



UNIVERSIDAD DE BUENOS AIRES
FACULTAD DE CIENCIAS EXACTAS Y NATURALES
DEPARTAMENTO DE COMPUTACIÓN

Recolección online de grabaciones para el estudio de las variantes argentinas del español

Tesis de Licenciatura en
Ciencias de la Computación

Fernando Bugni

Director: Agustín Gravano

Codirector: Miguel Martínez Soler

Buenos Aires, 2014

Recolección online de grabaciones para el estudio de las variantes argentinas del español

El uso de la lengua siempre ha caracterizado a las personas que la utilizan. La forma como nos comunicamos no sólo posee la información del mensaje a transmitir, sino que también posee características del hablante. Estas características pueden describir al hablante de distintas formas. Algunas de ellas pueden ser: su cultura, su nivel socioeconómico y su región, entre otras. En particular, Argentina no es la excepción. Nuestro país posee una fuerte componente dialéctica en su habla. Esto quiere decir que podemos saber de qué lugar proviene el hablante analizando su tonada. Hay varias regiones con esta característica definidas en el país. Nos enfocaremos en distinguir diferencias entre las regiones de Córdoba y Buenos Aires. En el presente trabajo desarrollamos un sistema de grabación a través de Internet que nos permitió recolectar grabaciones de hablantes provenientes de ambos grupos. Con este sistema de grabación, diseñamos un experimento que nos ayudó a comparar el habla de cada grupo haciendo hincapié en sus diferencias. Analizamos las dificultades técnicas que surgieron y cómo impactaron en el estudio final. Utilizando estos datos, analizamos efectivamente cuáles son las características más predominantes y cómo repercuten para una buena clasificación. Extrayendo estos atributos, utilizamos algoritmos de Machine Learning para la clasificación de hablantes en los dos grupos.

Agradecimientos

¡Uff! ¡Todo el tiempo que pasó desde que comencé la carrera! Mejor ni acordarse. Pero aunque haya pasado tanto tiempo lo bueno fue toda la gente copada que conocí...

Les voy a agradecer siempre a todos los amigos nuevos que conocí gracias a la facu. Tantas horas estudiando y divirtiéndose también hace que haya sido buenísimo pasarla juntos. Algunos de ellos son: *Alex Valencia, Pablo Echebarriga, Gonza Raposo, Checho González, Agustín Olmedo* entre otros... A todos los amigos de la lista *Gente de la Facu* que estudiamos juntos en varias oportunidades. Entre ellos *Pablo Laciana* y *Leo Rodriguez* que inclusive fuimos a ver un par de veces al *Matador de Victoria*. Seguramente me olvide de algunos que ahora no recuerdo pero bueno... las hojas son finitas :)

A mis compañeros y amigos del trabajo, que me dieron la oportunidad de trabajar con ellos y compartir muchísimas tardes juntos. Estos son: *Mariano Gonzalez, Francisco Tarulla, Lucas Bali, Leonardo Kuntscher, Graciela Defeo, Alejandro Aquesta*. Sin su buena onda y ayuda no hubiera sido lo mismo terminar la carrera.

A mi grupo de amigos, que la mayoría también van a la facu. Ellos son: *Midra, Dani, Mich, El cansado, Pablox, Nahuel, Maru, Luis, Vicky, El Punga*. Con la buena onda de ellos y su ayuda cuando uno andaba medio mal no hubiera sido lo mismo. Voy a agradecer siempre lo increíble que son cada uno de ellos como persona.

¡A mis directores de tesis, que me han soportado todo este tiempo! A *Agustín* que es un genio explicando y le pone una increíble pasión a lo que hace. A *Miguel* que me ha ayudado todo este tiempo realizando este trabajo. También a la gente del *Laboratorio de Investigaciones Sensoriales* que nos prestaron su diccionario para realizar el experimento. ¡Gracias por todo! :P

Y por último, a mi familia. Mi vieja, *Camila*, que sin su apoyo y ganas no hubiera podido hacer nada en mi vida. Mi viejo, *Quique*, que me dio todo su cariño y me cuida desde el cielo. Mi hermano que es el mejor compañero que tengo de mi vida. Y a mi novia, *Emilia*, que me bancó todo este tiempo realizando esta tesis. *¡Te amo!*

A mi viejo, que me ayuda desde el cielo.

A mi vieja, que gracias a ella soy lo que soy.

Y a mi hermano, que es el compañero de mi vida.

Índice general

1. Introducción	1
2. Diseño del experimento	5
2.1. Elección de las frases	5
2.1.1. Frases utilizando esquema AMPER	6
2.1.2. Frases comunes	7
2.1.3. Combinando los dos tipos de frases utilizando trazas	11
3. Sistema de grabación online	12
3.1. Recolección de datos	12
3.2. Grabación a través del browser	14
3.2.1. Requerimientos	15
3.3. Varias grabaciones por frase	15
3.4. Sistema de administración	16
3.4.1. Etiquetado de audio	16
3.5. Backups automáticos	17
3.6. Análisis del volumen	17
4. Extracción de información	18
4.1. Alineación forzada	18
4.1.1. Prosodylab Aligner	18
4.2. Extracción de atributos	21
4.2.1. Atributos temporales	22
4.2.2. Atributos acústicos	25
4.2.3. Atributos desconocidos	27
4.2.4. Nomenclatura utilizada	27
5. Datos obtenidos	29

5.1.	Evaluación manual de las grabaciones	29
5.2.	Alineación forzada	30
5.3.	Corrección de errores	31
6.	Análisis	33
6.1.	Baseline	33
6.2.	Clasificadores	34
6.3.	Tests estadísticos	35
6.3.1.	Test de Wilcoxon	35
6.3.2.	Análisis Shapiro-Wilk Test	35
6.3.3.	Student t Test	36
6.4.	Modelos de tests	36
6.4.1.	Grupos de hablantes	37
6.4.2.	Un hablante para test y los demás para train	45
6.4.3.	Promediando los atributos de cada hablante	48
6.4.4.	Promediando los atributos de cada hablante sólo si es desco- necido	51
6.5.	Selección de atributos de forma automática	53
6.6.	Combinando clases de atributos	54
7.	Conclusiones y Trabajo Futuro	56

Capítulo 1

Introducción

El uso de la lengua siempre ha caracterizado a las personas que la utilizan. La forma en que nos comunicamos no sólo posee la información del mensaje a transmitir, sino también características del hablante. Estudiar estas características del habla nos permite conocer mejor la cultura de las personas. Nos permite, por ejemplo, identificar a los hablantes para saber el lugar donde pertenecen.

Identificar y extraer características del habla son tareas muy difíciles de realizar. No sólo se deben obtener muestras muy variadas de muchos hablantes en distintas regiones, sino que también hay que prestarle importante atención a su edad, su sexo, su situación económica, etc. Realizar un estudio de estas características es muy complejo y sobre todo, costoso. Además de estudiar cada grupo se deben utilizar muchos recursos: por ejemplo, se debe utilizar soporte para grabar en buena calidad las muestras, varios viajes para buscar los diferentes hablantes, analizar cada uno de los audios de manera individual, entre otras cosas.

El objetivo de esta tesis es realizar un sistema que pueda facilitar la resolución de estos problemas. Vamos a enfrentar cada uno de ellos e intentar resolverlos de forma computacional. De los problemas descritos el principal radica en obtener cada grabación. Si los grupos se encuentran muy alejados esto puede ser muy costoso por los viajes. También estas grabaciones deben ser de calidad aceptable como para realizar el estudio en cuestión. Se podría utilizar el teléfono para algunos experimentos pero hay que tener en cuenta que este posee calidad muy baja. De hecho, se utiliza en algunos experimentos donde esta característica no es un inconveniente.

Es por esto que se desarrolló un sistema de grabación basado en Internet como herramienta para obtener muestras. De esta forma, se pueden realizar varias grabaciones sin necesidad de viajar a cada lugar. Es cierto que no todos los lugares poseen acceso a Internet y, si se realizara un experimento de estas características en lugares que no posean conexión, este sistema no sería útil. De cualquier forma, pensamos que su utilización soluciona muchos inconvenientes. Otra característica radica en que se puede manejar la calidad de la grabación. Utilizando distintas tecnologías a través de esta red se puede configurar la calidad para que sea lo más precisa posible para el experimento. El sistema desarrollado va a mejorar la forma de recolectar muestras



Figura 1.1: Dialectos del idioma español en Argentina

y lo utilizamos en un experimento para corroborar las ventajas y desventajas del mismo.

El objetivo del sistema es obtener muestras de habla para su posterior análisis o utilización en sistemas de procesamiento de voz. El experimento que tomamos como caso de estudio es medir las diferencias en el habla entre Córdoba y Buenos Aires. El primero de estos grupos se encuentra en la zona central de nuestro país mientras que el segundo cerca del Río de la Plata, como se puede observar en la Figura 1.1. En la literatura existen estudios que explican estas diferencias, por ejemplo *El español en la Argentina* [María Beatriz Fontanella de Weinberg, 2000] de Beatriz Fontanella de Weinberg y *Español en la Argentina* [Elena Vidal de Battini, 1964] de Elena Vidal de Battini.

Fontanella de Weinberg recopila varios trabajos de colegas que analizan el español de cada región de Argentina. Cada región se describe en un capítulo distinto y entre ellas se encuentra uno para Buenos Aires y otro para Córdoba. En la descripción de estos capítulos las diferencias hacen hincapié en los sonidos más suaves y cortos

de la /r/, el cambio de sonidos de /y/ y /ll/ a /j/ y en la aspiración de la /s/ al terminar una palabra. También describe el estiramiento de la sílaba anterior a la acentuada en cada palabra como distintivo del acento cordobés. Por su parte, Vidal de Battini analiza región por región el uso de los fonemas importantes. Destaca la diferencia entre las dos regiones en el uso de la /r/, de la /s/ y de la /ll/. También referencia a la pronunciación de la /s/.

Extrayendo el análisis de estos libros se puede definir las reglas que describen a cada grupo. Las reglas son:

- **Regla 1: Los hablantes de Córdoba estiran la sílaba anterior a la acentuada mientras los de Buenos Aires no lo hacen.** Cada palabra posee una sílaba con su acento primario. Para cumplir esta regla se debe estirar la sílaba anterior a esta. Si la sílaba acentuada es la primera de la palabra, entonces no se estira. Ejemplo: ‘Espectacular’ posee su sílaba acentuada en ‘-lar’. La sílaba anterior, o sea ‘-cu-’ se alarga solamente para hablantes de Córdoba.
- **Regla 2: Los hablantes de Córdoba aspiran y elisionan la /s/ al finalizar una palabra. Esto no sucede en Buenos Aires.** Para las palabras terminadas en /s/ se acorta la duración de éste último fonema en hablantes de Córdoba. Ejemplo: ‘Pájaros’ posee el fonema /s/ al final. Utilizando la dialéctica de Córdoba, la /s/ final sería más suave que una de Buenos Aires.
- **Regla 3: Para hablantes de Córdoba, la /s/ antes de la /c/ o /t/ suenan más suaves que para hablantes de Buenos Aires.** El fonema /s/, que precede a /c/ o /t/, suena más suave en cordobeces que en porteños. Ejemplo: ‘Moscaén la variante de Córdoba posee el fonema /s/ más suave que en Buenos Aires.
- **Regla 4: La ‘cántes de la ‘t’ se pronuncia con menor frecuencia para hablantes de Córdoba que para hablantes de Buenos Aires.** El fonema /c/, que precede a /t/, no se debe pronunciar en el dialecto cordobés. Ejemplo: ‘Doctorño debe sonar el fonema /c/.
- **Regla 5: Para hablantes cordobeces la ‘y’ y ‘ll’ se pasa a ‘i’. No sucede esto para Buenos Aires.** Palabras con el fonema /y/ o /ll/ se pronuncian /j/. Ejemplo: ‘lluvia’ se debe pronunciar utilizando el fonema /j/.
- **Regla 6: En hablantes cordobeces la /r/ no vibra mientras que en Buenos Aires pasa lo contrario.** Palabras con el fonema /r/ deben ser suaves y no vibrar en el dialecto cordobés. Ejemplo: ‘Espárragoén su fonema /r/ debe ser suave en comparación con Buenos Aires.

Normalmente estas reglas se producen en el habla espontánea y raramente en habla leída. Algunas pueden agudizarse si se encuentran en lugares económicamente más vulnerables, pero en cualquier ambiente se cumplen.

En el próximo capítulo describiremos el diseño del experimento. Este tiene como objetivo reconocer las diferencias planteadas con las reglas mediante la grabación de frases. Estas frases fueron grabadas tanto por hablantes de Córdoba como de Buenos Aires. También describiremos cuáles frases utilizamos y el medio empleado para grabar.

Capítulo 2

Diseño del experimento

Utilizando los estudios de [María Beatriz Fontanella de Weinberg, 2000] y [Elena Vidal de Battini, pudimos obtener las reglas definidas en el capítulo anterior. Recordemos que éstas describen la diferencia entre los dos grupos. En este capítulo proponemos realizar un experimento para extraer la información fonética de los mismos. La idea será realizar una serie de actividades donde el hablante sea grabado y que esas actividades hagan hincapié en estas diferencias descriptas. A continuación vamos a describir el experimento en más detalle.

2.1. Elección de las frases

El acento se potencia cuando se realiza habla espontánea. Utilizando este concepto intentamos que el hablante diga las frases de la forma más natural posible. Es por ello que elegimos como actividad pronunciar frases popularmente conocidas. Si el hablante conoce la frase y la utiliza con frecuencia entonces es más fácil que su pronunciación sea espontánea. Con esta idea vamos a cubrir las reglas 2 al 6.

Recordemos que en el capítulo anterior la regla 1 nos decía que había una diferencia en la duración de la sílaba previa a la acentuada: para hablantes de Córdoba esta duración es más larga que para hablantes de Buenos Aires. La sílaba acentuada varía según que tipo de palabra se refiere. No es lo mismo utilizar una palabra aguda que una esdrújula en esta regla. Para cubrir esta regla utilizamos un esquema de frases con una estructura fija pero que varía sus palabras según su entonación. Este esquema se llama AMPER [Gurlekian et al., 2009] y lo veremos más en detalle en la sección 2.1.1.

Entonces los esquemas van a ser:

- **Frases utilizando esquema AMPER que cubre cada tipo de acentuación:** Estas corresponden a la regla 1 que hace énfasis en la longitud de la sílaba anterior a la acentuada. Utilizamos este esquema para cubrir todo tipo de acentuación.

- **Frases comunes que tratan de cubrir la espontaneidad:** Estas frases van a cubrir las reglas 2 a 6. Estas tienen que ver con la duración y la pronunciación de distintos fonemas. Utilizar frases comunes favorece la espontaneidad.

A continuación vemos las reglas en sus dos conjuntos.

2.1.1. Frases utilizando esquema AMPER

Utilizamos este esquema para analizar todas las variantes posibles de la regla 1. Recordemos que esa regla nos dice que hay que estirar la sílaba anterior a la acentuada. Esta regla define comúnmente la tonada cordobesa y puede aparecer de varias formas según su acentuación.

Para tomar este esquema nos basamos en el trabajo de variabilidad rítmica en dos corpus [Gurlekian et al., 2009] que utilizan un esquema similar. Este trabajo estudió los acentos del español argentino utilizando un esquema de frase fija y intercambiando palabras para analizar todos sus casos. En nuestro trabajo tenemos un problema similar por ello lo tomamos como referencia.

Para el esquema AMPER se fija una estructura para la frase y se va cambiando las palabras que utiliza. El esquema AMPER utilizado en este trabajo es:

Sujeto + “salió” + Adjetivo

- Objeto puede ser “*El canapé*”, “*El repollo*”, “*El espárrago*”.
- Adjetivo puede ser “*espectacular*”, “*delicioso*”, “*riquísimo*”.

Utilizamos estas palabras ya que cubren los tres tipos de acentuación: aguda, grave y esdrújula.

Por ejemplo: “*El canapé salió delicioso*”. Canapé tiene acento en la última sílaba, es una palabra aguda, mientras que delicioso es grave. En este ejemplo podemos analizar la sílaba anterior a la acentuada de los dos grupos estudiados, Córdoba y Buenos Aires. Armamos las combinaciones para obtener muchas variantes de dónde se encuentra el acento. De esta forma estudiamos en detalle la regla 1. Todas las combinaciones se pueden ver en el tabla 2.1.

Frase			Aguda	Grave	Esdrújula
<i>El canapé</i>	<i>salió</i>	<i>espectacular</i>	espectacular, canapé		
<i>El canapé</i>	<i>salió</i>	<i>delicioso</i>	canapé	delicioso	
<i>El canapé</i>	<i>salió</i>	<i>riquísimo</i>	canapé		riquísimo
<i>El repollo</i>	<i>salió</i>	<i>espectacular</i>	espectacular	repollo	
<i>El repollo</i>	<i>salió</i>	<i>delicioso</i>		repollo, delicioso	
<i>El repollo</i>	<i>salió</i>	<i>riquísimo</i>		repollo	riquísimo
<i>El espárrago</i>	<i>salió</i>	<i>espectacular</i>	espectacular		
<i>El espárrago</i>	<i>salió</i>	<i>delicioso</i>		delicioso	
<i>El espárrago</i>	<i>salió</i>	<i>riquísimo</i>			riquísimo

Tabla 2.1: Frases AMPER

2.1.2. Frases comunes

Como afirmamos antes, se utilizaron frases comunes para poder obtener los acentos de cada grupo de forma natural. Se pensó que si se graba una frase popular, el hablante al estar acostumbrado a decirla no iba a poder evitar impregnarle su acento. Todas las frases utilizadas se pueden ver en la tabla 2.2.

Frase	2 - Aspiración y elisión de /s/	3 - La 'sántes de la 'có 'tño suenan	4 - La 'cántes de la 'tño suenan	5 - La 'y'y 'll'se pasa a í	6 - La 'rño debe sonar
No hay dos sin tres'	dos, tres				
'Más difícil que encontrar una aguja en un pajar'	más				
'Más perdido que turco en la neblina'	más				
No le busques la quinta pata al gato'	busques	busques			
'Se te escapó la tortuga'		escapó			
'Todos los caminos conducen a Roma'	todos, los, caminos				
No hay mal que dure cien anos'	años				
'Siempre que llovió paro'				llovió	
'Cría cuervos que te sacaran los ojos'	cuervos, los, ojos				
'La tercera es la vencida'	es				
'Calavera no chilla'				chilla	
'La gota que rebalsó el vaso'					rebasó
'La suegra y el doctor cuanto más lejos mejor'	más, lejos	doctor			
Á la mujer picaresca cualquiera la pesca'		picaresca			
'Quien siembra vientos recoge tempestades'	vientos				recoge
'La arquitectura es el arte de organizar el espacio'	es		arquitectura		
El amor actúa con el corazón y no con la cabeza'			actúa		
No dudes actúa'			actúa		
'Perro que ladra no muerde'					perro
'La musica es sinónimo de libertad, de tocar lo que quieras y como quieras'	es, quieras				
'La belleza que atrae rara vez coincide con la belleza que enamora'				belleza	
No esta mal ser bella lo que esta mal es la obligación de serlo'				bella	
'La batalla más difícil la tengo todos los días conmigo mismo'	más			batalla	
El que no llora no mama'				llora	
En la pelea se conoce al soldado solo en la victoria se conoce al caballero'			victoria	caballero	
'La lectura es a la mente lo que el ejercicio al cuerpo'	es		lectura		
El pez por la boca muere'	pez				
En boca cerrada no entran moscas'	moscas	moscas			cerrada
'Más vale pájaro en mano que cien volando'	más				
'Río revuelto ganancia de pescadores'	pescadores	pescadores			río, revuelto
No hay que pedirle peras al olmo'	peras				

Tabla 2.2: Frases conocidas

Algo interesante es que una misma frase puede extraer atributos para varias reglas. Por ejemplo: la frase ‘*En la pelea se conoce al soldado, sólo en la victoria se conoce al caballero*’ extrae atributos para las reglas 4 y 5. La palabra ‘*victoria*’ cubre la regla 4 que nos propone medir la duración de la /c/ antes de la /t/. Sucede igual con la palabra ‘*caballero*’ para la regla 5: el fonema /ll/ se pasa a /i/ cambiando su duración y sonido. En la Tabla 2.2 podemos ver que la cantidad de frases utilizadas con respecto a sus reglas es despareja. Hay más frases para la regla 2 que para las demás. Más adelante veremos como impacta esto en las frases que vamos a pedir grabar.

Orden de las frases

Ya definimos cuáles van a ser las frases, ahora debemos definir qué frases y en qué orden se deben decir durante el experimento. Diremos que una frase cubre una determinada regla si esta la satisface. Sucede que el orden que utilicemos va a ser crucial para obtener muestras: no es lo mismo empezar por una frase que sólo cubre una sola regla que varias. Si elegimos primero las frases que cubren varias reglas a la vez, en un sólo audio podremos obtener más cubrimiento de reglas.

¿Por qué quisiéramos cubrir más reglas en una misma frase? Más adelante veremos que una frase grabada que cubre una regla aportará información sobre el hablante de esa regla en particular. Si una frase cubre varias reglas estaríamos obteniendo más información sólo utilizando una grabación. Por eso es importante maximizar el cubrimiento de las frases.

El orden de las frases sigue el siguiente algoritmo:

```

1  OrdenDeFrasesConocidas:
2  Input: Frases (conj. String)
3  Output: listaFrases (lista de String)
4  listaFrases = {}
5  DicPct <- Diccionario de porcentajes de cada regla
6  Mientras Frases != {}:
7      regla <- ObtenerReglaConMenorPorcentaje(DicPct)
8      frase <- Frases.SacarLaMasPonderada(regla)
9      listaFrases.agregar(frase)
10     RecalcularPorcentajes(DicPct)
11 Devolver listaFrases

```

La idea del algoritmo es la siguiente: vamos a contabilizar la cantidad de muestras por cada regla. Esto se realiza calculando, para cada regla, la cantidad de frases que ya utilizamos cubriendo esa regla; sobre la cantidad total de frases que cubren esa regla. Entonces tendremos para cada regla su porcentaje de cubrimiento. Esto lo guardaremos en *DicPct*.

Mientras haya frases sin ser seleccionadas para grabar, elegimos la regla que menos está cubierta. O sea, elegiremos como próxima regla a cubrir la que tenga menor porcentaje de cubrimiento. Esto se realiza en la función *ObtenerReglaConMenorPorcentaje* utilizando como parámetro *DicPct*.

Tenemos definida la próxima regla a cubrir. Debemos elegir la frase que cubra esa regla. Recordemos que hay muchas frases que cubren una determinada regla. Para definir cuál frase elegir vamos a optar por la que cubra mayor cantidad de reglas. Si hay varias frases en esta situación, elegimos una de estas al azar. Entonces no sólo aumentamos la cantidad de frases de la regla menos cubierta, sino que con esa frase cubrimos otras reglas. Esto se realiza en la función *ObtenerLaMásPonderada*.

Para terminar, agregamos la frase elegida y recalculamos los porcentajes de cada regla en las líneas 9 y 10.

Esta idea es importante ya que llevamos al máximo la cantidad de información en cada frase y al hablante le hacemos perder menos tiempo realizando el experimento. Esto se puede ver en la Figura 2.1 que representa el porcentaje de frases completadas mientras se va aumentando la cantidad de grabaciones. Teniendo en cuenta este algoritmo podemos notar que aproximadamente a partir de 10 grabaciones ya tenemos un buen porcentaje de cubrimiento. Por ejemplo, en la décima grabación la regla 4 ya tiene el 75 % de sus frases ya grabadas. La regla 5 el 50 % de sus frases ya grabadas. O sea, en la grabación número 10 ya se grabó alrededor del 40 % de la cantidad total de frases para cada regla.

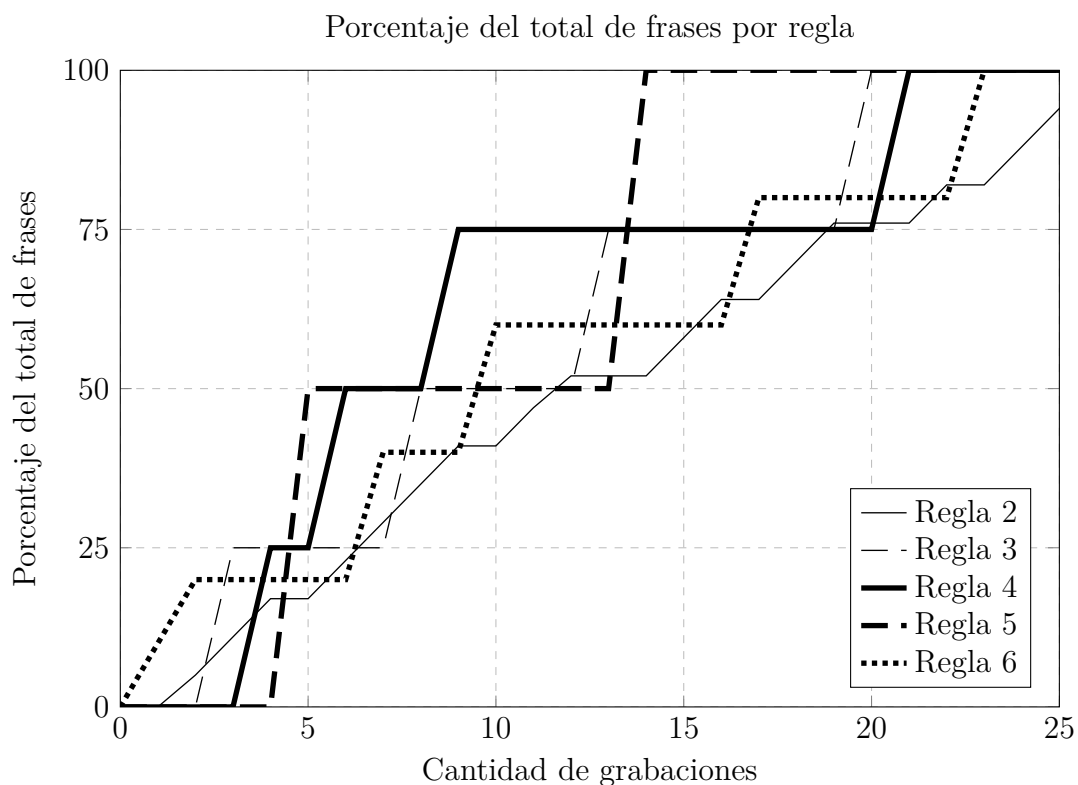


Figura 2.1: Porcentaje del total de frases grabadas por cada regla

2.1.3. Combinando los dos tipos de frases utilizando trazas

Definimos frases comunes y AMPER para grabar. Ahora debemos definir cómo vamos a ir intercalando cada tipo en el experimento. Para ello definimos traza. Una traza es una lista de las frases que va a grabar un hablante en el experimento. Esta va a estar compuesta entre 1 a 3 frases comunes extraídas del orden definido en *OrdenDeFrasesConocidas*, y luego una frase del esquema de AMPER. Tanto la cantidad de frases comunes como la elección de la frase AMPER se realiza al azar. Este patrón se repite sucesivamente hasta completar todas las frases. La idea es no cansar al hablante con frases repetitivas y evitar que sepa de antemano qué frase va a tener que grabar ya que si fuera el caso, podría exagerar la entonación.

Al empezar el experimento, al hablante se le dará una traza que grabará sucesivamente en ese orden. Elegimos tener precalculadas las trazas para evitar cálculos costosos a la hora de empezar el experimento. Si no se precalcularan las trazas, deberíamos realizar los cálculos cada vez que empieza el experimento y podríamos retrasar la grabación del hablante. Es por eso que guardamos 10.000 trazas generadas.

La mínima cantidad de grabaciones que puede realizar un hablante son 5 grabaciones. Luego se le pregunta si quiere continuar grabando. Si acepta, se le agregan otras 5 grabaciones así sucesivamente hasta llegar al total de frases a grabar. Elegimos grabar cada 5 grabaciones para que el hablante aporte el tiempo que tenga disponible y no obligarlo a grabar todas las frases. Aunque no grabe la totalidad de frases, las muestras van a servir en el experimento.

A continuación veremos cómo implementamos el sistema de grabación para soportar este experimento.

Capítulo 3

Sistema de grabación online

Para poder obtener audios de distintas personas se desarrolló una página web. Esto es muy conveniente ya que nos permite grabar fácilmente desde cualquier lugar. En esta sección explicaremos la arquitectura del sistema y sus detalles técnicos.

La página web está desarrollada en Django, versión 1.4.2. Se eligió este framework por su facilidad a la hora de guardar objetos a la base de datos y también por la cantidad importante de bibliotecas que posee Python. La versión de Python que se utilizó es 2.7.3.

En la base de datos se guarda la información de cada hablante, las frases a grabar y las trazas. La base de datos elegida fue PostgreSQL versión 9.1 y se escogió ésta ya que es de código abierto entre otras cosas. Los audios se guardan en archivos *wav* por separado y también se guarda una referencia al nombre del archivo generado en la base de datos. Para el servidor HTTP se utiliza Apache versión 2.2.22. El servidor utiliza el sistema operativo Ubuntu 12.04.4 LTS.

3.1. Recolección de datos

Cuando un usuario visita nuestra página, primero debe llenar un formulario. Este le pregunta: género, fecha de nacimiento, lugar donde se crió y donde reside actualmente. Al confirmar el formulario, estos son grabados en la base de datos de la aplicación en el servidor. Esto se puede apreciar en la Figura 3.1. Luego se procede a realizar las grabaciones.

En la pantalla de grabación el usuario debe confirmar el acceso al micrófono que posee en su computadora, como se puede apreciar en la Figura 3.2. Una vez hecho esto, se le explica las instrucciones del experimento y luego puede empezar a grabar.

Cada nuevo experimento utiliza una nueva traza del conjunto de trazas descriptas en el capítulo anterior. La interfaz que ve el usuario al grabar cada frase se puede ver en la Figura 3.3. Las grabaciones pueden ser escuchadas por el usuario. Si el usuario al escucharla nota que no es una buena grabación, tiene la posibilidad de grabarla

¡Bienvenido/a!

Este proyecto consiste en **grabar una serie de frases a través de tu computadora**, para luego poder estudiar las características del habla de cada región (por ejemplo, la tonada o los sonidos empleados).

Requisitos para poder participar:

1. Tener una **buena conexión a Internet**, preferentemente, no wireless.
2. Tener un **buen micrófono**, preferentemente, no usar el micrófono incluido en una laptop.
3. Estar en un **ambiente silencioso**.

Si cumplís estos requisitos, por favor completá los siguientes datos para comenzar:

Sexo:

Lugar donde te criaste:

Lugar donde vivís actualmente:

Mes de nacimiento: 01-1990

¡Empezar!

Figura 3.1: Encuesta inicial del sistema

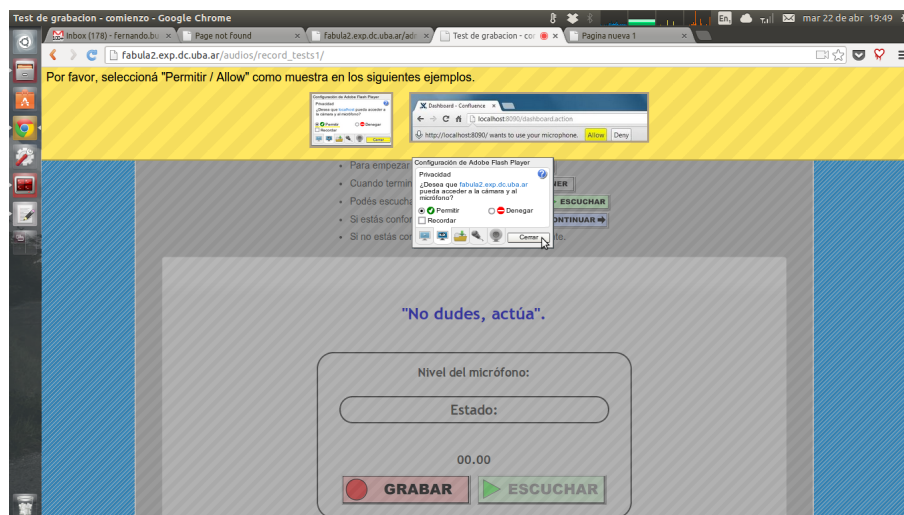


Figura 3.2: Se debe permitir micrófono para comenzar el experimento

otra vez. Lo importante es que la última grabación se escuche lo mejor posible. Para que el usuario reproduzca su grabación se aprieta el botón *Reproducir* como se ve en la figura 3.4.

Una vez que el hablante cree que su grabación se escucha bien, la confirma. Cada vez que se graba un audio, este se guarda en un archivo wav en el servidor. El archivo que se genera tiene una frecuencia de muestreo de 22050 Hz, cada muestra se analiza con 16 bits y posee un solo canal. Con estas características pudimos obtener un audio de buena calidad para el experimento que realizamos.

Recordemos que los hablantes graban en series de 5 frases. Una vez terminado estas 5 frases se le pregunta si quiere seguir grabando o terminar el experimento. De esta forma, el hablante aporta el tiempo que puede disponer.

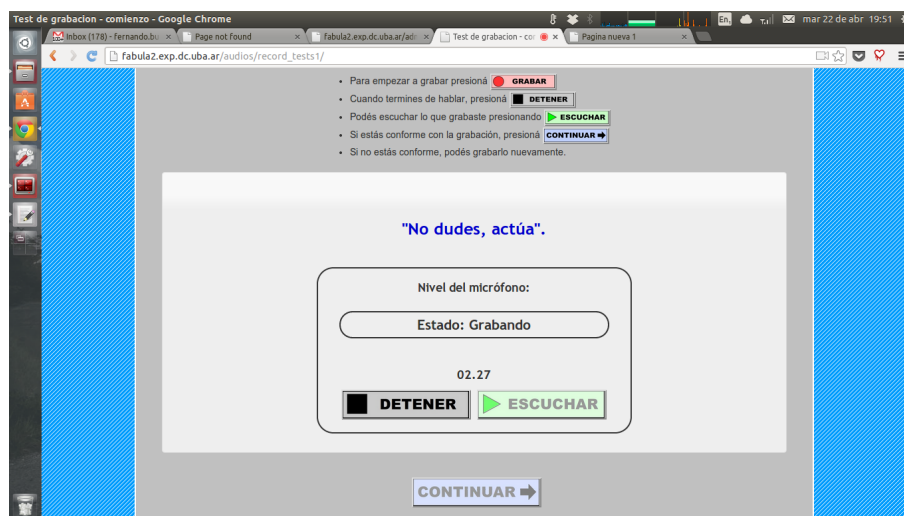


Figura 3.3: Grabando una frase



Figura 3.4: Reproduciendo la frase anteriormente grabada

3.2. Grabación a través del browser

Los navegadores actuales no permiten acceder al micrófono directamente. Actualmente se desarrolla HTML5 que permitirá acceder al micrófono y a recursos similares de forma más fácil. No se eligió basarse en éste porque sólo los browsers más nuevos lo soportan. Al ser un estándar muy nuevo necesita que el usuario tenga instalada últimas versiones de software y utilizarlo hubiera excluido mucha gente. Teniendo en cuenta esto debimos utilizar una tecnología alternativa.

Encontramos un proyecto llamado Web Accessible Multimodal Interfaces¹ (WAMI). WAMI es una aplicación Flash que nos permite acceder al micrófono a través

¹Página web: <https://code.google.com/p/wami/>

de JavaScript. El proyecto WAMI es muy utilizado en proyectos similares de procesamiento de habla. Esta herramienta nos permite definir dos URLs importantes: una que se utilizará para enviar el audio grabado y otra para escucharlo.

Cuando termina de grabar, se envía un mensaje POST al servidor a la URL configurada. El servidor obtiene el paquete de información y lo guarda como archivo wav. Cuando se quiere reproducir algún audio se envía un mensaje GET a la otra URL. El servidor lo responde con el audio requerido y se reproduce en el navegador. También con WAMI se puede configurar la calidad del audio grabado y analizar el nivel del volumen que posee.

3.2.1. Requerimientos

Los requerimientos para participar del experimento son básicos: se debe disponer de micrófono y conexión a Internet. Tuvimos problemas sobre el browser que se utilizaba: WAMI necesita Flash versión 11.04 que no se encuentra en los repositorios tradicionales de Ubuntu. De esta manera, los navegadores que utilicen Flash instalado por el sistema operativo Ubuntu no podrán correr. Otros sistemas operativos como Windows o MacOS no tienen problemas con la versión de Flash instalada. De todas formas el navegador Chrome posee preinstalado la última versión de Flash, con lo cual este navegador puede correr perfectamente la aplicación sin importar el sistema operativo que se utilice.

3.3. Varias grabaciones por frase

Recordemos que para tener la mejor grabación de cada frase, le dimos la opción al hablante de que pueda escuchar como quedó. Esto requiere un ida y vuelta de paquetes entre el cliente (navegador) y el servidor.

Al grabar el cliente manda un mensaje al servidor con el audio de la grabación en crudo. Las frases son de corta longitud entonces no es necesario preocuparse por la longitud del paquete. Cuando el cliente quiere escucharlo envía un mensaje pidiendo ese mismo audio anteriormente grabado. El servidor envía el audio y es reproducido en el cliente. Este ida y vuelta de la grabación podría ser optimizada para que la grabación pueda ser escuchada sin tener interacción con el servidor. En nuestro experimento, no tuvimos problemas graves en lo que respecta a latencias pero es un punto débil del sistema que hay que tener en cuenta.

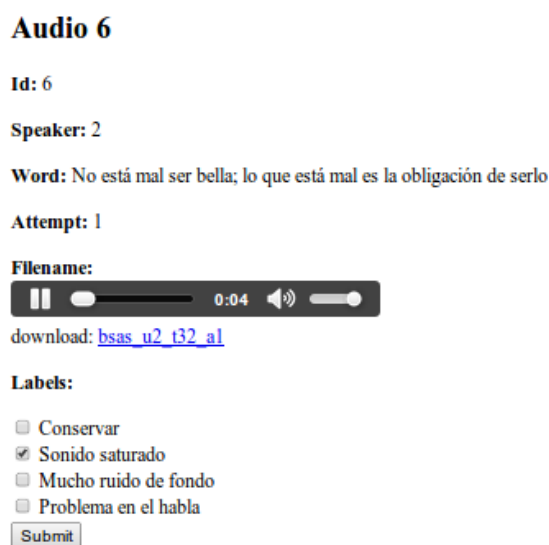
Puede resultar interesante analizar los anteriores audios grabados para ver por qué el hablante elige el último. Esta idea también puede motivar a algún trabajo futuro.

3.4. Sistema de administración

Además de la interfaz pública para grabar audios, implementamos un sistema privado para administrar las grabaciones. Éste nos permite ver las grabaciones que fueron grabadas, la cantidad de grabaciones por cada frase que tenemos recolectada, la cantidad de trazas que todavía no se utilizaron, entre otras cosas. También nos permite escuchar y marcar las grabaciones para utilizarlas como primer filtro de las grabaciones. Explicaremos esto en más detalle a continuación.

3.4.1. Etiquetado de audio

Cuando varias personas terminan el experimento, los administradores pueden acceder a una página donde se puede escuchar cada audio que se va grabando. Los administradores escuchan las grabaciones y según su calidad los etiquetan con alguna de las etiquetas definidas. Las etiquetas utilizadas esta vez son: ‘Conservar’, ‘Sonido saturado’, ‘Mucho ruido de fondo’, ‘Problemas en el habla’. Esto se puede ver en Figura 3.5.



The screenshot shows a web interface for labeling audio. At the top, it says 'Audio 6'. Below that, it displays 'Id: 6', 'Speaker: 2', and 'Word: No está mal ser bella; lo que está mal es la obligación de serlo'. It also shows 'Attempt: 1' and 'Filename:'. There is an audio player with a play button, a progress bar at 0:04, and a volume icon. Below the player, there is a download link: 'download: [bsas_u2_t32_a1](#)'. Under the 'Labels:' section, there are four checkboxes: 'Conservar' (unchecked), 'Sonido saturado' (checked), 'Mucho ruido de fondo' (unchecked), and 'Problema en el habla' (unchecked). At the bottom, there is a 'Submit' button.

Figura 3.5: Categorizando audios

Para acceder a los audios que fueron etiquetados de una determinada manera, el sistema tiene distintas URLs que nos permiten bajar todos esos audios en un archivo comprimido. Entonces si quisiéramos bajar todo el audio etiquetado con la categoría ‘Conservar’ podemos acceder a una url y bajarnos sin necesidad de entrar al servidor. Se necesita ser usuario del sistema para poder acceder a este sistema.

3.5. Backups automáticos

El sistema posee backups que se generan a la noche automáticamente. Los backups consisten en un volcado de información de toda la base de datos y en la sincronización de las grabaciones con una carpeta externa de backup.

3.6. Análisis del volumen

Un requerimiento primordial de este experimento es evitar grabaciones saturadas. Para ello medimos el volumen de la grabación mientras esta se produce. El resultado es una serie de valores entre 0 a 100. Sobre estos valores calculamos el máximo y el mínimo. Si el primero es mayor a un cierto umbral arbitrario (o sea mayor a 80 por ejemplo) significa que la grabación saturó en algún momento. Si el mínimo es mayor a un cierto umbral (o sea menor a 20 por ejemplo) quiere decir que hay mucho ruido ambiente. En cualquiera de los dos casos podemos pedirle al usuario que grabe nuevamente el experimento. De esta forma podemos filtrar grabaciones que no nos servirán para reconocer el acento.

Si bien la característica de filtrar por volumen fue programada, no fue utilizada en este experimento. El motivo fue que queríamos chequear cuán bien funcionaba la herramienta sin filtros y con completa participación de los usuarios. Otro motivo fue la paciencia de los hablantes: puede suceder que no se logre un ambiente beneficioso para grabar y el filtro rechace todos sus audios, perdiendo un posible hablante. También notamos que había grabaciones que fueron rechazadas por el filtrado de volumen pero en realidad eran de buena calidad. Esto no lo queremos como primer experimento del framework. Por eso elegimos aceptar todas las grabaciones.

A continuación veremos cómo utilizamos esta información recolectada para el objetivo del experimento.

Capítulo 4

Extracción de información

Utilizando nuestra página web podemos obtener distintas muestras de Córdoba y Buenos Aires. ¿Cómo podemos analizar estas grabaciones correctamente? Un archivo wav, similar al que se genera en cada una de las muestras, posee muchísima información. Es por esto que debemos seleccionar correctamente qué partes de la información nos sirven y qué partes podemos descartar.

4.1. Alineación forzada

Una grabación a partir de una frase posee muchísima información. Debemos seleccionar qué parte de esta grabación nos interesa y qué parte puede ser descartada. Para ello etiquetamos en qué partes del audio se pronunció cada fonema y también, uniendo cada uno de estos fonemas, etiquetamos cada palabra. Por ejemplo: si tenemos la grabación de la frase ‘*El canapé salió espectacular*’, utilizamos un archivo aparte que nos dice ‘*«espectacular» se escucha entre el segundo 0.90 y 1.18*’. Lo mismo sucede para cada palabra y fonema de la grabación. Para marcar estas anotaciones utilizamos el formato de archivos TextGrid del programa Praat [Boersma and Weenink, 2013].

Un dato muy importante es que este etiquetado no debe tener que ser realizado con intervención de un humano. Si fuera el caso, tendríamos que hacerlo uno por uno, y al tener muchos audios sería un trabajo muy arduo. De esto se encarga la alineación forzada. Las partes que debemos extraer de los audios son donde se encuentran la diferencias de cada regla descripta anteriormente.

4.1.1. Prosodylab Aligner

Debemos tener una herramienta que nos permita obtener estos pequeños fragmentos de audio para analizar sus diferencias. Usamos una, llamada ProsodyLab Aligner [Gorman et al., 2011]. Su función es realizar alineaciones automáticas en cada uno de los audios de forma fácil. Analiza uno por uno cada audio y mediante

un diccionario determina en qué momento se dijo cada fonema y palabra.

Una particularidad que se destaca de esta herramienta es que no necesita datos de entrenamiento. Sólo con una hora de grabación es suficiente para correrlo y obtener resultados. Otra característica es que puede utilizarse para cualquier idioma. Esta herramienta está hecha en lenguaje Python (versión 2.5) y sirve como *wrapper* para utilizar HTK fácilmente. HTK es una librería para crear y manipular Modelos Ocultos de Markov fácilmente, y SoX, que nos permite trabajar con audio a través de la consola.

Modelos ocultos de Markov

Los Modelos Ocultos de Markov [Rabiner, 1989] (en inglés HMM) son modelos estadísticos por los cuales tratan de predecir parámetros ocultos a través de observaciones. Se llaman “ocultos” ya que el estado a predecir no se puede observar directamente. Sólo se puede predecir en qué estado oculto está a través de observaciones.

Un ejemplo de este modelo podría ser predecir la presión atmosférica observando sólo si el tiempo está lluvioso o seco. Los estados ocultos serían “Baja” o “Alta” presión atmosférica, que corresponde a lo que queremos saber. Las observaciones para predecir estos estados serían “Lluvia” o “Seco”. En la Figura 4.1 se puede apreciar este ejemplo. Para saber en qué estado oculto se encuentra a través de las observaciones se utiliza el algoritmo Viterbi.

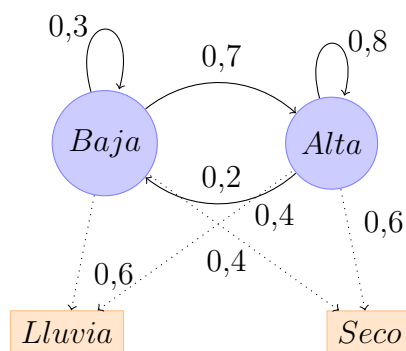


Figura 4.1: HMM

Llevado esta idea a nuestro trabajo, los Modelos Ocultos de Markov tratan de predecir qué fonemas aparecen en cada parte de los audios utilizando la lista de fonemas pronunciada en cada grabación. Mediante este modelo matemático, el programa analiza en cuáles grabaciones, de la misma frase, se produce un mismo patrón de sonido. Si sucede esto en todas estas grabaciones, se lo marca como un fonema de la frase. Ese fonema va a ser marcado de igual forma en el TextGrid de cada una de las grabaciones. Los estados ocultos a predecir son los fonemas de cada frase, en cambio las observaciones provienen del análisis de cada audio. Entonces, a través de

muestras va prediciendo los fonemas de las grabaciones.

Veamos un ejemplo de cómo funcionan los HMM en nuestro trabajo.

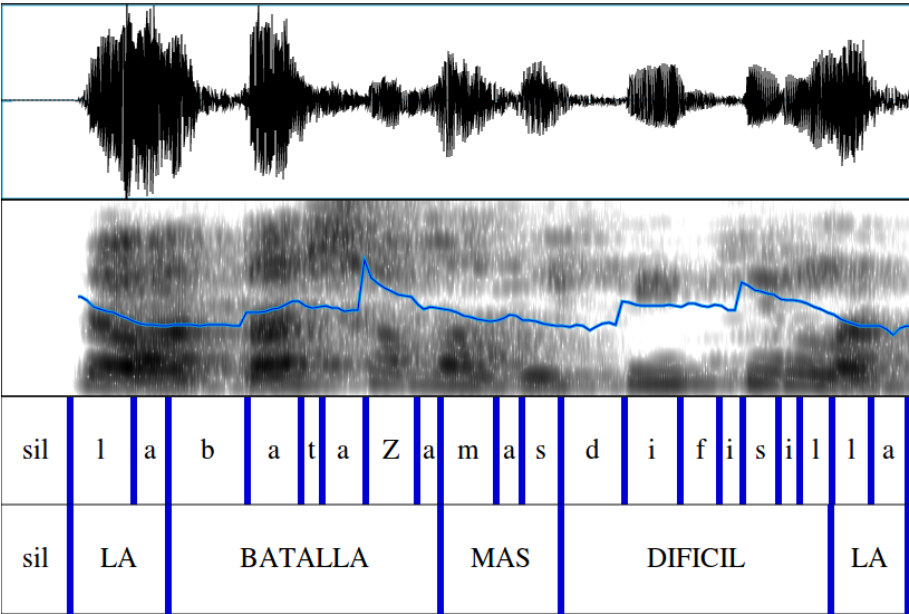


Figura 4.2: Características audio 1

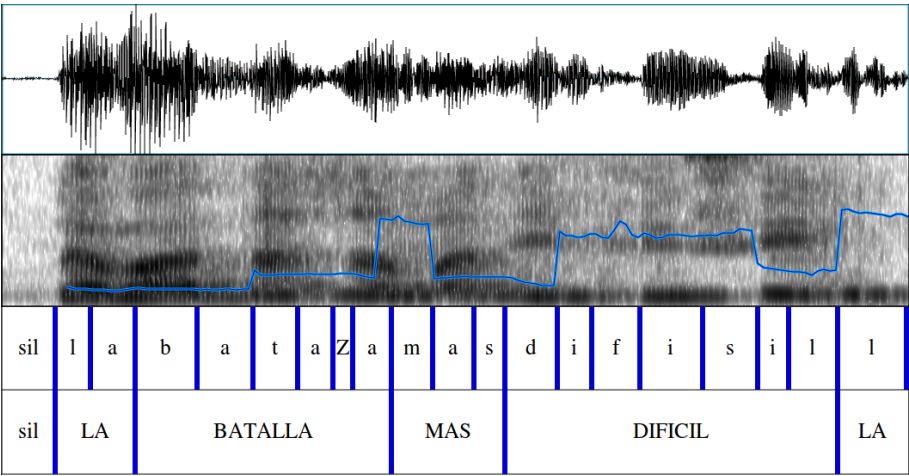


Figura 4.3: Características audio 2

Las imágenes 4.2 y 4.3 muestran las características de dos grabaciones, de dos hablantes distintos, que pronuncian la misma frase. Estas características están representadas por tres gráficos. El primero nos muestra la onda del sonido de la grabación a través del tiempo, el segundo el espectrograma y el tercero corresponde al comienzo y fin de cada fonema en la grabación.

Podemos observar que el espectrograma de ambas grabaciones es muy distinto. Si comparamos la zona del espectrograma que se encuentra el enésimo fonema con

respecto al mismo fonema del otro espectograma, vemos que ambas son muy diferentes. Aún así la alineación de los fonemas es similar. Si bien no son zonas iguales, poseen similares valores de MFCC¹ para esa región. Los Modelos Ocultos de Markov analizan los coeficientes MFCC y teniendo en cuenta la sucesión de fonemas de la frase, que es similar para ambas grabaciones, puede predecir los tiempos de inicio y fin de cada fonema. De esta forma, puede predecir desde y hasta qué tiempo se encuentra cada fonema.

Volviendo a nuestro sistema, los requisitos para utilizar ProsodyLab Aligner son: una hora de grabación con alineamientos y un diccionario fonético que nos provea para cada palabra los distintos fonemas que la componen. La hora de grabación la debíamos cumplir recolectando grabaciones de la página web. Esta meta era posible de realizar. La creación de un diccionario fonético era más complicado, ya que debía ser en español. Gracias al *Laboratorio de Investigaciones Sensoriales*² que nos prestó un diccionario, implementado por ellos, pudimos utilizar esta herramienta. Un diccionario fonético es básicamente un listado con las palabras que utilizamos y su transcripción en fonemas. Es importante esto ya que va a ser usado por el alineador para describir los fonemas de cada palabra en cada frase.

4.2. Extracción de atributos

La extracción de atributos fue realizada utilizando el lenguaje Python, que elegimos ya que es fácil de programar y tiene muchas librerías útiles para este tipo de casos. Utilizamos una librería muy conocida llamada Numpy (versión 1.6.1). Esta se utiliza para realizar cálculos matemáticos. Nosotros la utilizamos para tener buena precisión en el cálculo de los atributos.

Después de la alineación realizada, se ejecuta el extractor de atributos. Este posee como input los archivos Wav y TextGrid que corresponden a las alineaciones temporales de cada fonema en cada audio. El workflow del extractor se puede ver en la Figura 4.4.

La rutina principal del programa toma una por una las grabaciones y su etiquetado asociado. Esta se representa en el componente “Extractor”. Este llama a distintos componentes que van a calcular cada atributo. Para calcular estos atributos vamos a dividir en dos componentes: “Acoustic Extractor” que se encarga de extraer *atributos acústicos* y “TextGrid Extractor” que se encarga de extraer *atributos temporales*. Los atributos temporales son calculados solamente utilizando el TextGrid mientras que los atributos acústicos son calculados no sólo el TextGrid sino también el cálculo de MFCC, que veremos más adelante. Si el atributo está presente en la grabación tendremos ese dato en la extracción, si no se dejará como desconocido. Luego de calculado todos sus atributos para cada grabación, juntamos todos los resultados y

¹Mel Frequency Cepstral Coefficients se verán cómo se calcular más adelante

²Página web: <http://www.lis.secyt.gov.ar/>

generamos el archivo Arff utilizando el componente “Arff File Maker”.

El archivo Arff tiene por cada línea una grabación y seguido todos los resultados del cálculo de los atributos separado por comas. Necesitamos utilizar este formato ya que es necesario para ingresar datos en Weka, la plataforma que elegimos para correr algoritmos de *machine learning*. Veamos cada uno de los dos tipos de atributos:

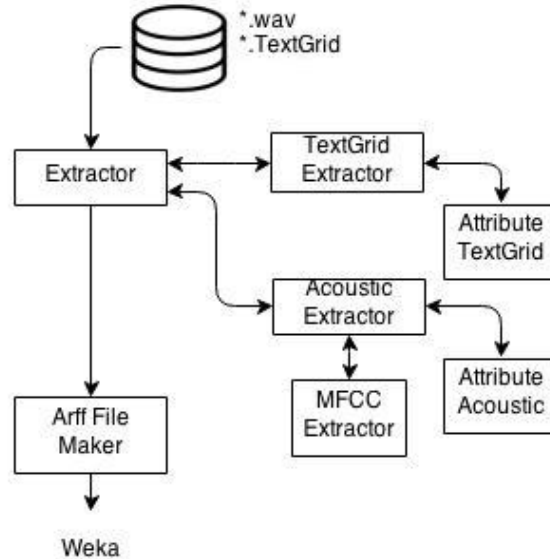


Figura 4.4: Diagrama workflow

4.2.1. Atributos temporales

Los atributos temporales corresponden a atributos sobre la duración de los fonemas y las sílabas de cada frase. Para calcularlos utilizamos como input el TextGrid generado en la alineación. Básicamente estas funciones recorren el TextGrid buscando un patrón en particular y miden su duración. Los atributos temporales se dividen en dos grupos: fonéticos y silábicos.

Luego, las mediciones son normalizadas. Se realizan dos normalizaciones. La primera, conocida como z-score, será utilizando la forma:

$$\frac{X - \mu}{\sigma}$$

donde:

- X es el valor a normalizar (por ejemplo: la duración de un fonema dado).
- μ es el promedio de duración de la unidad utilizada en la grabación.
- σ es el desvío estándar de la unidad utilizada en la grabación.

Y luego la segunda suponiendo que $\mu = 0$:

$$\frac{X}{\sigma}$$

Esta última tiene el nombre de Half-normal Distribution.

El valor a normalizar puede variar: mientras uno va a tener en cuenta fonemas, el otro tiene en cuenta sílabas. Debemos utilizar los datos normalizados ya que necesitamos atributos que nos muestren, para un hablante en particular, si el fonema en cuestión es relevante con respecto a los demás de la grabación. Al normalizar un atributo vemos cuán fuera de lo común resulta en el marco de *ese* hablante en particular en *esa* grabación. No importa si habla lento o rápido. Lo importante es la relación del fonema a medir con respecto a los demás. Lo mismo sucede para las sílabas. Si utilizáramos valores absolutos, se perdería esta relación ya que variaría con respecto a la velocidad del habla de cada hablante y cada grabación.

A continuación veamos los atributos en particular para cada uno de los grupos y cómo se realiza su cálculo. Para la regla 1 vamos a definir atributos silábicos, ya que corresponde a reglas que están definidas para sílabas, mientras que para las demás reglas (2 al 6) vamos a definir atributos fonéticos.

Atributos fonéticos

Los atributos que contabilizan fonemas son:

- **Duración de ‘kt’:** en este atributo buscamos el patrón /kt/ en los TextGrids y luego, en ese intervalo, medimos la duración del fonema /k/. Este atributo intenta extraer la diferencia explicada en la regla 4, que nos indica la duración de dicho fonema.
- **Duración de ‘sc’:** ídem con /sc/ y midiendo el fonema /s/. Este corresponde a la regla 3 que referencia a la duración del fonema /s/ anterior a /c/.
- **Duración de la ‘ll’:** buscamos el patrón /ll/ y medimos su duración. Este atributo hace referencia a la regla 5 que mide dicho fonema.
- **Duración de ‘rr’:** ídem para /r/ fuerte. Referencia a la regla 6 que hace hincapié en este fonema.
- **Duración de ‘s’ final:** ídem para las /s/ de final de palabra. Corresponde a la regla 2 que hace referencia a la aspiración de la /s/ de final de las palabras.
- **Duración de cada fonema:** este atributo mide la duración y la cantidad de todos los fonemas y luego realiza un promedio. Este no se realiza normalización ya que no se está tratando de analizar si un atributo es destacado en comparación de los demás si no que se trata de ver la duración en promedio de un fonema.

- **Duración de cada vocal:** medimos la duración media de las vocales y luego realizamos su normalización utilizando la duración de cada fonema.
- **Duración de cada consonante:** ídem anterior para consonantes.

El cálculo de un atributo fonético se realiza de la siguiente manera: supongamos por ejemplo que queremos calcular la duración de ‘kt’ en la frase “*En la pelea se conoce al soldado solo en la victoria se conoce al caballero*”. Analizamos el TextGrid asociado a la grabación que nos proveerá en qué instante se produjo cada fonema. Los fonemas en esta frase van a ser “*en la pelea se konose al soldaDo solo en la biktorja se konose al kaBaZero*”. En la Figura 4.5 se puede ver una representación gráfica del TextGrid marcando los tiempos para cada fonema³. Se marca en negro el segmento donde aparece la ‘k’.

Los valores de la normalización serán: para μ se calculará como el promedio de duración de los fonemas en la frase en cuestión. Viendo la Figura 4.5 será el promedio de los tiempos marcados para todos los fonemas. σ será el desvío estándar de la duración de todos los fonemas de la frase. Y X será el promedio de duración de los fonemas de la forma /k/ en el intervalo /kt/ correspondiente. La única aparición de este fonema es en la palabra “biktorja” y en el gráfico está marcado en negro. La misma idea se aplica en el cálculo de los demás atributos.

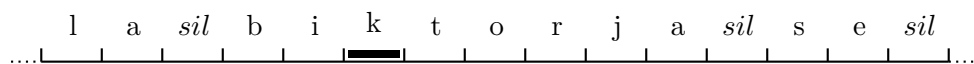


Figura 4.5: Ejemplo de cálculo de atributo

En definitiva, se busca el patrón definido por el atributo, se mide la cantidad de ocurrencias que posee y luego se realiza su normalización de las dos formas utilizando esos valores.

Atributos silábicos

Los atributos que contabilizan sílabas usados son:

- **Duración de la sílaba acentuada:** en cada una de las frases buscamos la sílaba acentuada de cada palabra, medimos su duración y normalizamos con las demás sílabas de la frase.
- **Duración de la sílaba anterior a la acentuada:** realizamos el mismo calculo anterior pero con la sílaba previa a la acentuada.

³Aclaración: la duración de los fonemas varia muchísimo. En el ejemplo se simplificó marcando todos los tiempos con el mismo tamaño para hacer más simple la figura.

Veamos cómo se realiza el cálculo de un atributo silábico: supongamos que queremos calcular el atributo que corresponde a la duración de la sílaba anterior a la acentuada y lo realizamos para la misma frase que en el caso anterior. μ representará el promedio de duración de las sílabas en la frase. σ será el desvío estándar de la duración de estas sílabas en la frase. Y finalmente X será el promedio de duración de las sílabas anteriores a las acentuadas. Para cada uno de estos valores se calculan los dos tipos de normalización. En la Figura 4.6 podemos ver este ejemplo gráficamente⁴. No pudimos escribir toda la frase y todos sus acentos por cuestiones de espacio.

En la frase del ejemplo marcamos las sílabas anteriores a la acentuada para distinguirlas. El analizador cuando calcule este atributo, va a identificar las sílabas acentuadas y tomará su antecesora.

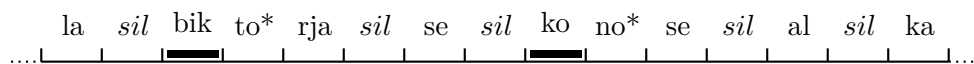


Figura 4.6: Ejemplo de cálculo de atributo

Para saber cuál es la sílaba acentuada se realizó un script que describe para cada frase cuáles son sus sílabas acentuadas. Este se encuentra en el apéndice de este informe. Estos atributos los usamos para poder medir la regla 1, la más prominente de la tonada cordobesa.

4.2.2. Atributos acústicos

Los atributos acústicos utilizan las propiedades de las grabaciones realizadas. Para ello debimos extraer información con algún método que permita medirlos. Elegimos el cálculo de coeficientes cepstrales en escala de mel (en inglés MFCC) ya que tiene relación directa con la percepción auditiva humana. Veamos el cálculo de estos coeficientes.

La forma en que hablamos se produce por varias articulaciones en nuestra boca. Estas conectan a dientes, lengua, tráquea, etc. Estas articulaciones trabajan en conjunto para darle forma al sonido producido.

⁴Aclaración: al igual al caso anterior, la duración de las sílabas varía muchísimo. En el ejemplo se simplificó marcando todos los tiempos con el mismo tamaño para ser más simple la figura.

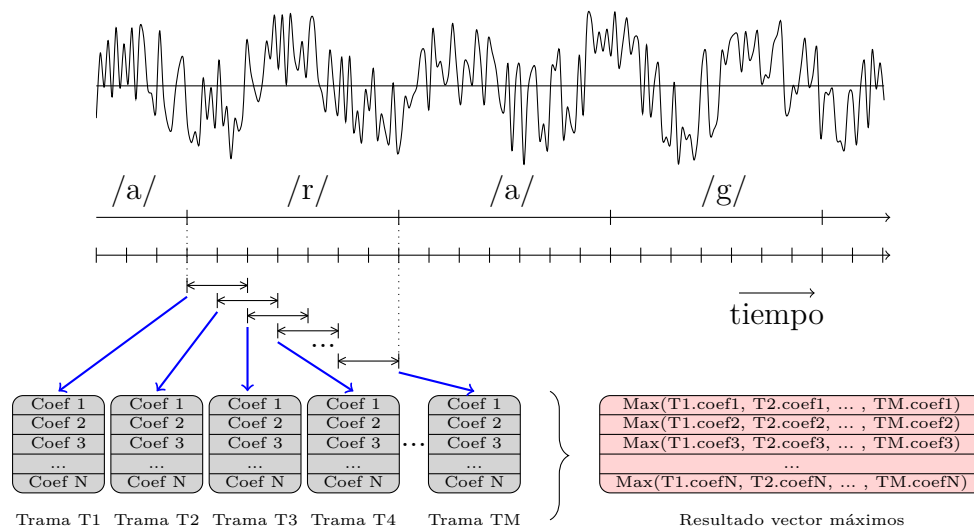


Figura 4.7: a

Las ventanas consecutivas, aplicadas a la señal del habla, se solapan. Esto permite evitar las transiciones abruptas de la señal. Las ventanas empleadas son de 20 a 40 ms de duración, pero los vectores de atributos suelen calcularse cada 10 ms. Estos intervalos son conocidos como trama o frame.

Luego de tener separada la señal en pequeñas tramas, para cada una de ellas se calcula la transformada discreta de fourier. posteriormente se utiliza un banco de filtros solapados y espaciados uniformemente sobre la escala de mel (conocido en inglés como mel filterbank). Como dijimos antes, esta escala representa la percepción auditiva humana. A los valores de energía que superaron este filtro se les aplica logaritmo. Para finalizar, a estos se les aplica la transformada discreta coseno (llamada en ingles DCT)

El siguiente pseudocódigo explica paso a paso cómo se calculan los coeficientes:

MFCC (Mel frequency cepstral coefficient):

- 1) Partir la señal **en** pequeñas tramas
- 2) Para cada frame estimar su espectro de energía utilizando Transformada Rapida de Fourier (FFT)
- 3) Aplicar el banco de filtros solapados sobre escala mel. Sumar los valores **en** cada filtro
- 4) Aplicar logaritmo a los valores de cada filtro
- 5) Aplicar Transformada de Coseno Discreta
- 6) Los MFCC son las amplitudes del espectro resultante.

Al finalizar el algoritmo obtenemos 13 atributos acústicos de ese segmento. Agregamos también la derivada de estos atributos y la segunda derivada que representan las variaciones temporales. En total derivando dos veces llegan a 33 atributos acústicos. Debemos extraer estas métricas para cada uno de los wavs grabados.

Para realizar el cálculo de estos coeficientes se utilizó un script en Matlab. El

creador del script es Kamil Wojcicki y este calcula sus primeras y segundas derivadas. El extractor necesita estos valores para cada audio a extraer. Es por eso que se conecta con Matlab a través de un wrapper para ejecutar el script y luego continuar con la extracción.

Este calculo de vectores de MFCC lo realizamos entre el principio y fin de cada atributo que represente una regla. Por ejemplo: si encontramos el atributo fonético /r/, calculamos los vectores MFCC en el intervalo donde suena este fonema. Con éstos vectores, calculamos el valor máximo, mínimo y promedio para cada enésima posición y armamos tres vectores con estos valores. O sea, el vector de valores máximos tendrá en su primer elemento el valor máximo de los primeros elementos de todos los vectores, así sucesivamente. Ídem para el vector mínimo y promedio.

4.2.3. Atributos desconocidos

Una grabación no puede definir todos los atributos que declaramos. Cada frase define sólo algunos atributos y los demás no sabe cuál es su posible valor. Es por eso que debemos tomar una decisión sobre qué hacer con los atributos que no conocemos.

Determinamos que a los atributos que no pudieron ser extraídos en la grabación los marcaremos como desconocidos. Si para una grabación un atributo es desconocido, no vamos a poder saber ese valor para ese hablante. Debemos analizar otra grabación del mismo hablante que defina ese atributo.

4.2.4. Nomenclatura utilizada

Para referenciar cada uno de los atributos debimos definir una nomenclatura. La definición que tomamos es la siguiente:

TIPO+“_”+*ATRIBUTO*+“_”+*NORMALIZACIÓN*

- *TIPO*: puede ser *FON*, *SIL* o *ACU*. Esto corresponde al tipo de atributo, si es fonético, silábico o acústico.
- *ATRIBUTO*: puede ser *kt*, *ll*, *sc*, *rr*, *Sfinal*, *vowel* o *consonant* haciendo alusión a cada una de los atributos. También aquí se encuentran los atributos generados por MFCC cuyos nombres son de la forma (*Min* | *Max* | *Avg*) + (*KT* | *LL* | *SC* | *RR*). Para hacer referencia a las reglas sobre atributos silábicos utilizamos los nombres de *syllableAccent* y *prevSyllableAccent* para la duración de la sílaba acentuada y de la sílaba anterior a esta.
- *NORMALIZACIÓN*: corresponde al tipo de normalización realizada. Estas pueden ser *norm* haciendo alusión a z-score o *normhd* haciendo alusión a normalización tomando $\mu = 0$.

Por ejemplo: definimos *SIL_prevSyllableAccent_normhd* como la duración de la sílaba anterior a la acentuada aplicando normalización con $\mu = 0$. Todos los nombres y a qué atributo se refieren se pueden ver en el apéndice de atributos.

En la próxima sección veremos los audios obtenidos en este experimento para luego analizar los atributos de cada uno.

Capítulo 5

Datos obtenidos

En este capítulo vamos a describir los datos obtenidos a través del framework desarrollado. Tuvimos algunos problemas al recolectar los audios. El principal problema fue que el ambiente utilizado por cada hablante no estaba completamente en silencio como para realizar buenas grabaciones. Muchos errores surgieron en esa dirección. Otros errores comunes pero no tan frecuentes fueron: interpretaciones erróneas de la consigna, errores de volumen del micrófono y saturación.

5.1. Evaluación manual de las grabaciones

Como primer paso, se realizó una clasificación manual de las grabaciones para determinar si las mismas se realizaron correctamente. Para la misma se utilizó la herramienta de administración que vimos en el capítulo 3. Las clases que utilizamos fueron: “Conservar”, “Sonido saturado”, “Mucho ruido de fondo”, “Problema en el habla”. La cantidad de cada clase se puede ver en la tabla 5.1.

	Bs.As.	Cba.	Total
Conservar	220	90	310
Problemas en el habla	33	15	48
Mucho ruido de fondo	2	12	14
Sonido saturado	2	0	2

Tabla 5.1: Evaluación manual de las grabaciones

Como puede observarse, los datos obtenidos están desbalanceados. No fue posible obtener la misma cantidad de audios para los dos grupos. Esto se va a reflejar en la clasificación y en el análisis posterior.

Las categorías establecidas anteriormente describen los errores comunes más frecuentes. Podemos observar que, del total de 374 grabaciones, 64 tuvieron algún problema. Esto representa alrededor del 17 % de los audios grabados. Es un número alto para ser un experimento guiado. La gran causa de este número es la falta de

chequeo al aceptar un audio nuevo. La detección automática de errores en el momento de grabación es un tema que excede los objetivos de esta tesis y se propone como trabajo futuro.

El análisis que se presenta a continuación está basado solamente en los audios clasificados como “Conservar”.

5.2. Alineación forzada

El alineador automático no realiza su función de forma perfecta. En ocasiones, el proceso de alineamiento forzado introduce errores. Es muy importante descartar los audios mal alineados, ya que si no lo hacemos, cuando los procese el extractor de atributos nos darían información errónea. Chequeamos cada audio etiquetado como “Conservar”, y analizamos con la aplicación Praat [Boersma and Weenink, 2013] si la alineación fue correcta. Los errores de alineamiento más comunes se debieron a:

- **Ruido de fondo:** los audios donde el alineador se comporta de peor manera son aquéllos en los que se escucha ruido de fondo. En esos casos, las alineaciones resultan muy malas. Lamentablemente en nuestro caso esto es bastante común. En la Figura 5.1 se puede ver un ejemplo utilizando Praat.

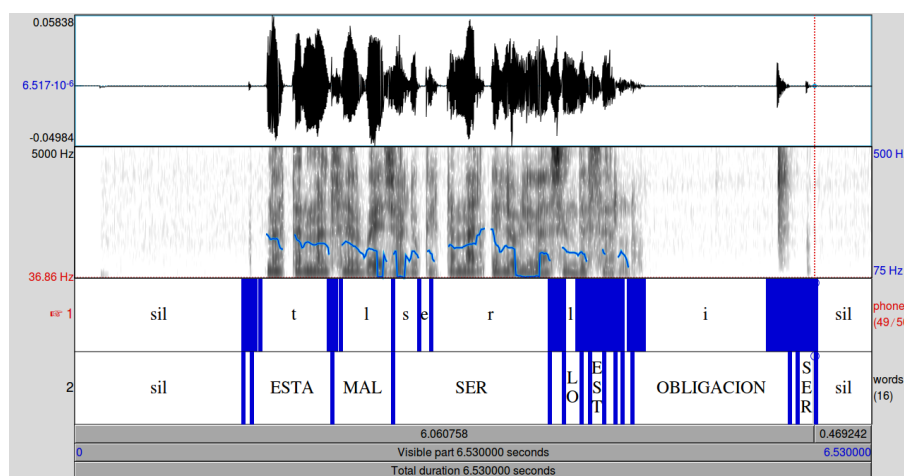


Figura 5.1: Ejemplo de alineación mala por ruido

- **”Mouse clic.” al finalizar:** las grabaciones recibieron ruido del movimiento propio del hablante. Pasó en muchas oportunidades que el clic de finalizar del mouse se grabó como parte final en el audio (Ver Figura 5.2). Ese sonido se grabó y afectó la alineación de forma tal que el alineador lo confunde con habla.
- **Saturación del micrófono:** el volumen del micrófono es configurado por el hablante. Es por ello que debemos confiar en su buena voluntad. Muchas

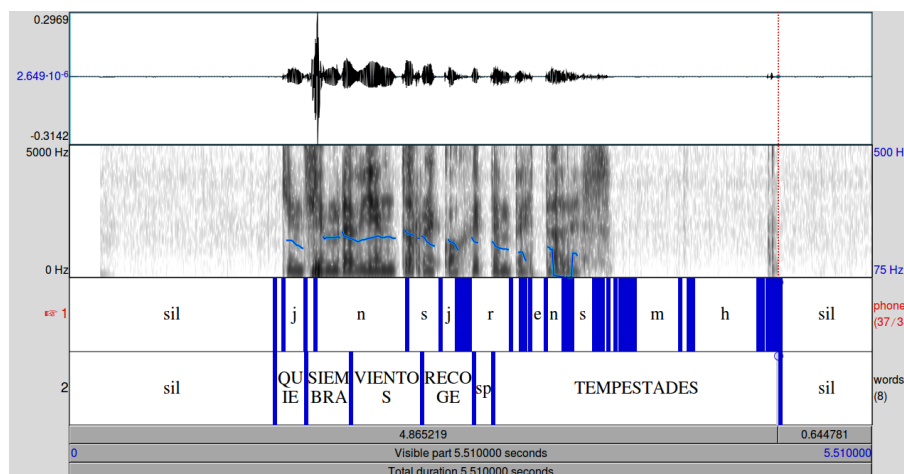


Figura 5.2: Clic al final

veces la grabación fue buena pero en algunas partes la entonación tuvo mayor volumen que en otras; haciendo que, posteriormente, la alineación no sea precisa.

- **Entonación exagerada:** en algunas grabaciones se quiso exagerar la entonación. Por ejemplo, las palabras finalizadas en /s/ fueron grabadas en muchos casos sosteniendo ese fonema por tiempo prolongado. En la mayoría de los audios no sucedió, por lo que no afectó el análisis del experimento. El problema que surgió en estos casos fue que el hablante no supo pronunciar la frase de la forma más natural posible.

5.3. Corrección de errores

Para corregir los errores descritos debimos chequear manualmente cada uno de los TextGrids. Los resultados de la cantidad de alineamientos corregidos se pueden ver en la tabla 5.2.

	Bs.As.	Cba.	Total
Modificados	101	88	189
Correctos	119	2	121
Total	220	90	310

Tabla 5.2: Cantidad de alineamientos corregidos

Entonces las grabaciones que utilizamos salieron de estas 310 alineaciones.

Recordemos que cada hablante tenía la posibilidad de grabar varias veces la misma frase con la idea de que la última grabación fuera la mejor grabada. Debemos quitar estos casos para no analizar frases que están mal grabadas. Contabilizamos los audios repetidos y los mostramos en la tabla 5.3.

	Bs.As.	Cba.	Total
Todos los intentos	220	90	310
Último intento	181	79	260

Tabla 5.3: Cantidad de audios repetidos

Entonces los audios extraídos con la herramienta que vamos a utilizar para clasificar cada uno de los hablantes son: 181 audios para Buenos Aires y 79 audios para Córdoba. Estos audios corresponden a 8 hablantes de Córdoba y 19 de Buenos Aires. En la próxima sección veremos el análisis que realizamos con estas grabaciones.

Capítulo 6

Análisis

En esta sección, analizamos los datos obtenidos para entrenar clasificadores que distingan entre hablantes de Buenos Aires y Córdoba. Recordemos que tomamos los 260 audios obtenidos a través de la página web y les aplicamos el extractor de atributos descrito en el capítulo 4. El resultado nos dio la descripción de cada grabación a través de los atributos que definimos.

Primero presentaremos el baseline que consideramos. Este nos servirá para tener una clasificación aceptable que luego trataremos de superar. Explicaremos los clasificadores utilizados para vencer esta marca y en base a tests estadísticos notaremos si aportan datos significativos. También describiremos el modelo de testing utilizando los datos recolectados. Por último, analizamos los atributos más descriptivos del habla de Buenos Aires y Córdoba.

Para el análisis de los datos, utilizamos la herramienta Weka¹. Ésta provee varios algoritmos de machine learning. Para los tests estadísticos utilizamos la herramienta R versión 3.0.1.

6.1. Baseline

El baseline define el clasificador más simple, que posteriormente tratamos de vencer. No encontramos ningún trabajo que trate de distinguir entre porteños y cordobeses a partir de su habla. Es por eso que definimos el baseline utilizando el algoritmo **majority class**. Este algoritmo clasifica eligiendo siempre la categoría que en el conjunto es mayoritaria. Por ejemplo, si nuestro conjunto de datos de train tiene más muestras de Córdoba para la clasificación de nuevas instancias vamos a elegir siempre Córdoba.

Recordemos que, como vimos en el capítulo anterior, el conjunto de datos que obtuvimos posee más grabaciones de Buenos Aires que de Córdoba. Utilizando nuestros datos, este baseline tuvo una performance alrededor del 69 % de efectividad. Nos re-

¹Página web: <http://www.cs.waikato.ac.nz/ml/weka/>

ferimos a efectividad a la probabilidad de clasificar correctamente un hablante. Este fue el porcentaje a superar. Si nuestro conjunto de datos estuviera debidamente balanceado este porcentaje sería exactamente del 50 %. Lo ideal sería poder tener misma cantidad de los dos grupos. Al tener este desbalance, puede suceder que al clasificar a un hablante en un test se obtenga mejores resultados para Buenos Aires que para Córdoba, ya que tengo más muestras para identificar a un hablante.

La herramienta Weka provee un clasificador basado en majority class llamado **ZeroR**. Utilizaremos este para el cálculo del baseline.

6.2. Clasificadores

Entrenamos varios clasificadores para poder determinar la procedencia de un hablante y superar la performance del clasificador baseline. Los clasificadores propuestos son:

Repeated Incremental Pruning to Produce Error Reduction (RIPPER) [Cohen, 1995] - **Implementación JRip**: Este algoritmo intenta describir el conjunto de entrada definiendo pequeños grupos. Primero localiza un grupo que posee la característica a clasificar, y genera reglas que lo describan. Va agregando reglas de forma golosa. Luego cuando se supera una cierta condición (por ejemplo: cantidad de reglas), lo extrae y sigue con otro grupo. Finaliza cuando describe todos los grupos del conjunto de entrada. Este algoritmo es útil para datos no balanceados.

C4.5 [Quinlan, 1993] - **Implementación J48**: Este algoritmo se basa en un árbol de decisión. Dada una serie de muestras con varios atributos se realiza lo siguiente: para cada atributo se calcula su ganancia de información. Elige el atributo que tenga mejor ganancia entre todos los atributos y con él crea un nodo en el árbol. Por cada rama que se crea el algoritmo sigue de la misma forma. Si las muestras pertenecen a la misma clase o los atributos no proveen información se crea sólo una hoja.

Support Vector Machines [Platt, 1998] - **Implementación Function SMO**: Support vector machines define uno o varios hiperplanos para intentar clasificar muestras. Este hiperplano se construye utilizando transformaciones lineales de los datos de entrada y sirve para clasificar las muestras en dos grupos. Utilizando este hiperplano, se puede etiquetar cada dato de entrada con su clasificación observando de qué lado del hiperplano se encuentra.

Naive Bayes [Zhang, 2004] - **Implementación homónima**: Un clasificador de tipo Naive Bayes asume que cada atributo describe una característica de su clase y no está relacionado con otro atributo. Cada uno de estos atributos contribuye de manera

independiente a la clasificación de su clase. Se define una regla de decisión utilizando un modelo probabilístico basado en el teorema de Bayes para la clasificación de cada grupo.

6.3. Tests estadísticos

Utilizamos los resultados de cada clasificador para ver si son significativamente relevantes en la predicción de cada hablante en comparación con el baseline. Los resultados que utilizamos surgen del porcentaje de instancias correctas para los folds generados. Los clasificadores utilizados son los descriptos en la sección anterior más el baseline. Para estos resultados realizamos dos tests principales: Prueba de rangos con signo de Wilcoxon y Test t de Student.

6.3.1. Test de Wilcoxon

Utilizamos el test de Wilcoxon ya que no estamos seguros que nuestros datos provengan de una distribución Normal. Este nos ayudará a determinar si hay razones estadísticas para decir si un clasificador es distinto que otro. Para realizar este test armamos un vector para cada clasificador incluyendo el baseline. Este vector tendrá el porcentaje de instancias correctas para cada uno de los folds. Entonces correremos el test estadístico utilizando el vector del clasificador baseline y el vector de otro clasificador, por ejemplo Support vector machines.

Las hipótesis fueron:

H_0 : Clasificador alternativo no es diferente que ZeroR

H_1 : Clasificador alternativo es diferente que ZeroR

Clasificador alternativo se refiere a los demás clasificadores descriptos.

Cada uno de los tests nos va a dar un p-valor. Si este es mayor 0,05, consideramos que no hay evidencia suficiente para determinar que el clasificador alternativo es mejor. Si de lo contrario es menor, sí podemos rechazar H_0 y asegurar que el alternativo es mejor.

Luego chequeamos si nuestra muestra corresponde a una distribución Normal. Para chequear Normalidad utilizamos el test de Shapiro-Wilk.

6.3.2. Análisis Shapiro-Wilk Test

Utilizamos el test de Shapiro-Wilk para poder afirmar si un conjunto de datos proviene de una distribución Normal.

El test de Shapiro-Wilk se basa en plantear como hipótesis nula que la población

esta distribuida de forma Normal. Aplicamos el estadístico de este test: si el p-valor nos da menor a 0,05 entonces la hipótesis nula es rechazada y se afirma que los datos no provienen de una distribución Normal. Si, en cambio, es mayor a 0,05 no hay evidencia suficiente para rechazar H_0 y por ende se afirma que los datos siguen una distribución Normal.

Este test se realiza individualmente para cada vector resultado de porcentaje de instancias correctas para cada clasificador. O sea, chequeamos que los resultados de cada clasificador se asemejen a la distribución Normal. Por ejemplo si los resultados de ambos clasificadores ZeroR y J48 tuvieron en el test Shapiro-Wilk un p-valor mayor a 0,05, se puede realizar el t de Student para ellos dos. En caso contrario, se debe usar el test de Wilcoxon.

6.3.3. Student t Test

Para los vectores resultado provenientes de una distribución Normal se les aplica este test. Este nos provee una forma de determinar si dos conjuntos de test son significativamente distintos, suponiendo que surgen de una distribución Normal. Para realizar este test utilizamos, como en el Test de Wilcoxon, dos vectores: uno para los resultados de Zero R y otro para un clasificador alternativo. También de la misma forma que planteamos la hipótesis del test de Wilcoxon, este va a tener las siguientes hipótesis.

H_0 : No hay diferencias entre ZeroR y clasificador

H_1 : Hay diferencias entre ZeroR y clasificador

La ventaja de usarlo es que, al saber qué distribución representa, vamos a tener resultados más precisos. Aplicando el estadístico obtuvimos un p-valor. De la misma forma, si este es mayor a 0,05 no hay evidencia suficiente para rechazar H_0 . De lo contrario, sí hay evidencia y rechazamos H_0 .

6.4. Modelos de tests

En esta sección vamos a explicar en detalle varios modelos de tests así como también qué resultados obtuvimos en cada uno de ellos. Éstos se basaron en la técnica de validación cruzada (en inglés cross-validation) por la cual utilizando el conjunto de los datos se arman varias iteraciones, donde en cada una de ellas se separa parte de los datos para entrenar y otra para testear. Los modelos que probamos fueron:

6.4.1. Grupos de hablantes

La complejidad del problema y la forma en que fue realizado el experimento nos llevó a tener que descartar un modelo de testing común. Si utilizamos un modelo estándar deberíamos dividir los audios en dos grupos, uno lo usaríamos para entrenar y otro para testear. En este contexto, podría surgir el problema de que un hablante tenga audios en el conjunto de train y también en el de test. En ese caso el test sería erróneo ya que podría suceder que estaríamos entrenando con audios de un hablante que luego sería testeado en otro audio.

Para evitar este inconveniente debimos tomar en cuenta los hablantes a la hora de dividir los grupos. Dividimos **a los hablantes** en dos conjuntos: uno llamado train que se utiliza para entrenar, y otro test que testea el clasificador entrenado. Si bien esto evita el problema anterior, la cantidad de audios aportados por cada hablante fue muy variable: viendo el conjunto de datos, el hablante que aportó más grabaciones realizó 30 de las mismas, mientras que el hablante que menos aportó realizó 1 solamente. Tomando en cuenta esto, la cantidad de audios de un conjunto puede quedar muy desbalanceada con respecto al otro. Por ejemplo, puede pasar que la cantidad de audios en test sea mayor a la de train. Este caso lo queremos descartar ya que estaríamos intentando clasificar sin haber entrenado lo suficiente. Para mitigar este problema, hicimos que el conjunto de train tenga el 70 % de las instancias mientras que el restante 30 % sea destinado para test. Estos dos grupos conformaron un par que lo llamaremos *fold*.

No podemos quedarnos con un solo fold, ya que podría encasillar el resultado para *ese* conjunto en particular. Es por esto que creamos 5 folds de la forma <train, test> con las características antes descritas. Este modelo de testing se denomina **cross-validation test de 5-folds**. El promedio de los porcentajes de instancias correctas de cada fold va a darnos la performance total. De esta forma, el resultado se garantiza independiente de la partición de los datos de entrenamiento y prueba. En la Figura 6.1 se diagrama este modelo de testing.

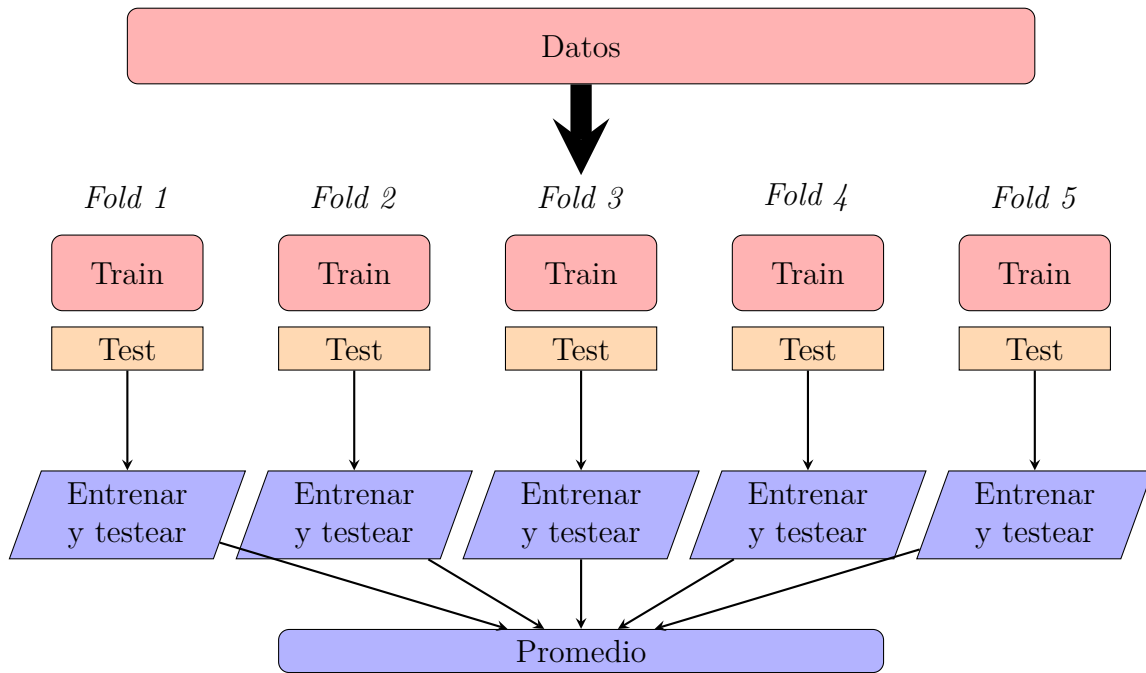


Figura 6.1: Esquema de test 5-folds

Otro problema a tener en cuenta es que los grupos de test generados sean lo más distintos posibles. Este requerimiento no es tan sencillo si uno posee tan pocas instancias, ya que podría suceder que se repita mucha cantidad de instancias entre folds haciendo que no sea provechoso el test. Para solucionar esto, el test de cross-validation fue realizado de la siguiente forma: armamos conjuntos a partir de las instancias para train y test, pero con una salvedad. Para armar el conjunto de test utilizamos el 20% de los elementos de las instancias ya utilizadas en los tests anteriores y el resto de instancias nuevas. De esta forma, regulamos la cantidad de instancias repetidas en los tests y nos aseguramos que sean todos distintos. Esto nos permitió utilizar instancias nuevas en los 5 folds.

A continuación el código generador del test cross-validation:

```

1 GeneradorDeTest:
2 Input: conjunto audios
3 Output: conjunto de <train, test>
4 resultado ← {}
5 train ← {}
6 test ← {}
7 hablantesEnTest ← {}
8 hablantes ← ObtenerHablantes(audios)
9 tamTest ← tam(hablantes) * 0.3
10 #Esta cte nos define el tamaño del fold
11 Repetir 5 veces:
12 {
13 hablantesUsadosEnTest ← ElegirRandom(hablantesEnTest, tamTest * 0.2)
14 #Esta cte nos define porcentaje de hablantes usados
15
  
```

```

16 hablantesNoUsadosEnTest ← ElergirRandom(hablantes – hablantesEnTest ,
    ↪ tamTest * 0.8) #Idem hablantes nuevos
17
18 hablantesTest ← hablantesEnTest + hablantesNoUsadosEnTest
19
20 test ← ObtenerAudios(hablantesTest , audios)
21 train ← ObtenerAudios(hablantes – hablantesTest , audios)
22
23 Si <train , test> no est'a en resultado
24 #No es un fold ya creado
25 y porcentajeMaximoDeSimilitud(test , resultado , 0.2)
26 #El test creado solo puede ser 20 % parecido a uno anterior
27 y checkBalance(train) y checkBalance(test)
28 #Chequeo del balance entre audios de BsAs y Cba
29 {
30   Agregar <train , test> a resultado
31   Agregar hablantesTest a hablantesEnTests
32 }
33 }
34 Devolver resultado
35
36 porcentajeMaximoDeSimilitud:
37 Input: conj , conjunto de fold , pctMax
38 Output: booleano
39 Recorrer test de conjunto de fold:
40 {
41   pct ← Maximo(pct , porcentajeSimilitud(conj , test))
42 }
43 Devolver pct < pctMax
44
45 checkBalance:
46 Input: conj
47 Output: booleano
48 ck_bsas ← #(conj) * 0.50 < #bsas(conj) < #(conj) * 0.70
49 ck_cba ← #(conj) * 0.25 < #cba(conj) < #(conj) * 0.45
50 ck_both ← #cba(conj) < #bsas(conj)
51 Devolver ck_bsas y ck_cba y ck_both

```

El algoritmo principal itera 5 veces para crear cada fold. En cada iteración, elige el 80 % de hablantes anteriormente no usados en ningún test y el 20 % restante de hablantes anteriormente ya usados. La elección de cada uno se realiza de forma azarosa. Esto se puede ver en las líneas 13 y 16. Luego, se obtienen de cada uno de los hablantes sus respectivos audios (líneas 19 y 20). Finalmente se chequea que el fold generado cumpla con las restricciones que habíamos definido. Estas son: el fold generado no se creó anteriormente, el test sólo tiene como máximo 20 % de elementos repetidos de los demás tests (líneas 20 y 26) y que las cantidades de audios de Córdoba y de Buenos Aires están balanceadas.

La función *porcentajeMaximoDeSimilitud* nos permite chequear que el conjunto de test generado en una iteración en particular tiene a lo sumo 20 % de audios en común con las demás instancias.

La función *checkBalance* determina si un conjunto cumple con las restricciones de balance. La primera restricción corresponde a que el porcentaje de audios de Buenos Aires en este conjunto debe ser entre 50-70 %. La segunda restricción corresponde al porcentaje de Córdoba y debe ser entre 25-45 %. Estos porcentajes intentan balancear cada test evitando que un sólo test acapare todas las instancias posibles a testear. Finalmente, la última condición pide tener más audios de Buenos Aires que de Córdoba para no agotar las instancias cordobesas.

Resultados

Recordemos que al tener pocos datos debimos repetir instancias en los grupos de tests. El porcentaje de instancias repetidas es menor al 20 %. Vamos a mostrar como son estos conjuntos:

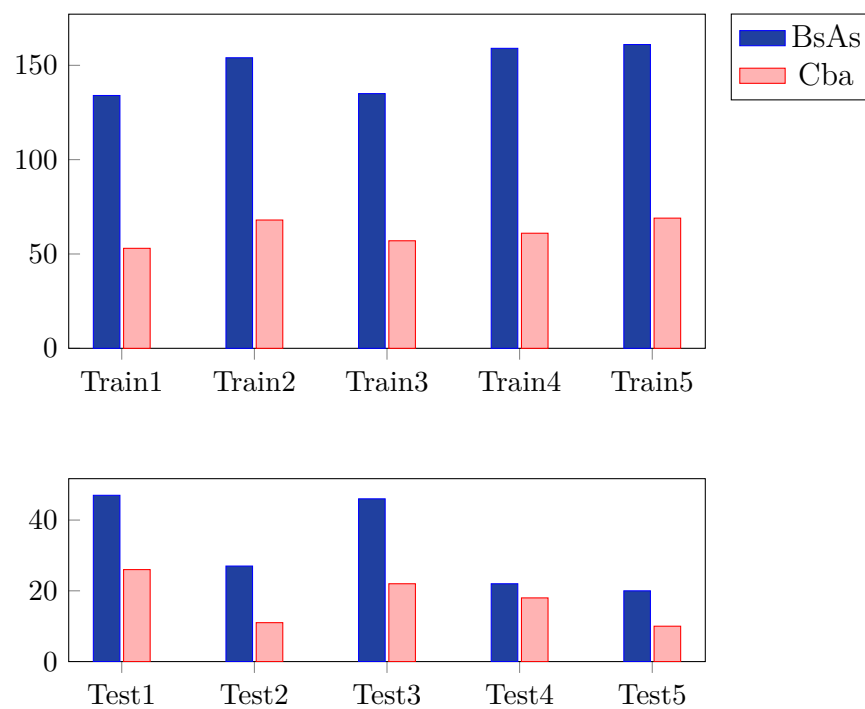


Figura 6.2: Cantidad de instancias de Buenos Aires y Córdoba según cada grupo de Train y Tests

La tabla 6.12 resume los resultados de clasificación correcta (en porcentaje) para los distintos clasificadores.

	ZeroR	JRip	J48	Function SMO	NaiveBayes
Fold 1	64	61	64	73	63
Fold 2	71	68	71	76	71
Fold 3	67	54	45	75	67
Fold 4	55	52	55	67	80
Fold 5	66	70	66	70	70
Promedio	64	61	60	72	70

Tabla 6.1: Clasificación correcta en porcentaje

En esta tabla, Fold 1 corresponde al primer par <train, test>, Fold 2 al segundo par y así sucesivamente. En la Figura 6.3 se puede ver gráficamente los resultados de clasificación correcta de dicha tabla. Excluimos a JRip y J48 por dar muy parecido a ZeroR.

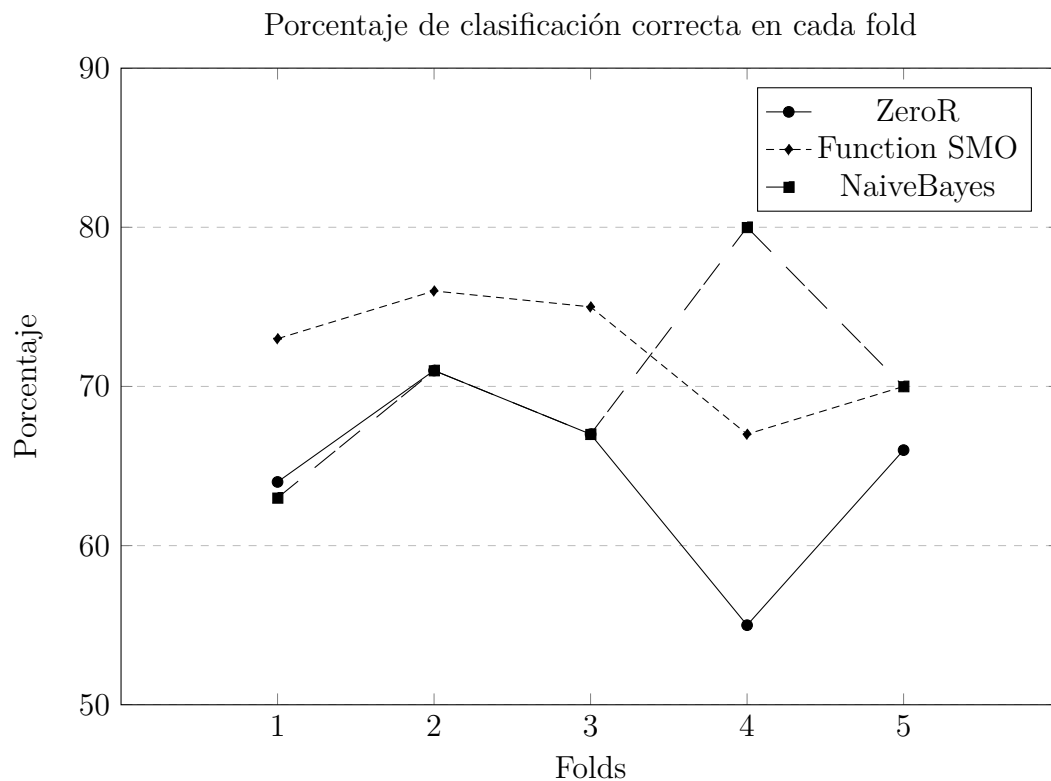


Figura 6.3: Porcentaje de ZeroR, Function SMO y NaiveBayes

Recordemos que el clasificador ZeroR elige siempre la clase mayoritaria en su grupo de test. Viendo la Figura 6.3 podemos notar que ZeroR mantiene un porcentaje en los diferentes tests entre un 55 % y casi 70 %. El clasificador Function SMO siempre se mantiene por arriba de este Baseline.

Algo interesante sucede con el clasificador NaiveBayes: muestra resultados muy similares a ZeroR pero en el test 4 posee mucha mejor performance. Viendo los

grupos generados en la Figura 6.2, el fold 4 es el que tiene menos diferencia entre cantidad de hablantes de Buenos Aires y Córdoba. Al tener mitad instancias de cada grupo y como ZeroR elige entre uno de esos dos, va a poseer un porcentaje de acierto cercano al 50 %, como sucede.

El porcentaje de exactitud en la clasificación se puede apreciar con las métricas *Precision* y *Recall*. *Precision* se define como la cantidad de verdaderos positivos sobre la cantidad de verdaderos y falsos positivos. *Recall* se define como la cantidad de verdaderos positivos sobre la cantidad de verdaderos positivos y falsos negativos. Tomamos como verdaderos positivos a la condición de que fue clasificado como Buenos Aires y efectivamente es de ahí. Falsos negativos si el hablante es de Buenos Aires pero es clasificado como Córdoba. Y falso positivo si el hablante es de Córdoba pero es clasificado como Buenos Aires.

Las Figuras 6.4, 6.5 y 6.6 muestran los valores que surgen de la matriz de confusión. Vamos a analizar como son estas métricas para el caso del test 4, que es el más interesante.

ZeroR:

BsAs	Cba	
22	18	Clasificado como BsAs
0	0	Clasificado como Cba

$$\text{Precision} = 22/40 = 0.55; \text{Recall} = 1$$

$$\text{Instancias correctas} = 55\%$$

Figura 6.4: Matriz de confusión para ZeroR en el test 4

Function SMO:

BsAs	Cba	
22	13	Clasificado como BsAs
0	5	Clasificado como Cba

$$\text{Precision} = 22/35 = 0.63; \text{Recall} = 1$$

$$\text{Instancias correctas} = 67\%$$

Figura 6.5: Matriz de confusión para Func. SMO en el test 4

NaiveBayes:

BsAs	Cba	
20	6	Clasificado como BsAs
2	12	Clasificado como Cba

$$\text{Precision} = 20/26 = 0.77; \text{Recall} = 20/22 = 0.9$$

$$\text{Instancias correctas} = 80\%$$

Figura 6.6: Matriz de confusión para NaiveBayes en el test 4

Viendo estas matrices de confusión y sus métricas podemos observar cuál es el error que se produce en cada uno de los clasificadores. Notamos que ZeroR produce mucho *Error de tipo I* (clasificador afirma que es de Buenos Aires y en realidad es de Córdoba). Esto sucede ya que elige sólo una categoría siempre. En los demás clasificadores se intenta realmente predecir y por eso los Errores de tipo I y II están más distribuidos.

Otro dato a tener en cuenta es que, si bien el clasificador ZeroR tuvo un valor alto en la métrica *Recall*, no fue lo mismo para *Precision* y por eso el valor de instancias correctas dio bastante malo. Ambos valores deben estar cercanos al 1 para tener una buena performance. Por eso en el caso de NaiveBayes; si bien ningún valor dio 1, ambos están cerca y posee en mayor porcentaje de instancias correctas.

Puede suceder que el porcentaje de instancias correctas sea el mismo pero los Errores de tipo I y II sean más balanceados. Esto es el caso del conjunto de test 1. Las tablas de las Figuras 6.7, 6.8 muestran este caso para los clasificadores ZeroR y NaiveBayes.

ZeroR:

BsAs	Cba	
47	26	Clasificado como BsAs
0	0	Clasificado como Cba

$$\text{Precision} = 47/73 = 0.64; \text{Recall} = 47/47 = 1$$

$$\text{Instancias correctas} = 64\%$$

Figura 6.7: Matriz de confusión para ZeroR en el test 1

NaiveBayes:

BsAs	Cba	
33	13	Clasificado como BsAs
14	13	Clasificado como Cba

$$\text{Precision} = 33/46 = 0.7; \text{Recall} = 33/47 = 0.7$$

$$\text{Instancias correctas} = 63\%$$

Figura 6.8: Matriz de confusión para NavieBayes en el test 1

En este caso notamos que, a pesar de que el porcentaje de instancias correctas den valores cercanos, ZeroR concentra gran parte del error en un tipo sólo, mientras que NaiveBayes lo distribuye entre los dos tipos.

No realizaremos los tests estadísticos de Wilcoxon y Test t de Student para este modelo de test. La razón es que los conjuntos de test de cada fold no son independientes. Recordemos que tienen 20 % de instancias repetidas, entonces cada fold por separado no sería estadísticamente independiente. Es por ello que dejamos de lado los tests estadísticos.

Clasificadores encontrados

Analizamos los clasificadores armados para cada fold. Estos nos dan más información para entender los resultados. Notamos que los clasificadores Support vector machines y NaiveBayes utilizaron todos los atributos para su clasificación. Fueron los que más provecho sacaron a los atributos. Por otro lado, el clasificador C4.5 armó un árbol de decisión sólo de un nivel. Es por esto que tuvo una mala performance y no aprovechó los atributos.

Para analizar mas en profundidad, veamos la salida del clasificador JRip ya que, de los clasificadores elegidos, es el que nos provee reglas utilizando una cantidad pequeña de atributos.

Cada uno de los folds devolvió un conjunto de reglas y no fueron necesariamente iguales entre sí. Estos datos corresponden a la clasificación para fold 1.

Clasificador Ripper:

- $(FON_ll_norm \leq -11,08) \text{ and } (ACU_AverageLL_6 \leq 4,308) \Rightarrow place = cba(12,0/0,0)$
- $(FON_Sfinal_normhd \leq 27,874) \text{ and } (SIL_prevSyllableAccent_norm \geq -4,265) \Rightarrow place = cba(11,0/1,0)$
- $(FON_rr_normhd \leq 31,355) \Rightarrow place = cba(10,0/2,0)$

- $else \Rightarrow place = bsas(154,0/23,0)$

En esta regla; la duración sobre /ll/, el estiramiento de la /s/ al final de la palabra, la duración sobre /r/ y la duración de la sílaba anterior a la acentuada fueron los elegidos para clasificar los dos grupos. Si bien este clasificador no obtuvo buena performance, analizar sus reglas nos permite pensar cuáles atributos tienen mayor importancia a la hora de la clasificación.

Características del modelo de test

En esta sección vamos a analizar los datos que obtuvimos en este cross-validation. Para ello analizamos los valores de la Tabla 6.12 y las características del armado de cada fold. Las características son:

- **Los grupos de tests no son independientes:** Si tomamos dos tests al azar de este cross-validation sucede que tiene instancias de audios repetidas. Esto es una característica que intentaremos evitar en los próximos CVs.
- **El clasificador C4.5 tiene el mismo rendimiento que ZeroR:** Si vemos el promedio de porcentajes clasificados correctamente y lo comparamos con ZeroR vemos que inclusive es menor. Viendo el clasificador creado por C4.5 notamos que, en todos los folds, armaba árboles de sólo un nodo de altura.
- **ZeroR tiene un rendimiento mejor al 50 %:** Este clasificador elige siempre la clase mayoritaria. Si su rendimiento es mayor al 50 % quiere decir que hay más instancias de una clase que de otra. Particularmente, hay más instancias de Buenos Aires que de Córdoba.

En los próximos cross-validations intentaremos mejorar estos problemas.

6.4.2. Un hablante para test y los demás para train

Vamos a definir un modelo de test distinto. Este va a enfatizar más la idea del hablante y de cómo distinguirlo. Para cada fold generado, excluimos un hablante y entrenaremos con todos los demás. Luego testearnos contra ese hablante excluido. Este nuevo esquema evita que tengamos audios repetidos en los grupos de tests. Podemos ver este esquema en la Tabla 6.2. Este esquema también es conocido como **validación cruzada dejando uno fuera** (en inglés: Leave-one-out cross-validation)

En nuestro conjunto de datos tenemos 27 hablantes: 19 de Buenos Aires y 8 de Córdoba. Vamos a tener 27 folds distintos para cada uno de ellos. Cada uno de estos folds va a excluir todos los audios de este hablante, por eso cuando nos referimos a un hablante nos referimos a todos los audios grabados por él.

 Hablante para train  Hablante para test

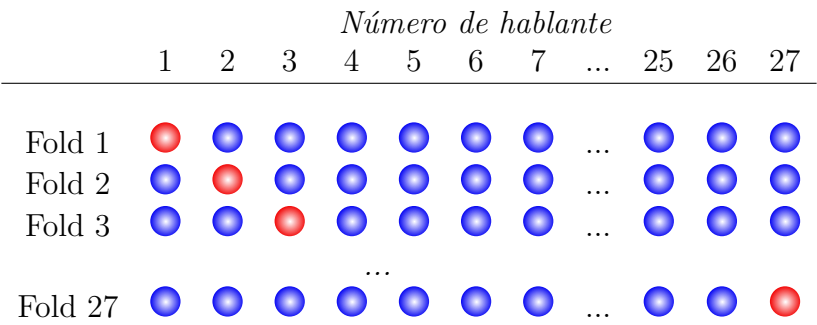


Tabla 6.2: Esquema de cross-validation

Resultados

Espesaremos solamente el promedio de clasificación correcta de los folds para cada clasificador. No resultó relevante mostrar el porcentaje de clasificación correcta para cada fold.

	ZeroR	JRip	J48	Function SMO	NaiveBayes
Promedio	70.37	69.47	70.37	71.34	71.46

Tabla 6.3: Clasificación correcta en porcentaje

Clasificadores encontrados

Analizamos los clasificadores armados para cada fold. En este modelo de test también los clasificadores Support vector machines y NaiveBayes utilizaron todos los atributos ponderando cada uno de diferente forma. El clasificador C4.5 tuvo también una performance pobre por tener un árbol de un nivel solo.

Veamos el clasificador Ripper para el fold 4. Este es el conjunto de reglas que generó:

Clasificador Ripper:

- $(FON_{rr_norm} \leq -6,901) \text{ and } (ACU_AverageRR_7 \leq 11,23) \Rightarrow place = cba(18,0/3,0)$
- $(FON_{ll_norm} \leq -7,975) \text{ and } (ACU_AverageLL_6 \leq 4,308) \Rightarrow place = cba(15,0/0,0)$
- $else \Rightarrow place = bsas(222,0/49,0)$

Notamos que sigue utilizando los atributos temporales y en menor medida los atributos acústicos.

Wilcoxon y Test t de Student

Vamos a correr los test estadísticos. Estos resultados se pueden observar en la tabla 6.13.

	Student Test	Wilcoxon Test
ZeroR y JRip	0.5816	0.6234
ZeroR y J48	1	1
ZeroR y NaiveBayes	0.4383	0.4042
ZeroR y Function SMO	0.4302	0.2646

Tabla 6.4: Resultados de cada test representado en p-valor

Los valores expresados corresponden al p-valor. Todos los clasificadores pasaron el test Shapiro-Wilk, entonces podemos afirmar que los resultados de cada clasificador corresponden a una distribución Normal. A pesar de esto, realizamos ambos tests estadísticos para cada uno. Estos test mostraron que los resultados de cada clasificador dieron que no son estadísticamente significativos.

Características del modelo de test

Si bien este modelo de test simplificó mucho la forma de modelar, nos sigue mostrando resultados llamativos.

El clasificador ZeroR sigue teniendo muy buen rendimiento solamente por tener un conjunto de datos desequilibrado. También el clasificador C4.5 posee muy bajo performance y su árbol de clasificación sigue siendo muy pobre.

Analizando el algoritmo del clasificador C4.5 notamos que el árbol generado puede ser muy pobre si en sus datos de entrenamiento hay muchos valores desconocidos.

Recordemos que el algoritmo C4.5 para armar el árbol de clasificación va tomando el atributo con mayor ganancia de información y agregándolo como un nuevo nodo al árbol. Luego descarta el atributo ya elegido y se llama recursivamente en cada rama.

Si hay muchas instancias con atributos desconocidos, no muchos atributos tendrán buena ganancia de información y no podrá armar un árbol extenso. Recordemos que para un hablante, cada uno de sus audios sólo define atributos para la frase grabada. Si en la grabación no se extrae ese atributo, quedará con valor desconocido.

Atributos		A1	A2	A3	A4	A5	A6	A7	...	AN
Hablante 1	Audio1	1	2	?	?	?	?	?	...	2
	Audio2	?	?	1	3	4	?	?	...	?
	Audio3	?	?	?	?	?	4	2	...	?

Tabla 6.5: Ejemplo de atributos para un hablante

Veamos un ejemplo: en la tabla 6.5 podemos ver los atributos extraídos de un hablante ficticio. Supongamos que este hablante grabó sólo 3 audios. En el primer audio se define los atributos A1, A2 y AN. En el segundo audio se define A3, A4 y A5. En último audio se define los atributos A6 y A7. Todos los demás atributos poseen valores desconocidos, pero en realidad corresponden a un mismo hablante. Si bien, para ese audio en particular es un atributo desconocido, la realidad es que corresponde a un mismo hablante y podemos inferirlo de otro audio grabado por él. Deberíamos tener alguna forma de evitar tener valores desconocidos cuando pueden ser extraídos de otro audio.

Suponemos que el mal rendimiento de C4.5 viene de la mano de los valores desconocidos. Creemos que si tenemos menor cantidad de valores desconocidos esta performance mejorará. Pensamos esta idea luego de leer cómo funciona los árboles de decisión del libro [Witten and Frank and Hall, 2011]. En la página 194 está la explicación que nos ayudó a razonar esta idea. Vamos a probar otro modelo de test que intente evitar tener tantos valores desconocidos.

6.4.3. Promediando los atributos de cada hablante

Vamos a armarnos un modelo de test similar al anterior pero donde su conjunto de datos no sea desequilibrado con respecto a la cantidad de instancias de cada clase. Nunca es bueno descartar datos pero debemos saber si utilizando los atributos que definidos podemos realizar una mejor clasificación.

Tomaremos 8 hablantes de Buenos Aires y de 8 Córdoba. De esta forma, el conjunto de datos está equilibrado en cantidad. La elección de cada uno de estos hablantes fue al azar. Como el esquema anterior, vamos a tener un fold por cada hablante. En la Tabla 6.6 vemos este esquema.

Hablante para train

Hablante para test

	Número de hablante											
	1	2	3	4	5	6	7	...	14	15	16	
Fold 1								...				
Fold 2								...				
Fold 3								...				
							
Fold 16								...				

Tabla 6.6: Esquema de cross-validation

Otro problema que surgió del anterior cross-validation es la masiva cantidad de valores desconocidos. Esto hace que clasificadores como C4.5 devuelvan resultados muy pobre.

Para evitar esto realizamos lo siguiente: juntamos las grabaciones de cada hablante calculando su promedio para cada atributo. Veamos en la tabla 6.7 un ejemplo de datos extraídos para entender la idea.

Atributos		A1	A2	A3	...	AN
Hablante 1	Audio1	1	?	2		2
	Audio2	?	?	1	...	?
	Audio3	2	?	3		?
Hablante 2	Audio1	1	?	?	...	?
	Audio2	1	2	?		?

Tabla 6.7: Datos originales

Para cada hablante vamos a juntar sus audios realizando el promedio de cada atributo. Por ejemplo: el Hablante 1 grabó 3 audios donde cada uno posee distintos atributos. Juntamos todos esos audios en uno promediando sus atributos: El Audio1 y Audio3 poseen el atributo A1 con valor 1 y 2 respectivamente. Entonces en la tabla 6.8 tendremos Audio1 con A1 definido como el promedio de estos valores: $1+2 = 1,5$. Ninguno de los audios grabados originalmente por Hablante 1 definió A2, entonces en este caso no vamos a poder definir ningún valor y va a quedar como valor desconocido.

Atributos		A1	A2	A3	...	AN
Hablante 1	Audio1	1.5	?	1.667	...	2
Hablante 2	Audio1	1	2	?	...	?

Tabla 6.8: Atributos modificados

Resumiendo por cada hablante se define una fila de atributos. Esto minimizaría la cantidad de valores desconocidos por cada grabación.

Esta variante la realizamos solamente utilizando los atributos temporales. Descartamos los atributos acústicos porque corresponden a grabaciones que en muchas oportunidades sufren de ruido y pensamos que también podía ser una causa por la cual los clasificadores tienen mal rendimiento.

Resultados

Los resultados de este cross-validation podemos observarlo en la página 6.12.

	ZeroR	JRip	J48	Function SMO	NaiveBayes
Promedio	53.33	60	60	93.33	80

Tabla 6.9: Clasificación correcta en porcentaje

Podemos observar que el clasificador Zero Rule estuvo más cerca de un valor esperable para el tipo de clasificador que es. Todos los demás dieron por arriba de este valor. Se destacaron los dos clasificadores que utilizan todos los atributos ponderados.

Wilcoxon y Test t de Student

Los resultados sobre el p-valor de cada test estadístico se puede ver en la tabla 6.13.

	Student Test	Wilcoxon Test
ZeroR y Ripper	0.3351	0.3828
ZeroR y C4.5	0.2908	0.3864
ZeroR y NaiveBayes	0.05191	0.06472
ZeroR y SVM	0.004282	0.009828

Tabla 6.10: Resultados de cada test representado en p-valor

Podemos observar que para el clasificador Support vector machines posee p-valor menor a 0,05 en ambas columnas. Esto quiere decir que para **Support vector machines hay evidencia suficiente de ser mejor que el baseline**. Por otro lado, los demás no pudieron lograr este cometido.

Clasificadores encontrados

Analizando los clasificadores notamos que el clasificador C4.5 armó árboles de decisión más elaborados. Notamos eso también viendo que su performance superó a Zero Rules.

Veamos un árbol de decisión generado por C4.5. Este corresponde al fold 7.

Clasificador C4.5

```

root
├── FON_vowel_norm ≤ 7,221824
│   ├── FON_ll_norm ≤ -24,007 : cba(2,33/0,22)
│   ├── FON_ll_norm > -24,007 : bsas(18,67/0,89)
│   └── FON_vowel_norm > 7,221824 : cba(5,0)

```

Los árboles de decisión generados esta vez utilizan mucho atributos como *FON_ll_norm* y *FON_vowel_norm* en varios folds. Los demás clasificadores también armaron sus reglas: SVM y NaiveBayes ponderaron cada atributo para su clasificación mientras que Ripper armó sus reglas parecidas a C4.5.

Características del modelo de test

Este modelo de test dió muy buenos resultados. El clasificador ZeroR tuvo una performance esperada de alrededor del 50 % mientras que el clasificador C 4.5 pudo armar árboles mejores.

Sin embargo, la forma que evitamos los valores desconocidos no es la mejor. El resultado de un fold para un determinado clasificador es 0 % o 100 %. Esto sucede porque es sólo una instancia la que representa.

Esto también se ve reflejado en las matrices de confusión de cada fold. Cada una de ellas son de la forma:

Buenos Aires	Córdoba	
1	0	Buenos Aires
0	0	Córdoba

Donde siempre se encuentra sólo una instancia.

6.4.4. Promediando los atributos de cada hablante sólo si es desconocido

En el anterior esquema vimos que promediando los atributo evitamos tener valores desconocidos. Sí bien esto es cierto, no es necesario promediar en todos los casos.

Supongamos que tenemos el mismo conjunto de datos a la Tabla 6.7. Si para cada hablante promediamos sus atributos estaríamos perdiendo información. El hablante 1 tiene en el Audio1 el atributo A1 definido como 1, mientras que en el Audio3 como 2. Si realizáramos el promedio, estos valores específicos para estos audios los perderíamos.

Es por ello que proponemos esta variante. Cuando haya un atributo desconocido en un audio, vamos a promediarlo con los atributos de los demás audios del mismo hablante. En la tabla 6.11 se puede ver el resultado de esta variante marcado con negrita los nuevos valores.

Por ejemplo: para el Hablante 1, el Audio1 no tiene definido el atributo A1. Entonces vamos a promediarlo con los demás audios. El Audio1 y Audio3 sí tienen este atributo definido y sus valores son 2 y 1 respectivamente. Realizamos el promedio nuevamente: $2 + 1/2 = 1,5$ entonces el valor del atributo 1 para el Audio2 es 1,5. De esta forma, no perdemos información con respecto al audio extraído.

Atributos		A1	A2	A3	...	AN
Hablante 1	Audio1	1	?	2		2
	Audio2	1.5	?	1	...	2
	Audio3	2	?	3		2
Hablante 2	Audio1	1	2	?	...	?
	Audio2	1	2	?		?

Tabla 6.11: Atributos modificados 2

Podemos observar los resultados del promedio de los folds para cada clasificador en la tabla 6.12. En este caso Zero rule dió el porcentaje esperado y los demás por arriba de su valor. Esto nos muestra que utilizando estos atributos se puede superar el baseline.

	ZeroR	JRip	J48	Function SMO	NaiveBayes
Promedio	50	72.44	73.48	77.19	74.62

Tabla 6.12: Clasificación correcta en porcentaje

Wilcoxon y Test t de Student

Corrimos los test estadísticos y obtuvimos que los resultados cada fold de NaiveBayes y Support vector machines tienen evidencia estadística para ser mejor que Zero rules. Podemos ver los p-valores en la tabla 6.13. Recordemos que para que tenga evidencia suficiente su p-valor debe ser menor a 0,05.

	Student Test	Wilcoxon Test
ZeroR y Ripper	0.06537	0.1284
ZeroR y C4.5	0.06156	0.1111
ZeroR y NaiveBayes	0.03916	0.06111
ZeroR y SVM	0.02936	0.03522

Tabla 6.13: Resultados de cada test representado en p-valor

Clasificadores encontrados

Analizando los clasificadores para cada fold notamos que todos armaron clasificadores utilizando los atributos definidos. Como el modelo de test anterior, Support vector machine y NaiveBayes utilizaron todos los atributos mientras C4.5 y Ripper utilizaron reglas más simples.

Características del modelo de test

Con esta esquema mejoramos las matrices de confusión. La tabla 6.14 corresponde al primer fold. Notamos que ahora sí se analiza cada audio y se intenta clasificarlo.

Buenos Aires	Córdoba	
33	1	Buenos Aires
0	0	Córdoba

Tabla 6.14: Matriz de confusión fold 1

Analizamos a continuación cuáles atributos aportan mayor información.

6.5. Selección de atributos de forma automática

En esta sección aplicaremos evaluadores a los distintos atributos para analizar cuáles poseen mayor importancia. El evaluador utilizado fue la ganancia de información (InfoGain en Weka) para analizar la importancia de cada atributo y utilizamos Ranker para el puntaje de los atributos.

Estos algoritmos trabajan de la siguiente forma: para cada atributo calcula la entropía de la clase y luego se calcula la entropía de la misma sabiendo el valor de este atributo. La ganancia de información de ese atributo es la resta de esos dos resultados. Esto se puede expresar como: $InfoGain(Class, Attribute) = H(Class) - H(Class|Attribute)$. Veamos cada uno de estos términos.

$H(Class)$ representa el valor de la entropía de la clase a predecir. En otras palabras, mide la incertidumbre asociada a la clase sin tener en cuenta el valor de ningún atributo en particular. Recordemos que cuando decimos clase nos referimos a Buenos Aires o Córdoba. $H(Class|Attribute)$ representa el valor de la entropía de la clase sabiendo el valor del atributo *Attribute*. A esta se le llama *entropía condicional*. Si este atributo tiene información que ayude a predecir la clase, entonces la entropía condicional será menor a la entropía de la clase. O sea, $H(Class|Attribute) < H(Class)$.

De esta forma; cuanto menor sea la entropía condicional con respecto a la entropía de la clase, mayor será la ganancia de información para ese atributo.

Ganancia de Información	Atributo
0.07231	FON_consonant_norm
0.07217	FON_vowel_norm
0.03963	SIL_syllableAccent_normhd
0.03963	SIL_prevSyllableAccent_normhd
0.02332	FON_ll_norm
0.02285	FON_Sfinal_norm
0.02226	ACU_MinLL_1
0.02144	ACU_AverageLL_1

Tabla 6.15: Resultados de InfoGain

La tabla 6.15 nos muestra que los más preponderantes se refieren a la duración de consonantes (utilizando nuestra nomenclatura FON_consonant_norm), vocales

(FON_vowel_norm), duración de la sílaba acentuada (SIL_syllableAccent_normhd) y su sílaba anterior (SIL_prevSyllableAccent_normhd). El atributo sobre la duración de la sílaba y su anterior es entendible que aporte la mayor ganancia de información, ya que es la característica más conocida para distinguir los dos grupos. Son las primeras características que uno piensa al definir el habla de un cordobés. No es extraño encontrarlos entre los primeros lugares.

Los atributos sobre duración de consonantes y vocales sorprenden con sus valores pero luego de analizarlos son entendibles. Todas las reglas definidas, salvo la regla 1 sobre estirar la sílaba anterior a la acentuada, están definidas utilizando consonantes. Vocales también pero en menor medida. Esto quiere decir que, si se cumple que la duración es menor para un par de tipos de consonantes, luego para el total va a seguir respetándose. Son variables fuertemente correlacionadas.

También algo que se desprende de este análisis es: todos los atributos del tipo fonético (empezadas con *FON*) y silábicas (empezadas con *SIL*) son sobre duración de tiempos. Si tomamos todos estos atributos y los separamos en dos grupos; uno de vocales y otro de consonantes se podría reconstruir aproximadamente los valores de los atributos sobre vocales y consonantes. Esta suma de atributos sobre vocales o consonantes van a estar definidos para todos los hablantes, mientras que atributos sobre otras reglas, por ejemplo duración de la /r/ o de la /ll/, pueden ser desconocidos o tener pocas instancias si ese hablante no grabó una frase con ese atributo.

6.6. Combinando clases de atributos

Combinando los tipos de atributos definidos pudimos apreciar cuánto aporta cada clase de los mismos. Realizamos todas las combinaciones de cada uno de los tipos de atributos. Estos son: silábicos, fonéticos y acústicos. Para cada una de esas combinaciones, corrimos los clasificadores NaiveBayes y Function SMO, que son los que mejores resultados arrojaron. Las instancias utilizadas para estos tests fueron las del cross-validation generado anteriormente. En la tabla 6.16 y en la figura 6.9 se puede apreciar los resultados. Cabe aclarar que los valores de la tabla surgieron del promedio de los resultados de los 5 tests generados.

	NaiveBayes	Functions SMO
SIL + FON + ACU	70 %	73 %
SIL + FON	69 %	66 %
FON + ACU	71 %	71 %
SIL + ACU	67 %	70 %
ACU	68 %	69 %
SIL	66 %	66 %
FON	69 %	65 %

Tabla 6.16: Combinación de atributos y su resultado

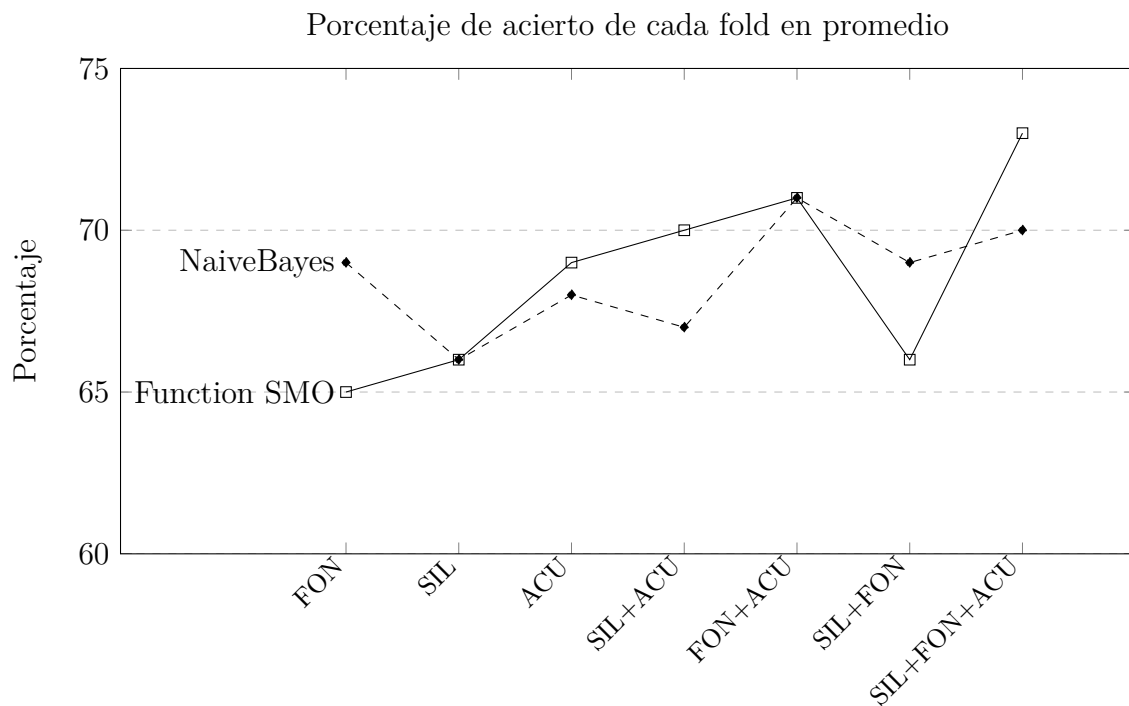


Figura 6.9: Gráfico combinando distintos grupos de atributos

Lo esperable es que aumentando la cantidad de atributos se aumente el porcentaje de instancias clasificadas correctamente. Esta idea se comprueba ya que la combinación que obtuvo mejor porcentaje fue $SIL + FON + ACU$ para el clasificador Function SMO. En segundo lugar salió la combinación de $FON + ACU$ para ambos clasificadores y en tercero $SIL + ACU$ sólo para Function SMO.

Podemos notar que el tipo de atributo que posee mayor presencia es el que corresponde a los atributos acústicos (ACU), ya que se encuentran en los tres primeros grupos de atributos que obtuvieron mejor porcentaje (estos son: $SIL + FON + ACU$, $FON + ACU$ y $SIL + ACU$). Quizás entre los atributos acústicos no haya uno con información predominante, pero la combinación de todos los atributos de esa clase hace que los clasificadores tengan buenas métricas.

Capítulo 7

Conclusiones y Trabajo Futuro

En este trabajo pudimos desarrollar una herramienta para el estudio de las variantes argentinas del español. Diseñamos desde cero una página web que nos permitió llevar a cabo experimentos del habla. Pudimos analizar las diferencias de dos variantes importantes de la Argentina: una radicada en Buenos Aires y la otra en Córdoba.

Al analizar las grabaciones recolectadas, nos enfrentamos a distintos problemas para poder utilizar esos datos. La elección de atributos a tener en cuenta y la precisa alineación de los mismos para cada audio fueron actividades importantes que se reflejaron en los datos posteriores. Fue muy interesante analizar todos los pasos, desde la obtención del audio hasta el análisis de clasificación, para ver cómo influyen en el resultado final.

Una vez obtenidos los datos y extraídos los atributos, analizamos cómo testear debidamente los clasificadores entrenados. Pudimos definir un clasificador baseline que logramos superar proponiendo diferentes clasificadores. Encontramos que *Function SMO* tuvo mejor performance ya que es un clasificador que busca el hiperplano de margen máximo entre los dos grupos, es decir, con esta división generaliza mejor que los otros clasificadores probados. También analizamos qué atributos tienen más información a la hora de clasificar en cada grupo y comprobamos que estos son los mismos que uno intuye popularmente.

Realizando este trabajo aprendimos la dificultad que surge al tener un dataset desequilibrado. Esto fue un inconveniente difícil de sobrepasar y se notó al realizar el modelo de testing. Este modelo tuvo una forma de armar los *folds* poco tradicional que si tuviéramos un dataset balanceado en clases no sería necesario. Otra lección que aprendimos es que, si bien los atributos elegidos fueron correctos, creemos que se podrían mejorar los atributos acústicos; teniendo audios más limpios y sin ruido, la calidad de estos sería mejor y por ende mejor la información que aportan.

A partir de este trabajo pueden surgir muchas mejoras descriptas en los trabajos futuros, que enumeraremos a continuación:

Chequeador cruzado: Una mejora, por parte del análisis de datos, podría ser que los audios sean chequeados entre los hablantes. Un hablante, entre grabaciones, podría escuchar un audio y su frase asociada de otro hablante que previamente realizó el experimento. Luego de escucharla, debería decidir si en la grabación se escucha correctamente la frase en cuestión. Si es así, se podría decidir que esa grabación es buena para el extractor y que se mantiene como conservada.

Si se logra que cada hablante pueda chequear que otra frase se dijo correctamente, permitiría que no sea necesario por parte de los administradores del sistema realizar este trabajo. De esta forma, se simplificaría mejor la recolección de audios y su filtrado si están mal grabados.

Validación de calidad de sonido: En el momento de grabación, se podría analizar el audio grabado y rechazarlo si no supera un nivel aceptable auditivo. Esto puede implementarse de varias formas. Una posibilidad sería cuando se está grabando medir el volumen del micrófono cada cierta cantidad de tiempo (por ejemplo: 1 segundo). Si en esa medición el volumen no se encuentra entre un rango máximo y mínimo de volumen, descartar el audio y pedirle al hablante que vuelva a grabar.

También se le podría dar más información al hablante. Sabiendo que el micrófono tuvo un pico de volumen, se podría pedir al hablante que baje el nivel de voz o se aleje del micrófono. Ídem si habla muy bajo. Otras posibles soluciones a este problema son analizar antes de empezar el experimento. Si el ruido ambiente genera saturación, pedir al usuario que realice el experimento en un lugar más silencioso.

Podemos realizar análisis más precisos sobre la calidad del audio cuando llega la grabación al servidor. En ese momento, el servidor ya puede obtener el archivo wav y realizarle todo tipo de análisis (por ejemplo: detección de ruido ambiente). Recordemos que el servidor está implementado en Python, que posee muchas bibliotecas útiles para realizar esto. Al momento de terminar de procesar el audio en cuestión, deberá enviar la respuesta al hablante informándole si se debe realizar de vuelta la grabación o si fue exitosa. Es importante notar que esta solución necesita buena conexión al servidor.

Puntaje en alineaciones: En el informe analizamos manualmente si cada audio fue alineado correctamente. Esto se pudo lograr gracias a que eran pocos audios. En un sistema automático que recolecte los audios y además extraiga cada atributo para luego entrenar los clasificadores, esta tarea no se podría realizar.

Una vez terminada una alineación, ProsodyLab-Aligner genera un archivo donde muestra cómo fueron esas alineaciones. Este archivo se llama ‘*SCORES*’ y en él se encuentra una lista de todos los audios seguidos de un puntaje, corresponde a la verosimilitud de las alineaciones. Si una alineación fue similar a otra va a tener aproximadamente un valor similar. En cambio, si posee una alineación muy distinta va a tener valores distintos. Este puede ser un buen filtro para saber si se pudo alinear bien.

Se podría definir un umbral para filtrar cuándo se realizó una alineación correcta. Si esta alineación no supera ese umbral, se debería descartar ese audio. Una cuestión importante a tener en cuenta es la cantidad de falsos positivos que pueden surgir. O sea, la cantidad de audios que no pasan el umbral pero están bien alineados.

Clasificación en vivo: Se podría realizar una prueba online de clasificación de hablantes. La misma consiste en realizar el mismo experimento para un hablante y que al final le devuelva el resultado si pertenece a Buenos Aires o Córdoba.

Esta clasificación en vivo necesitaría de los trabajos futuros relacionados con automatización del sistema. Cada vez que se realice el experimento para un hablante nuevo, se deberá recalcular el clasificador. De esta forma, se puede darle una respuesta a que grupo pertenece.

Apéndice: marcas prosódicas

En la tabla 7.1 podemos ver las marcas prosódicas de cada frase.

Frase	Prosodia
No hay dos sin tres'	[[ñó'], [áj'], ['dos*'], ['sin'], ['tres*']]
'Más difícil que encontrar una aguja en un pajar'	[[mas'], ['di', 'fi*', 'sil'], ['ke'], [éN', 'kon', 'trar*'], [úna'], [á', 'Gu*', 'xa'], [én'], [ún'], ['pa', 'xar*']]
'Más perdido que turco en la neblina'	[[mas'], ['per', 'Di*', 'Do'], ['ke'], ['tur*', 'ko'], [én'], ['la'], [ñe', 'Bli*', 'ña']]
No le busques la quinta pata al gato'	[[ño'], ['le'], ['buh*', 'kes'], ['la'], ['kin*', 'ta'], ['pa*', 'ta'], [ál'], ['ga*', 'to']]
'Se te escapó la tortuga'	[[se'], ['te'], [éh', 'ka', 'po*'], ['la'], ['tor', 'tu*', 'Ga']]
'Todos los caminos conducen a Roma'	[[to', 'Dos'], ['los'], ['ka', 'mi*', 'nos'], ['kon', 'du*', 'cen'], [á'], ['Ro*', 'ma']]
Ño hay mal que dure cien anos'	[[ño'], [áj'], ['mal*'], ['ke'], ['du*', 're'], ['cien*'], [á*, 'nos']]
'Siempre que llovió paro'	[[sjem*', 'pre'], ['ke'], ['Zo', 'Bjo*'], ['pa', 'ro*']]
'Cría cuervos que te sacaran los ojos'	[[krj*', á'], ['kwer*', 'Bos'], ['ke'], ['te'], ['sa', 'ka', 'ran*'], ['los'], [ó*, 'xos']]
'La tercera es la vencida'	[[la'], ['ter', 'se*', 'ra'], [és'], ['la'], ['ben', 'si*', 'Da']]
'Calavera no chilla'	[[ka', 'la', 'Be*', 'ra'], [ño'], ['Hi*', 'Za']]
'La gota que rebalsó el vaso'	[[la'], ['go*', 'ta'], ['ke'], ['Re', 'Bal', 'so*'], [él'], ['ba*', 'so']]
'La suegra y el doctor cuanto más lejos mejor'	[[la'], ['swe*', 'Gra'], ['y'], [él'], ['dok', 'tor*'], ['kwan', 'to'], ['mas'], ['le*', 'xos'], ['me', 'xor*']]
Á la mujer picaresca cualquiera la pesca'	[[á'], ['la'], ['mu', 'Cer*'], ['pi', 'ka', 'reh*', 'ka'], ['kwal', 'kje*', 'ra'], ['la'], ['peh*', 'ka']]
'Quien siembra vientos recoge tempestades'	[[kj*', én'], ['sjem*', 'bra'], ['bjem*', 'tos'], ['Re', 'ko*', 'Ce'], ['tem', 'peh', 'ta*', 'Des']]
'La arquitectura es el arte de organizar el espacio'	[[la'], [ár', 'ki', 'tek', 'tu*', 'ra'], [és'], [él'], [ár*, 'te'], ['de'], [ór', 'Ga', 'ni', 'sar*'], [él'], [éh', 'pa*', 'sjo']]
El amor actúa con el corazón y no con la cabeza'	[[él'], [á', 'mor*'], [ák', 'tw*', 'á'], ['kon'], [él'], ['ko', 'ra', 'son*'], [í'], [ño'], ['kon'], ['la'], ['ka', 'Be*', 'sa']]
Ño dudes actúa'	[[ño'], ['du*', 'Des'], [ák', 'tw*', 'á']]
'Perro que ladra no muerde'	[[pe*', 'Ro'], ['ke'], ['la*', 'Dra'], [ño'], ['mwer*', 'De']]
'La musica es sinónimo de libertad, de tocar lo que quieras y como quieras'	[[la'], ['mu*', 'si', 'ka'], [és'], ['si', 'ño*', 'ni', 'mo'], ['de'], ['li', 'Ber', 'taD*'], ['de'], ['to', 'kar*'], ['lo'], ['ke'], ['kje*', 'ras'], [í'], ['ko', 'mo'], ['kje*', 'ras']]
'La belleza que atrae rara vez coincide con la belleza que enamora'	[[la'], ['be', 'Ze*', 'sa'], ['ke'], [á', 'tra*', 'é'], ['Ra*', 'ra'], ['Bes*'], ['kojn', 'si*', 'De'], ['kon'], ['la'], ['be', 'Ze*', 'sa'], ['ke'], [é', 'ña', 'mo*', 'ra']]

‘No esta mal ser bella lo que esta mal es la obligación de serlo’	[[ñó’], [éh’], [ta*’], [mal*’], [ser’], [be*’, ’Za’], [lo’], [ke’], [éh’], [ta*’], [mal*’], [és’], [la’], [ó’, ’Bli’, ’Ga’, ’sjon*’], [de’], [ser*’, ’lo’]]
‘La batalla más difícil la tengo todos los días conmigo mismo’	[[la’], [ba’, ’ta*’, ’Za’], [mas’], [di’, fi*’, sil’], [la’], [teN*’, ’go’], [to’, Dos’], [los’], [dj*’, ás’], [kon’, ’mi*’, ’Go’], [mis*’, mo’]]
‘Él que no llora no mama’	[[él’], [ke’], [ño’], [Zo*’, ra’], [ño’], [ma*’, ma’]]
‘En la pelea se conoce al soldado solo en la victoria se conoce al caballero’	[[én’], [la’], [pe’, le*’, á’], [se’], [ko’, ño*’, se’], [ál’], [sol’, da*’, Do’], [so*’, lo’], [én’], [la’], [bik’, to*’, rja’], [se’], [ko’, ño*’, se’], [ál’], [ka’, Ba’, Ze*’, ro’]]
‘La lectura es a la mente lo que el ejercicio al cuerpo’	[[la’], [lek’, tu*’, ra’], [és’], [á’], [la’], [men*’, te’], [lo’], [ke’], [él’], [é’, Cer’, si*’, sjo’], [ál’], [kwer*’, po’]]
‘El pez por la boca muere’	[[él’], [pes*’], [por’], [la’], [bo*’, ka’], [mwe*’, re’]]
‘El canapé salió espectacular’	[[él’], [ka’, ña’, pe*’], [sa’, ljo*’], [éh’, pek’, ta’, ku’, lar*’]]
‘El canapé salió delicioso’	[[él’], [ka’, ña’, pe*’], [sa’, ljo*’], [de’, li’, sjo*’, so’]]
‘El canapé salió riquísimo’	[[él’], [ka’, ña’, pe*’], [sa’, ljo*’], [Ri’, ki*’, si’, mo’]]
‘El pollo salió espectacular’	[[él’], [Re’, po*’, Zo’], [sa’, ljo*’], [éh’, pek’, ta’, ku’, lar*’]]
‘El pollo salió delicioso’	[[él’], [Re’, po*’, Zo’], [sa’, ljo*’], [de’, li’, sjo*’, so’]]
‘El pollo salió riquísimo’	[[él’], [Re’, po*’, Zo’], [sa’, ljo*’], [Ri’, ki*’, si’, mo’]]
‘El esparrago salió espectacular’	[[él’], [éh’, pa*’, Ra’, Go’], [sa’, ljo*’], [éh’, pek’, ta’, ku’, lar*’]]
‘El esparrago salió delicioso’	[[él’], [éh’, pa*’, Ra’, Go’], [sa’, ljo*’], [de’, li’, sjo*’, so’]]
‘El esparrago salió riquísimo’	[[él’], [éh’, pa*’, Ra’, Go’], [sa’, ljo*’], [Ri’, ki*’, si’, mo’]]
‘En boca cerrada no entran moscas’	[[én’], [bo*’, ka’], [se’, Ra*’, Da’], [ño’], [én*’, tran’], [moh*’, kas’]]
‘Más vale pájaro en mano que cien volando’	[[mas’], [ba*’, le’], [pa*’, xa’, ro’], [én’], [ma*’, ño’], [ke’], [sjen*’], [bo’, lan*’, do’]]
‘Río revuelto ganancia de pescadores’	[[Rj*’, ó’], [Re’, Bwel*’, to’], [ga’, ñan*’, sja’], [de’], [peh’, ka’, Do*’, res’]]
‘No hay que pedirle peras al olmo’	[[ño’], [áj’], [ke’], [pe’, Dir*’, le’], [pe*’, ras’], [ál’], [ól*’, mo’]]

Tabla 7.1: Marcas prosódicas

Apéndice: nomenclatura de atributos

En la tabla 7.2 se puede ver la nomenclatura de los atributos elegidos.

Nomenclatura		Referencia
ACU_AverageKT_0 ACU_AverageKT_32	al	Componentes promedio de MFCC del fonema /k/ anterior a /t/
ACU_AverageLL_0 ACU_AverageLL_32	al	Componentes promedio de MFCC del fonema /l/
ACU_AverageRR_0 ACU_AverageRR_32	al	Componentes promedio de MFCC del fonema /r/ fuerte
ACU_AverageSC_0 ACU_AverageSC_32	al	Componentes promedio de MFCC del fonema /s/ anterior a /c/
ACU_MaxKT_0 ACU_MaxKT_32	al	Componentes máximo de MFCC del fonema /k/ anterior a /t/
ACU_MaxLL_0 ACU_MaxLL_32	al	Componentes máximo de MFCC del fonema /l/
ACU_MaxRR_0 ACU_MaxRR_32	al	Componentes máximo de MFCC del fonema /r/ fuerte
ACU_MaxSC_0 ACU_MaxSC_32	al	Componentes máximo de MFCC del fonema /s/ anterior a /c/
ACU_MinKT_0 ACU_MinKT_32	al	Componentes mínimo de MFCC del fonema /k/ anterior a /t/
ACU_MinLL_0 ACU_MinLL_32	al	Componentes mínimo de MFCC del fonema /l/
ACU_MinRR_0 ACU_MinRR_32	al	Componentes mínimo de MFCC del fonema /r/ fuerte
ACU_MinSC_0 ACU_MinSC_32	al	Componentes mínimo de MFCC del fonema /s/ anterior a /c/
FON_phoneme		Duración del fonema
place		Lugar del hablante en la grabación

Tabla 7.2: Atributos normalizados

En la tabla 7.3 se muestra la nomenclatura de los atributos que se les aplicó normalización.

Nomenclatura	Referencia
FON_vowel_norm	Duración de las vocales
FON_consonant_norm	Duración de la consonantes
FON_Sfinal_norm	Duración de la /s/ final de palabra
FON_kt_norm	Duración del fonema /k/ anterior a /t/

FON_ll_norm	Duración de la /ll/
FON_rr_norm	Duración del fonema /r/ fuerte
FON_sc_norm	Duración del fonema /s/ anterior a /c/
SIL_prevSyllableAccent_norm	Duración de la sílaba anterior a la acentuada aplicando normalización
SIL_syllableAccent_norm	Duración de la sílaba acentuada aplicando normalización

Tabla 7.3: Atributos normalizados

En la tabla 7.4 se muestra la nomenclatura de los atributos que se les aplicó normalización tomando como $\mu = 0$.

Nomenclatura	Referencia
FON_vowel_normhd	Duración de las vocales
FON_consonant_normhd	Duración de las consonantes
FON_Sfinal_normhd	Duración de la /s/ final de palabra
FON_kt_normhd	Duración del fonema /k/ anterior a /t/
FON_ll_normhd	Duración de la /ll/
FON_rr_normhd	Duración del fonema /r/ fuerte
FON_sc_normhd	Duración del fonema /s/ anterior a /c/
SIL_prevSyllableAccent_normhd	Duración de la sílaba anterior a la acentuada aplicando normalización
SIL_syllableAccent_normhd	Duración de la sílaba acentuada aplicando normalización

Tabla 7.4: Atributos normalizados

Bibliografía

- [Witten and Frank and Hall, 2011] Witten and Frank and Hall (2011). Data mining, practical machine learning tools and techniques.
- [Boersma and Weenink, 2013] Boersma, P. and Weenink, D. (2013). Praat: doing phonetics by computer [computer program] version 5.3.51, retrieved 2 june 2013 from <http://www.praat.org/>.
- [Cohen, 1995] Cohen, W. W. (1995). Fast effective rule induction. In *Twelfth International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann.
- [Elena Vidal de Battini, 1964] Elena Vidal de Battini (1964). Español en la argentina.
- [Gorman et al., 2011] Gorman, K., Howell, J., and Wagner, M. (2011). Prosodylab-aligner: A tool for forced alignment of laboratory speech. canadian acoustics. 39.3. 192–193. In *Proceedings of Acoustics Week in Canada, Quebec City*.
- [Gurlekian et al., 2009] Gurlekian, J. A., Yanagida, R., Trípodí, M. N., and Toledo, G. (2009). Amper-argentina: Variabilidad rítmica en dos corpus.
- [María Beatriz Fontanella de Weinberg, 2000] María Beatriz Fontanella de Weinberg (2000). El español en la argentina y sus variedades regionales.
- [Platt, 1998] Platt, J. C. (1998). Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report MSR-TR-98-14, Microsoft Research.
- [Quinlan, 1993] Quinlan, R. (1993). C4.5: Programs for machine learning.
- [Rabiner, 1989] Rabiner, L. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.
- [Zhang, 2004] Zhang, H. (2004). The optimality of naive bayes. In *FLAIRS Conference*, pages 562–567.