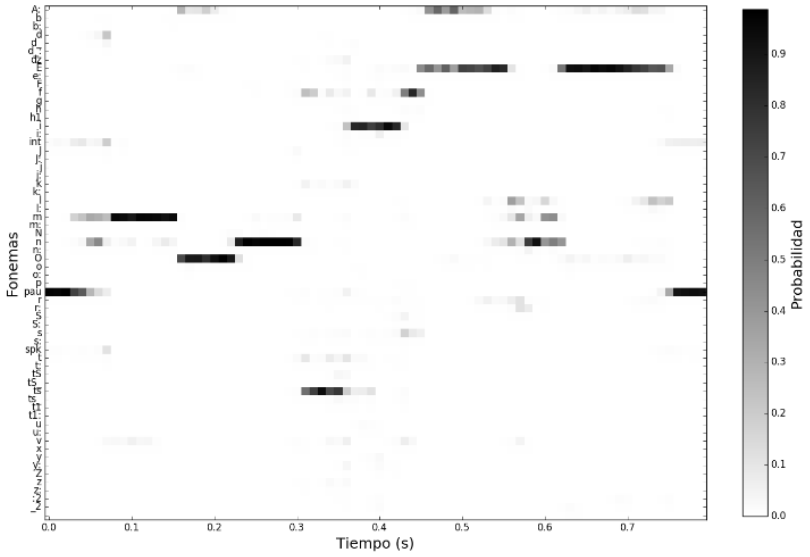

Figura 4.2: Ejemplo de posteriorgrama de la palabra “manzana”.

Por otra parte, Zhang y Glass [23] plantean utilizar los audios para obtener un modelo basado en mixturas de gaussianas, obteniendo en este caso posteriorgramas gaussianos. En este caso, el modelo se obtiene de forma no supervisada.

Si la secuencia de audio se define como una secuencia de *frames* $S = (s_1, s_2, \ldots, s_n)$, el posteriorgrama gaussiano se define como una secuencia de vectores de igual longitud que S.

$$PG(S) = (q_1, q_2, \ldots, q_n) \tag{4.1}$$

Cada vector q_i del posteriorgrama viene determinado por la siguiente ecuación

$$q_i = (P(C_1|s_i), P(C_2|s_i), \ldots, P(C_m|s_i)) \tag{4.2}$$

donde C_i hace referencia a la i-ésima componente gaussiana del modelo y m es el número de componentes.

Por último, Abad et al. [24] plantean la posibilidad de utilizar múltiples sistemas para, posteriormente, fusionarlos. En este caso, los sistemas son de tipo *low-resources* y utilizan modelos acústicos de otros idiomas. Por ello, cada sistema utiliza el modelo de un idioma diferente. Tras obtener los resultados de cada sistema por separado se introduce una etapa final de fusión de los *scores*, explorando varias alternativas, mostrando una mejora significativa del sistema final.

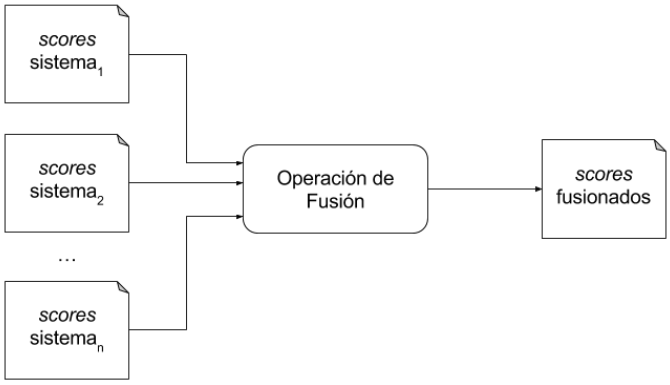

Figura 4.3: Fusión de sistemas.

Capítulo 5

Query by Example Search on Speech Task 2015

Query by Example Search on Speech Task (QUESST) [25] es una tarea presentada en las ediciones 2014 y 2015 de MediaEval. MediaEval es una iniciativa de *benchmaking* que permite evaluar nuevos algoritmos dedicados al acceso o recuperación de información multimedia. Entre los diferentes aspectos que cubre, se encuentran el reconocimiento del habla, análisis de contenidos multimedia, análisis de audio y música o redes sociales.

En cuanto a la tarea que nos incumbe, QUESST, se ha presentado en dos ediciones consecutivas, sin embargo, deriva de la tarea *Spoken Web Search Task*, presente en MediaEval desde la edición 2011. En estas ediciones, las tareas han aumentado su dificultad abriendo nuevos escenarios de gran interés, por ello en los siguientes apartados vamos a describir no solo la tarea de la edición a presentar, también los cambios que se han incorporado en ediciones anteriores.

5.1. Descripción de la tarea

En la edición de 2011 del *MediaEval Benchmark*, se presenta por primera vez la tarea *Spoken Web Search*. En esta primera edición se introducen las bases de la tarea del año 2015. El objetivo de la tarea es buscar entre un conjunto de audios aquellos que se correspondan con una búsqueda también en audio. Además, ya se plantea que el sistema que se debe desarrollar sea independiente del idioma, debido a que los audios proporcionados incluirán varios idiomas.

Durante las diversas ediciones se han ido planteando cambios como modificar los idiomas utilizados en los audios o el tamaño de estos. Sin embargo,

los cambios más destacables vienen con el cambio del nombre de la tarea a *Query by Example Search on Speech Task*. En la edición de 2014 se plantean tres modificaciones importantes:

1. **No se debe indicar la localización temporal de las palabras de búsqueda.**

2. **Nuevos tipos de búsquedas.**
   Se introducen búsquedas más complejas, donde la similitud no es completa. En esta tarea, se permiten pequeñas diferencias léxicas entre palabras o cambios del orden si la búsqueda consiste en más de una palabra.

3. **Nueva métrica principal.**
   En esta edición, se introduce una nueva métrica con el objetivo de que no se evalúe de forma binaria, si no en base a un nivel de confianza. En el apartado 5.1.1 se incluye más información sobre las métricas.

Entrando ya de forma concreta en la tarea *QUESST* 2015, debemos destacar algunas características que hacen que sea más compleja que una tarea de *Query by Example* "típica".

Para empezar, tenemos un corpus multilingüe, es decir, los audios que forman el conjunto de datos están en diferentes idiomas, en este caso, siete idiomas: albanés, rumano, eslovaco, checo, portugués, inglés y mandarín. En el caso de estos dos últimos idiomas, se trata de audios donde aparecen ambos idiomas de forma alternada, produciéndose el efecto de *code-switching*.

Como se ha explicado anteriormente, existen diferentes tipos de búsquedas en esta tarea. La división entre estos tipos va asociado a una mayor dificultad, pero también conlleva que nuestro sistema deba adaptarse a escenarios más cercanos a la realidad. Los tres tipos de búsquedas son:

- **Tipo 1 (T1)**.
  En este tipo de búsquedas, la palabra o palabras a buscar están en el documento de forma exacta. Se trata del tipo de búsquedas más sencillo.

- **Tipo 2 (T2)**.
  En este segundo tipo, pueden darse pequeñas variaciones entre la *query* y el documento. Existen dos posibles variaciones: diferencias léxicas o reordenamiento de las palabras.
  En el primer caso podrían entrar cambios de género o número o diferentes desinencias verbales (por ejemplo, "casa roja" y "casas rojas" o "escribir" y "escribió").

En el segundo caso, si se trata de una búsqueda de varias palabras, debemos detectar si las palabras están en diferente orden o, incluso, si existen otros términos entre las palabras .
Además, ambos tipos de variaciones se pueden producir en una misma búsqueda (por ejemplo, "blanca nieves" y "nieve blanca").

- **Tipo 3 (T3)**.
  Dentro de este tipo de búsquedas tenemos consultas con contexto, es decir, la consulta también contiene términos irrelevantes o pausas entre las diferentes palabras que forman la frase. Debido a la dificultad de determinar si un término es relevante o no, se proporciona información temporal de donde se ubican las palabras de relevancia para la búsqueda.

El conjunto de datos de la tarea se divide en tres subconjuntos: documentos de audio, *queries* de desarrollo y *queries* de evaluación. En la tabla 5.1 se pueden ver el número de audios de cada subconjunto.

	# audios
Documentos	11 662
Queries Dev	445
Queries Eval	447

Tabla 5.1: Información sobre el corpus *QUESST* 2015.

5.1.1. Métricas

Antes de entrar en la explicación de cómo hemos afrontado esta tarea, vamos a ver con más detalle las métricas que se emplean para evaluar los sistemas que se presentan. Como se ha visto anteriormente, en esta tarea se utilizan dos métricas: *Cross entropy score* y *Actual Term Weighted Value* [26]. La primera sustituyó en la anterior edición a la segunda como métrica principal. Para ver que ventajas o inconvenientes tiene cada una de ellas, vamos a describirlas con mayor nivel de detalle.

Actual Term Weighted Value

La métrica *Actual Term Weighted Value* se calcula a partir de la decisión del sistema de los audios que contienen la búsqueda. Dado que es necesario proporcionar un *score* para todo par de búsqueda y audio, los pares cuyo *score* supere un determinado umbral (θ) se considerarán como positivos.

Esta métrica asume que el sistema es perfecto, para penalizar posteriormente los errores del sistema. Para ello, utiliza los falsos negativos y los falsos positivos. El ratio de falsos negativos se calcula siguiendo la ecuación 5.1

$$P_{miss}(q,\theta) = \frac{N_{miss}(q,\theta)}{N_{act}(q)} \tag{5.1}$$

donde $N_{miss}(q,\theta)$ es el número de ocurrencias de la *query* q no encontradas con un umbral θ y $N_{act}(q)$ es el número total de ocurrencias de la *query* en los documentos de audio.

En cuanto al ratio de falsos positivos, se calcula a partir de la ecuación 5.2

$$P_{fa}(q,\theta) = \frac{N_{fa}(q,\theta)}{N_{nt}(q)} \tag{5.2}$$

donde $N_{fa}(q,\theta)$ es el número de falsos positivos de la *query* q con un umbral θ y $N_{nt}(q)$ es el máximo posible de falsos positivos. Este último valor se determina estableciendo un número de posibles coincidencias por segundo (n_{tps}), habitualmente una, y siguiendo la ecuación $N_{nt}(q) = n_{tps} \cdot T_{audio} - N_{act}(q)$.

Finalmente, a partir del umbral determinado, el *Term Weighted Value* se calcula siguiendo la ecuación 5.3.

$$TWV(\theta) = 1 - \frac{1}{|\mathcal{Q}|} \sum_{\forall q \in \mathcal{Q}} \left(P_{miss}(q,\theta) + \beta \cdot P_{fa}(q,\theta)\right) \tag{5.3}$$

donde $\mathcal{Q}$ es el conjunto de *queries* y el parámetro β está definido por la ecuación 5.4.

$$\beta = \frac{C_{fa} \cdot (1 - P_{target})}{C_{miss} \cdot P_{target}} \tag{5.4}$$

Para esta tarea, los valores de los parámetros a utilizar en la ecuación 5.4 son los mismos que en la edición del *QUESST* 2014 [27].

$$P_{target} = 0.0008$$
$$C_{fa} = 1$$
$$C_{miss} = 100$$

Además del *Actual Term Weighted Value*, también se calcula el *Maximum Term Weighted Value* (MTWV). Para calcular esta métrica, se determina un umbral por cada *query* de forma que se maximice el TWV sin modificar los *scores* proporcionados.

Para un sistema perfecto, el valor de esta métrica será 1. En el caso que el sistema no proporcione ningún tipo de información, el valor será 0. Por

último, si el umbral que determina si un documento está asociado a una *query* no es el adecuado, el valor del *Term Weighted Value* puede ser negativo. Por este motivo es útil el *Maximum Term Weighted Value*.

Normalized Cross Entropy Score

La métrica *Normalized Cross Entropy Score* (C_{nxe}) se basa en una puntuación por cada posible par de búsqueda y audio. Esta métrica asume que el *score* asociado a cada par es un ratio de *log-likelihood*. Al final, el objetivo es indicar si el sistema ofrece unos *scores* calibrados adecuadamente.

Para empezar, partiendo de los parámetros anteriores P_{target}, C_{fa} y C_{miss}, se calcula un nuevo parámetro, P_{tar}, siguiendo la ecuación 5.5.

$$P_{tar} = \frac{C_{miss} \cdot P_{target}}{C_{miss} \cdot P_{target} + C_{fa} \cdot (1 - P_{target})} \tag{5.5}$$

Además, se definen los conjuntos $T_{true}(\mathcal{S})$ y $T_{false}(\mathcal{S})$, que contienen los pares de documentos y *queries* devueltos por el sistema $\mathcal{S}$ que son correctos o erróneos, respectivamente.

$$T_{true}(\mathcal{S}) = \{t \in T(\mathcal{S}) \mid \mathcal{G}_t = true\}$$
$$T_{true}(\mathcal{S}) = \{t \in T(\mathcal{S}) \mid \mathcal{G}_t = false\}$$

La métrica parte de una función de coste para cada par según si el documento se corresponde con la *query* o no. Esta función de coste está definida en la ecuación 5.6.

$$C_{log}(llr_t) = \begin{cases} -\log\left(\text{sigmoid}\left(llr_t + \log\left(\frac{P_{tar}}{1-P_{tar}}\right)\right)\right) & t \in T_{true}(\mathcal{S}) \\ -\log\left(\text{sigmoid}\left(-\left(llr_t + \log\left(\frac{P_{tar}}{1-P_{tar}}\right)\right)\right)\right) & t \in T_{false}(\mathcal{S}) \end{cases} \tag{5.6}$$

A partir de aquí se calcula la llamada entropía cruzada empírica, partiendo del coste de todos los pares devueltos por el sistema (ecuación 5.7).

$$C_{xe} = \frac{1}{\log 2} \cdot \left(\frac{P_{tar}}{|T_{true}(\mathcal{S})|} \sum_{t \in T_{true}(\mathcal{S})} C_{log}(llr_t) + \frac{1 - P_{tar}}{|T_{false}(\mathcal{S})|} \sum_{t \in T_{false}(\mathcal{S})} C_{log}(llr_t) \right) \tag{5.7}$$

Lo que nos va a permitir obtener nuestra métrica, es el valor de entropía que alcanzaría un sistema que asignase a cualquier par el mismo *score*. De esta forma, y siguiendo la ecuación 5.8, calculamos la entropía *a priori*.

$$C_{xe}^{prior} = \frac{1}{\log 2} \cdot \left(P_{tar} \cdot \log \frac{1}{P_{tar}} + (1 - P_{tar}) \cdot \log \frac{1}{1 - P_{tar}} \right) \tag{5.8}$$

Por lo tanto, la *Normalized Cross Entropy* no es más que el ratio entre la entropía cruzada empírica y la entropía *a priori*, como se puede ver en la ecuación 5.9.

$$C_{nxe} = \frac{C_{xe}}{C_{xe}^{prior}} \tag{5.9}$$

En el mejor de los casos, un sistema conseguiría un valor de C_{nxe} cercano a cero. Si el sistema no es capaz de extraer la información, el valor estaría cercano a uno. Por último, si la métrica supera el valor de uno, estaría indicando que el sistema no tiene una calibración adecuada de los *scores*.

Debido a esto último es importante saber si, ante un valor elevado de la entropía cruzada, el problema es únicamente de calibración o, por el contrario, el sistema es incapaz de resolver la tarea. Para ello, se ofrece la métrica secundaria *Minimum Normalized Cross Entropy Score* (C_{nxe}^{min}), que muestra el valor que puede alcanzar el *Normalized Cross Entropy Score* si la calibración de los *scores* es la más adecuada posible.

Mediante una transformación lineal se intenta recalibrar el sistema, como se muestra en la ecuación 5.10

$$\hat{llr}_t = \gamma \cdot llr_t + \delta \tag{5.10}$$

donde γ y δ serán los parámetros utilizados para minimizar la entropía cruzada, siguiendo la ecuación 5.11.

$$C_{nxe}^{min} = \min_{\gamma,\delta} \hat{C}_{nxe} \tag{5.11}$$

Las métricas están ampliamente explicadas en el informe correspondiente de la tarea del año 2013 [26].

5.2. Descripción de los sistemas

Para la tarea, se presentaron dos sistemas (uno principal y otro secundario) [28]. Ambos sistemas comparten, en general, la aproximación a la tarea. En este apartado, vamos a detallar esta aproximación y, posteriormente, explicaremos las diferencias entre ambos. Será en el próximo apartado donde se explique la experimentación llevada a cabo para alcanzar los sistemas finales.

5.2.1. Parametrización

Con el objetivo de parametrizar los ficheros de audio (tanto los documentos como las consultas), hemos utilizado el reconocedor fonético *PhnRec* [11], de forma similar a la tarea de *Spoken Term Discovey*. Como ya hemos

visto anteriormente, este reconocedor es capaz de ofrecer una decodificación fonética a partir de un audio.

Una vez parametrizados los audios, vamos a utilizar la información del contexto de las *queries* para eliminarlo. Lo eliminamos en este punto y no antes de la parametrización para que el reconocedor tenga información del contexto. De esta manera, la decodificación debería ser de mayor calidad. Además, a partir de la información que nos ofrece el reconocedor, eliminamos los fragmentos iniciales o finales etiquetados como silencios.

5.2.2. Búsqueda

A partir de los audios ya parametrizados, afrontamos la fase de búsqueda de nuestro sistema. En esta fase, nuestro objetivo es determinar en que documentos de audio puede estar contenida cada una de las *queries*.

Para ello, utilizaremos el algoritmo *Subsequence DTW*, presentado en el anterior capítulo de estado del arte. Como se ha explicado, esta variación del *Dynamic Time Warping* permite que el alineamiento de la *query* comience en cualquier punto del documento. Siguiendo esto la ecuación del algoritmo queda de la siguiente forma:

$$M(i,j) = \begin{cases} +\infty & i < 0 \\ +\infty & j < 0 \\ 0 & j = 0 \\ \min\limits_{\forall (x,y) \in S} M(i-x, j-y) + D(A_i, B_j) & j \geq 1 \end{cases} \tag{5.12}$$

donde M es la matriz de programación dinámica; S es el conjunto de movimientos permitidos, representados como pares (x, y) de incrementos horizontales y verticales; A_i, B_j son los objetos que representan las posiciones i-ésima y j-ésima de sus respectivas secuencias; y D es la función que calcula la distancia o la disimilitud entre dos objetos.

Habitualmente, los movimientos permitidos del algoritmo (S) son {(0,1), (1,1), (1,0)}. En nuestro caso, el conjunto de movimientos S es {(1,2), (1,1), (2,1)}, que garantiza (a diferencia de los movimientos típicos de DTW) que cualquier detección tendrá una longitud entre 0.5 y 2 veces el tamaño de la búsqueda. En cuanto a la función distancia utilizada, se trata del menos logaritmo del producto escalar.

Además, también se ha probado a modificar la minimización realizada por el algoritmo, de forma que se valore la relación entre la suma de distancias del camino con su longitud. El objetivo de esta modificación es que entre dos

caminos de distancia similar pero de longitudes diferentes gane aquel cuya longitud sea mayor, ya que su distancia media será menor. De esta forma, se modificaría la ecuación 5.12 por la minimización de la ecuación 5.13.

$$\min_{\forall(x,y)\in S} \frac{M(i-x, j-y) + D(A_i, B_j)}{longitud(M(x,y)) + 1} \tag{5.13}$$

Utilizando este algoritmo obtendremos una lista de posibles detecciones para cada *query*. A partir de esta lista, se eliminan aquellas detecciones que se solapen con otras con una menor distancia. De esta lista se escogen los n fragmentos con el valor de distancia más baja y se calcula el *score* que se asignará a este par (*query*, documento).

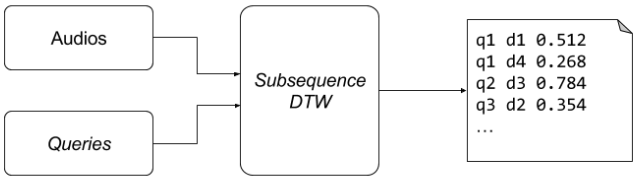

Figura 5.1: Sistema de búsqueda con *Dynamic Time Warping*.

El *score* se calcula normalizando las distancias de los n fragmentos, utilizando la media y la desviación típica (ecuación 5.14). El cambio de orden de la media y la distancia se debe a que una menor distancia debe corresponder a un *score* mayor. Para el resto de pares, se asigna un *score* por defecto.

$$score = \frac{\text{media} - \text{distancia}}{\text{desviación típica}} \tag{5.14}$$

5.2.3. Fusión de sistemas

Por último, vamos a plantear la posibilidad de fusionar diferentes sistemas para obtener mejores resultados. Para ello, aprovechamos los tres modelos de idioma del reconocedor fonético *PhnRec* para obtener tres representaciones diferentes del conjunto de audios.

La fusión se puede llevar a cabo de diferentes formas, como puede ser concatenar las tres representaciones para realizar el proceso de búsqueda u obtener por separado los resultados de los tres sistemas y fusionar, posteriormente, los *scores*. Dada la cantidad de memoria que necesitamos para la primera aproximación, en el apartado 5.3.1 se explora la segunda aproximación.

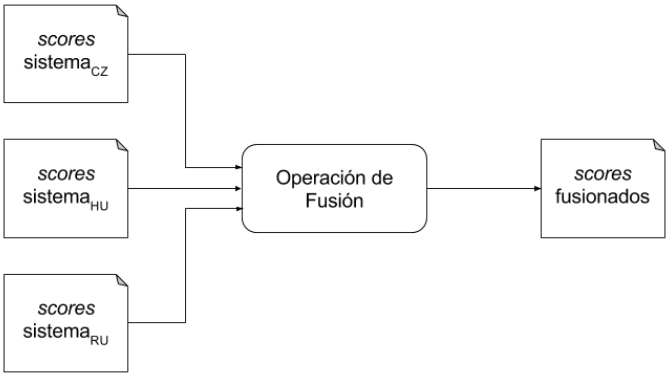

Figura 5.2: Fusión de tres sistemas de diferentes idiomas.

5.3. Experimentación y resultados

En este apartado vamos a ver en detalle los experimentos realizados para afinar los sistemas planteados y, posteriormente, veremos los resultados alcanzados en el marco de la tarea.

5.3.1. Experimentación

Antes de comenzar la experimentación, debemos establecer una serie de parámetros iniciales que, posteriormente, podremos ir variando. En el caso del modelo de idioma, Si asumimos que un modelo con un mayor número de unidades fonéticas debería ser capaz de abarcar con mayor amplitud diferentes sonidos, deberíamos utilizar el modelo de idioma húngaro (61 unidades, frente a las 45 del checo y 52 del ruso). En cuanto al número de documentos a los cuales proporcionamos un *score* diferente al por defecto, serán en principio 1000. Por último, para la distancia utilizada en el algoritmo de búsqueda, será el primer parámetro que probaremos, aunque al principio no utilizaremos la distancia media.

Función distancia

El primer parámetro a probar es la función de distancia que utiliza el algoritmo de búsqueda. Para este punto nos planteamos diferentes funciones como la distancia coseno, el logaritmo del producto escalar y la divergencia Kullback-Leibler.

La primera función, la distancia coseno, es muy utilizada con el algoritmo *Dynamic Time Warping* (ecuación 5.15).

$$cosine(u, v) = 1 - \frac{u \cdot v}{||u|| \cdot ||v||} \tag{5.15}$$

En segundo lugar, nos planteamos el producto escalar, por su similitud a la distancia coseno. El producto escalar no es una función distancia, ya que en el caso de dos vectores iguales, su valor es 1. De igual forma, y en este caso donde los vectores son probabilidades, en el peor de los casos, el valor del producto escalar es 0. Para convertirlo en una función distancia, y dado que en este caso los valores posibles están en el rango $[0, 1]$ aplicamos el logaritmo con un cambio de signo, de forma que los valores de esta función entren en el rango $[0, +\infty]$.

$$logdot(u, v) = -\log(u \cdot v) \tag{5.16}$$

Por último, también podemos utilizar la divergencia Kullback-Leibler (ecuación 5.17). Ya que los vectores con los que trabajamos indican probabilidades, puede ser adecuada para la función distancia, aunque no se puede considerar, de forma estricta como distancia, ya que no está garantizada la igualdad $d(a, b) = d(b, a)$ [29].

$$Kullback\text{-}Leibler(u, v) = \sum_i u_i \cdot \log\left(\frac{u_i}{v_i}\right) \tag{5.17}$$

En la tabla 5.2 se pueden ver los resultados para las tres funciones de búsqueda planteadas:

Función distancia	C_{nxe}	C_{nxe}^{min}	ATWV	MTWV
Coseno	1.142	0.946	-1.425	0.049
Producto escalar	1.116	0.915	-0.610	0.097
Kullback-Leibler	1.154	0.938	-0.639	0.071

Tabla 5.2: Resultados obtenidos por función de distancia.

Como se puede observar en la tabla, los mejores resultados se obtienen con el logaritmo del producto escalar, por lo que será la función distancia que utilizamos desde este punto.

Llama la atención los valores negativos del *Actual Term Weighted Value*. Esto se debe a que los 1000 documentos puntuados para cada consulta son marcados por defecto como audios que contienen la *query*. Si nos fijamos en la métrica *Maximum Term Weighted Value*, deberíamos ser capaces de

encontrar un umbral para los *scores*, de forma que el sistema únicamente marcará los documentos cuyo *score* supere este umbral. Dado que en estos primeros experimentos los *scores* pueden ser muy diferentes, introduciremos más adelante este umbral y nos guiaremos por el valor de la métrica *MTWV*.

Número de hipótesis

Para determinar si el número de documentos devueltos por el sistema es más o menos adecuado, vamos a ajustar este número. Para cada *query*, el sistema debe determinar los documentos en los que se encuentra, asignando para todo par un *score* y una decisión. Hasta ahora, nuestro sistema devolvía las 1000 hipótesis (en este caso, documentos) más probables por cada *query* y asigna un *score* por defecto al resto. Para determinar si este valor es el adecuado, hemos probado otras configuraciones (tabla 5.3).

# hipótesis	C_{nxe}	C_{nxe}^{min}	ATWV	MTWV
10	1.263	0.954	0.096	0.096
100	1.210	0.935	0.066	0.097
500	1.150	0.921	-0.230	0.097
1000	1.116	0.915	-0.610	0.097
2000	1.082	0.914	-1.334	0.096
3000	1.064	0.916	-1.997	0.093

Tabla 5.3: Resultados según el número máximo de hipótesis.

Como se puede ver en la tabla, la métrica principal va mejorando conforme aumenta el número de documentos por cada *query*. Sin embargo, si nos fijamos en el valor mínimo que puede alcanzar, apenas existen diferencias con valores superiores a 1000.

En cuanto a la métrica secundaria, empeora con valores altos del número de hipótesis. Esto es lógico, ya que el sistema indica que todas las hipótesis son positivas, mientras el resto no. Si observamos el valor máximo alcanzable, las diferencias son escasas, por lo que el problema es que debemos fijar un nuevo umbral para que el sistema determine si el *score* es lo suficientemente alto como para que realmente el documento contenga la *query*.

Como todavía estamos a mitad del proceso de experimentación y las métricas son muy similares, vamos a determinar en 2000 el número de hipótesis, debido a que es con este valor con el que obtenemos el menor C_{nxe}^{min}

Modelo de idioma

En este punto de la experimentación, nos planteamos si nuestra elección del modelo de idioma es la más adecuada. Por ello, comparamos los resultados entre los tres idiomas del reconocedor: checo, húngaro y ruso. Los resultados se muestran en la tabla 5.4.

Idioma	C_{nxe}	C_{nxe}^{min}	ATWV	MTWV
Checo	1.093	0.917	-1.311	0.106
Húngaro	1.082	0.914	-1.334	0.096
Ruso	1.037	0.890	-1.352	0.112

Tabla 5.4: Resultados obtenidos por idioma.

Como podemos comprobar en la tabla, la hipótesis que planteamos al inicio de la experimentación no es correcta, ya que los mejores resultados se obtienen con el modelo de ruso. Esto puede deberse a que, aunque está formado por un menor número de unidades fonéticas, estas son más representativas en los idiomas que forman el conjunto de datos de esta tarea.

En cualquier caso, los resultados de los diferentes idiomas son muy similares, por lo que nos planteamos, como ya se ha visto en el apartado del estado del arte, fusionar los resultados de los tres sistemas.

Fusión de sistemas

A continuación, vamos a explorar la posibilidad de fusionar los sistemas de diferentes idiomas para conseguir un mejor resultado. Para ello, debemos analizar como fusionar los resultados obtenidos por cada sistema.

Dado que cada sistema nos devuelve un *score* por cada par de documento y *query*, vamos a aplicar una función a esos *scores* obteniendo un nuevo valor. En la tabla 5.5 se pueden ver los resultados de aplicar diferentes funciones de fusión a los tres sistemas representados en la tabla 5.4. Es en este punto donde introducimos el umbral para decidir si el *score* de un par (*query*, documento) es suficientemente alto para contener realmente la consulta.

Las funciones evaluadas son media, mediana, sumatorio y máximo. Estas funciones se han escogido por los siguientes motivos:

- Media: se plantea la media porque en el caso que los diferentes sistemas asignen un valor de confianza alto o bajo, el *score* será similar. Si hay diferencias entre estos valores se ponderan de igual forma.

- Mediana: será similar en muchos casos a la media, pero en casos donde algunos sistemas valoren positivamente la hipótesis y el resto de siste-

mas le otorguen un valor negativo, esta función fuerza una especie de votación, de forma que se asigna un valor del signo más frecuente.

- Sumatorio: con esta función se refuerzan aquellas hipótesis en las que todos los sistemas dan valores del mismo signo, aumentando el *score* si son positivos y reduciéndolo si son negativos. Si existen discrepancias entre los sistemas, los *scores* con mayor valor determinaran el valor final.

- Máximo: en este caso se valora que, al menos un sistema, proporcione a una hipótesis un valor alto. Si existen diferencias entre los sistemas se opta por el máximo debido a que es posible que uno de los idiomas (y su modelo) se adecue mejor al fragmento.

Idioma	C_{nxe}	C_{nxe}^{min}	ATWV	MTWV
Media	1.140	0.884	0.111	0.118
Mediana	1.285	0.893	0.111	0.144
Sumatorio	1.279	0.871	0.136	0.154
Máximo	1.123	0.871	0.138	0.150

Tabla 5.5: Resultados de la fusión de sistemas.

Observando la tabla se puede comprobar como el mejor valor de las métricas principal y secundaria se obtiene asignando el *score* máximo entre los tres sistemas. A partir de este punto, los sistemas seguirán esta estrategia de fusión.

Normalización de distancia

Por último, hemos probado a modificar la minimización realizada en el proceso de búsqueda. Hasta ahora la minimización tenía en cuenta el sumatorio de distancias. Sin embargo, no se valora la longitud del camino, de forma que un camino corto podría tener preferencia sobre un camino más largo aunque las distancias fueran similares. Por ello, introducimos una normalización según la longitud del camino, para que se tenga en cuenta la distancia media.

	C_{nxe}	C_{nxe}^{min}	ATWV	MTWV
Distancia total	1.070	0.870	0.140	0.149
Distancia media	1.065	0.868	0.147	0.154

Tabla 5.6: Resultados según el tipo de minimización.

A partir de la tabla 5.6 se puede comprobar como los resultados mejoran si la minimización se basa en la distancia media. No obstante, dado que los resultados son similares, aprovecharemos la posibilidad de entregar ambos sistemas para comparar el rendimiento de los dos, especialmente en el conjunto de test.

5.3.2. Resultados oficiales

En este último apartado se presentan los resultados obtenidos en la tarea. En las tablas 5.7 y 5.8 se muestran los resultados de los dos sistemas presentados para los conjuntos de desarrollo y test, respectivamente. El sistema nombrado como **SDTW-avg** se presenta como primario, mientras el sistema **SDTW** se presenta como secundario.

Como se ha visto anteriormente, estos dos sistemas se diferencian en el método de minimización del algoritmo SDTW. Mientras el sistema **SDTW** utiliza la distancia directamente, el sistema **SDTW-avg** utiliza la distancia normalizada con la longitud de los fragmentos.

Sistema	C_{nxe}	C_{nxe}^{min}	ATWV	MTWV
SDTW-avg	1.0651	0.8677	0.1446	0.1543
SDTW	1.0701	0.8702	0.1404	0.1493

Tabla 5.7: Resultados obtenidos en desarrollo.

Sistema	C_{nxe}	C_{nxe}^{min}	ATWV	MTWV
SDTW-avg	1.0734	0.8751	0.1125	0.1181
SDTW	1.1879	0.9338	0.0449	0.0581

Tabla 5.8: Resultados obtenidos en test.

En las tablas anteriores podemos comprobar como, mientras en el conjunto de desarrollo ambos sistemas consiguen rendimientos similares, en el conjunto de test no ocurre igual. Mientras el sistema presentado como primario consigue unos valores cercanos a los conseguidos en el conjunto de desarrollo, el sistema secundario se comporta significativamente peor con el conjunto de test, ampliando la diferencia entre ambos sistemas.

En la tabla 5.9 se pueden ver los resultados de los dos sistemas presentados según el tipo de *query*. Como se puede comprobar, el sistema primario consigue mejores resultados en los tres tipos de forma similar.

Podemos explicar que los dos sistemas obtengan mejores resultados con el tipo de *query* T1, ya que se corresponden con búsquedas exactas, que se

Sistema	Tipo query	C_{nxe}	C_{nxe}^{min}	ATWV	MTWV
SDTW-avg	T1	0.9167	0.7870	0.1978	0.2043
	T2	1.1276	0.9052	0.0755	0.0836
	T3	1.1381	0.8959	0.0801	0.0939
SDTW	T1	1.0641	0.8675	0.1006	0.1166
	T2	1.2316	0.9524	0.0241	0.0385
	T3	1.2413	0.9526	0.0111	0.0370

Tabla 5.9: Resultados por tipo de query en test.

pueden encontrar con mayor facilidad al ser sistemas basados en *Dynamic Time Warping*. Quizás esperábamos mejores resultados de los observados en el tipo T2, ya que parte de estas búsquedas se corresponden con búsquedas en el mismo orden a nivel de palabra pero con pequeñas diferencias al inicio o al final.

Finalmente, en la tabla 5.10 podemos ver los resultados de los sistemas principales de los equipos participantes de la tarea ordenados por la métrica principal, C_{nxe}.

Equipo	C_{nxe}	C_{nxe}^{min}	$ATWV$	MTWV
NNI	0.7610	0.7472	0.2703	0.2738
SPL-IT-UC	0.7866	0.7809	0.2064	0.2162
BUT	0.8452	0.8263	0.1513	0.1557
GTM-UVIGO	0.9185	0.9046	0.0403	0.0430
IIT-B	0.9536	0.9364	0.0254	0.0421
TUKE	0.9714	0.9530	0.0029	0.0221
SPEED	1.0379	0.9963	-0.0762	0.0000
CUNY	1.0674	0.9853	-4.0205	0.0006
ELiRF	1.0734	0.8751	0.1125	0.1181
NTU	2.0067	0.9972	-1.0828	0.0000

Tabla 5.10: Resultados oficiales de QUESST 2015.

A partir de esta tabla podemos extraer información muy interesante. Nuestro principal sistema ocupa la novena posición por la métrica principal. Sin embargo, si tenemos en cuenta la métrica C_{nxe}^{min} —que como hemos explicado en el punto 5.1.1, indica el valor que puede alcanzar la métrica principal— ocupa la cuarta posición. También podemos comprobar como en las dos métricas de *Term Weighted Value* alcanzamos también la cuarta posición.

Esta diferencia de posiciones entre la métrica principal y la métrica secundaria se debe a la falta de ajuste de los *scores* que se podría haber realizado a partir del conjunto de desarrollo. Si aplicamos la transformación lineal utilizada en el calculo de la métrica C_{nxe}^{min} para el conjunto de desarrollo, los valores de la métrica principal deberían ser más similares.

Por último, vamos a comparar nuestro sistema con los tres que han alcanzado mejores resultados.

El sistema del equipo NNI [30] es, a su vez, una fusión de diferentes sistemas de tres grupos (NWPU, NTU y I^2R). Entre todos los sistemas, se utilizan diferentes características como posteriorgramas o *Bottleneck features*. Además, aunque la mayoría de estos sistemas están basados en el algoritmo de *DTW*, también se utilizan sistemas de búsqueda simbólica (*Symbolic Search*).

En el caso del sistema del grupo SPL-IT-UC [31], se utiliza una parametrización similar a la que planteamos, utilizando el mismo reconocedor fonético. Sin embargo, también se añaden dos reconocedores propios para los idiomas de inglés y portugués. Para la fase de búsqueda también se utiliza el algoritmo *Dynamic Time Warping*, aunque se introducen varias modificaciones intentando así cubrir las consultas más complejas.

Las diferencias con el tercer sistema, del grupo BUT [32], se limitan a la parametrización del audio, ya que también emplean el algoritmo *Subsequence DTW*. En este caso, se obtienen *Bottleneck features* con modelos de los idiomas checo, portugués, ruso y español.

Capítulo 6

Conclusiones y trabajo futuro

Para finalizar el presente trabajo, vamos a comprobar hasta que punto hemos alcanzado los objetivos que nos habíamos marcado anteriormente y también vamos a detallar en qué podemos continuar trabajando en el futuro.

6.1. Conclusiones

En el apartado 1.3 planteamos dos objetivos para el presente trabajo. El primero era el desarrollo de un sistema capaz de descubrir palabras dentro de un conjunto de documentos de audio. Como se puede comprobar en el capítulo 3, se han planteado diferentes aproximaciones para este problema, obteniendo diferentes resultados.

En primer lugar, planteamos una aproximación basada en el algoritmo de programación dinámica *Dynamic Time Warping* que, a partir de *frames* de audios similares, intenta encontrar secuencias que correspondan a una misma palabra. Como se detalla en la experimentación, esta aproximación conlleva un coste temporal elevado, teniendo en cuenta la cantidad de fragmentos a analizar.

La segunda aproximación se basa en la decodificación fonética. A partir de esta decodificación, se buscan secuencias de fonemas iguales entre todos los audios. Esta aproximación consigue buenos resultados en determinadas métricas, aunque la calidad tanto de los emparejamientos como de los *clusters* formados son mejorables.

Con la tercera aproximación el objetivo era comprobar si es interesante flexibilizar la búsqueda de secuencias de fonemas, de forma que se admitiese ciertas variaciones. Los resultados han sido los esperados: mientras las métricas que indican la precisión tienden a empeorar, los valores de métricas de *recall*, mejoran.

Nuestro segundo objetivo consistía en construir un sistema de búsqueda de documentos de audio a partir de consultas también en forma de audio. En el capítulo 5 se ha presentado un sistema capaz de realizar esta tarea, tras realizar diferentes experimentaciones con el objetivo de mejorar este sistema.

En esta tarea, la aproximación planteada es utilizar el algoritmo *Subsequence DTW* para detectar si en un determinado documento de audio existe un patrón similar a la consulta. Esta aproximación se va refinando a partir de las experimentaciones.

Además, utilizamos una técnica de fusión de sistemas que permite que los resultados finales estén basados en modelos de tres idiomas diferentes.

6.2. Trabajo futuro

A partir de las conclusiones extraídas en el apartado anterior, vemos como determinados aspectos del trabajo desarrollado pueden ser mejorados o también existen otras aproximaciones que se pueden explorar. A continuación, se enumeran posibles trabajos a futuro en estos puntos:

- Seguir trabajando en el algoritmo planteado para la tarea de *Spoken Term Detection*.
- Aplicar en la tarea de *Spoken Term Detection* técnicas aplicadas posteriormente en *Query by Example*, como la fusión de sistemas.
- Ampliar el sistema presentado de *Query by Example* para tratar las consultas complejas. En el sistema planteado no se tratan, por ejemplo, consultas formadas por varias palabras cuyo orden sea diferente.

Apéndice A

Publicaciones relacionadas

A continuación se enumeran las diferentes publicaciones relacionadas con el trabajo presentado:

- Sergio Laguna, Marcos Calvo, Lluís-F. Hurtado, and Emilio Sanchis. ELiRF at Mediaeval 2015: Query by Example Search on Speech Task (QUESST). In *Working Notes Proceedings of the MediaEval 2015 Workshop, Sept. 14-15, 2015, Wurzen, Germany*, 2015.

Además, actualmente se está trabajando en otras tareas:

- *Zero Cost Speech Recognition Task* (MediaEval 2016)
  El objetivo de esta tarea es desarrollar un sistema de reconocimiento del habla partiendo de un conjunto de datos públicos, es decir, sin que sea necesario comprar conjuntos de datos.
- *ALBAYZIN 2016 Search on Speech Evaluation* (IberSpeech 2016)
  Esta tarea presenta dos subtareas similares a las presentadas en esta memoria: *Spoken Term Detection* y *Query by Example*.

Bibliografía

[1] Dredze, Mark, Aren Jansen, Glen Coppersmith y Ken Church: *NLP on Spoken Documents without ASR*. En *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, páginas 460–470. Association for Computational Linguistics, 2010.

[2] Zhu, Xiaodan, Gerald Penn y Frank Rudzicz: *Summarizing multiple spoken documents: finding evidence from untranscribed audio*. En *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, páginas 549–557. Association for Computational Linguistics, 2009.

[3] Park, Alex y James R Glass: *Towards unsupervised pattern discovery in speech*. En *Automatic Speech Recognition and Understanding, 2005 IEEE Workshop on*, páginas 53–58. IEEE, 2005.

[4] Lin, Yaw Ling, Tao Jiang y Kun Mao Chao: *Efficient algorithms for locating the length-constrained heaviest segments with applications to biomolecular sequence analysis*. Journal of Computer and System Sciences, 65(3):570–586, 2002.

[5] Zhang, Yaodong y James R Glass: *Towards multi-speaker unsupervised speech pattern discovery*. En *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, páginas 4366–4369. IEEE, 2010.

[6] Muscariello, Armando, Guillaume Gravier y Frédéric Bimbot: *Unsupervised Motif Acquisition in Speech via Seeded Discovery and Template Matching Combination*. Audio, Speech, and Language Processing, IEEE Transactions on, 20(7):2031–2044, 2012.

[7] Jansen, Aren y Benjamin Van Durme: *Efficient Spoken Term Discovery Using Randomized Algorithms*. En *Automatic Speech Recognition*

and Understanding (ASRU), 2011 IEEE Workshop on, páginas 401–406. IEEE, 2011.

[8] Versteegh, Maarten, Roland Thiolliere, Thomas Schatz, Xuan Nga Cao, Xavier Anguera, Aren Jansen y Emmanuel Dupoux: *The Zero Resource Speech Challenge 2015*. En *Proc. of INTERSPEECH*, 2015.

[9] Pitt, Mark A., Laura Dilley, Keith Johnson, Scott Kiesling, William Raymond, Elizabeth Hume y Eric Fosler-Lussier: *Buckeye corpus of conversational speech (2nd release)*. Columbus, OH: Department of Psychology, Ohio State University, 2007. `http://www.buckeyecorpus.osu.edu/`.

[10] De Vries, Nic J., Marelie H. Davel, Jaco Badenhorst, Willem D. Basson, Febe De Wet, Etienne Barnard y Alta De Waal: *A smartphone-based ASR data collection tool for under-resourced languages*. Speech communication, 56:119–131, 2014.

[11] Schwarz, P.: *Phoneme Recognition based on Long Temporal Context, PhD Thesis*. Brno University of Technology, 2009.

[12] Garofolo, John S, Lori F Lamel, William M Fisher, Jonathan G Fiscus, David S Pallett, Nancy L Dahlgren y Victor Zue: *TIMIT acoustic-phonetic continuous speech corpus*. Linguistic data consortium, Philadelphia, 33, 1993.

[13] Heuvel, Henk van den, Valery Galounov y Herbert Tropf: *The SpeechDat (E) project: Creating speech databases for Eastern European languages*. 1998.

[14] Apache Software Foundation: *Apache Lucene*. `http://lucene.apache.org/`.

[15] Johnson, Mark, Thomas L Griffiths y Sharon Goldwater: *Adaptor grammars: A framework for specifying compositional nonparametric Bayesian models*. En *Advances in neural information processing systems*, páginas 641–648, 2006.

[16] Räsänen, Okko, Gabriel Doyle y Michael C Frank: *Unsupervised word discovery from speech using automatic segmentation into syllable-like units*. En *Proc. Interspeech*, 2015.

[17] Lyzinski, Vince, Gregory Sell y Aren Jansen: *An evaluation of graph clustering methods for unsupervised term discovery*. En *Proceedings of Interspeech*, 2015.

[18] *Information Retrieval for Music and Motion*, capítulo Dynamic Time Warping, páginas 69–84. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, ISBN 978-3-540-74048-3.

[19] Anguera, Xavier y Miquel Ferrarons: *Memory efficient subsequence DTW for Query-by-Example spoken term detection.* En *2013 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 2013.

[20] Muscariello, Armando, Guillaume Gravier y Frédéric Bimbot: *Audio keyword extraction by unsupervised word discovery.* En *INTERSPEECH 2009: 10th Annual Conference of the International Speech Communication Association*, 2009.

[21] Muscariello, Armando, Guillaume Gravier y Frédéric Bimbot: *Zero-resource audio-only spoken term detection based on a combination of template matching techniques.* En *INTERSPEECH 2011: 12th Annual Conference of the International Speech Communication Association*, 2011.

[22] Hazen, Timothy J, Wade Shen y Christopher White: *Query-by-Example Spoken Term Detection Using Phonetic Posteriorgram Templates.* En *IEEE Workshop on Automatic Speech Recognition & Understanding, 2009. ASRU 2009.*, páginas 421–426. IEEE, 2009.

[23] Zhang, Yaodong y James R Glass: *Unsupervised Spoken Keyword Spotting via Segmental DTW on Gaussian Posteriorgrams.* En *IEEE Workshop on Automatic Speech Recognition & Understanding, 2009. ASRU 2009.*, páginas 398–403. IEEE, 2009.

[24] Abad, Alberto, Luis Javier Rodriguez-Fuentes, Mikel Penagarikano, Amparo Varona y Germán Bordel: *On the Calibration and Fusion of Heterogeneous Spoken Term Detection Systems.* En *INTERSPEECH*, páginas 20–24, 2013.

[25] Szoke, Igor, Luis J. Rodriguez-Fuentes, Andi Buzo, Xavier Anguera, Florian Metze, Jorge Proenca, Martin Lojka y X. Xiao: *Query by Example Search on Speech at Mediaeval 2015.* En *Working Notes Proceedings of the MediaEval 2015 Workshop, Sept. 14-15, 2015, Wurzen, Germany*, 2015.

[26] Rodriguez-Fuentes, Luis J. y Mikel Penagarikano: *MediaEval 2013 Spoken Web Search Task: System Performance Measures.* n. TR-2013-1, Department of Electricity and Electronics, University of the Basque Country, 2013.

[27] Anguera, Xavier, Luis Javier Rodriguez-Fuentes, Igor Szöke, Andi Buzo y Florian Metze: *Query by Example Search on Speech at Mediaeval 2014.* En *MediaEval*, 2014.

[28] Laguna, Sergio, Marcos Calvo, Lluís F. Hurtado y Emilio Sanchis: *ELiRF at MediaEval 2015: Query by Example Search on Speech Task (QUESST).* En *Working Notes Proceedings of the MediaEval 2015 Workshop, Sept. 14-15, 2015, Wurzen, Germany*, 2015.

[29] *The Elements of Complex Analysis*, capítulo Metric Spaces, página 20. New Age International, 1992, ISBN 978-81-224-0399-2.

[30] Hou, Jingyong, Cheung Chi Leung Van Tung Pham, Lei Wang, Haihua Xu, Hang Lv, Lei Xie, Zhonghua Fu, Chongjia Ni, Xiong Xiao, Hongjie Chen *y cols.*: *The NNI Query-by-Example System for MediaEval 2015.* 2015.

[31] Proença, Jorge, Luis Castela y Fernando Perdigão: *The SPL-IT-UC Query by Example Search on Speech system for MediaEval 2015.* 2015.

[32] Skácel, Miroslav y Igor Szöke: *BUT QUESST 2015 System Description.* 2015.