



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 presentada para optar al título de  
Licenciado 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 en 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 economía, su región entre otras.

Particularmente en Argentina no es la excepción. Nuestro país posee una fuerte componente dialéctica en su habla. Esto quiere decir que podemos saber de que lugar proviene el hablante analizando su tonada. Hay varias regiones definidas a través del país. En este trabajo nos enfocaremos en distinguir diferencias entre la región de Córdoba y Buenos Aires. Realizaremos un experimento donde compararemos el habla de cada grupo. Utilizando estos datos analizaremos efectivamente cuales son las características mas predominantes y como repercute esas diferencias en el habla. Por último, mostraremos distintos clasificadores para determinar de que grupo proviene una grabación, analizaremos las atributos mas importantes y testaremos la solución propuesta.

# Agradecimientos

A mucha gente...

*A mi viejo.*

# Índice general

<b>1. Introducción</b>	<b>1</b>
<b>2. Diseño del experimento</b>	<b>5</b>
2.1. Elección de las frases . . . . .	5
2.1.1. Frases utilizando esquema AMPER . . . . .	6
2.1.2. Frases comunes . . . . .	6
2.1.3. Combinando los dos tipos utilizando trazas . . . . .	8
<b>3. Sistema de grabación online</b>	<b>10</b>
3.1. Recolección de datos . . . . .	10
3.2. Grabación a través del browser . . . . .	12
3.2.1. Requerimientos . . . . .	13
3.3. Varias grabaciones por frase . . . . .	13
3.4. Sistema de administración . . . . .	14
3.4.1. Etiquetando audios . . . . .	14
3.5. Backups automáticos . . . . .	14
3.6. Análisis del volumen . . . . .	15
<b>4. Extracción de información</b>	<b>16</b>
4.1. Alineación forzada . . . . .	16
4.1.1. Prosodylab Aligner . . . . .	16
4.2. Extracción de atributos . . . . .	17
4.2.1. Atributos temporales . . . . .	18
4.2.2. Atributos acústicos . . . . .	22
4.2.3. Nomenclatura utilizada . . . . .	23
<b>5. Datos obtenidos</b>	<b>24</b>
5.0.4. Mediciones . . . . .	24

5.0.5. Errores comunes . . . . .	24
5.1. Alineación forzada . . . . .	25
5.2. Corrección de errores . . . . .	27
<b>6. Análisis</b>	<b>28</b>
6.1. Baseline . . . . .	28
6.2. Modelo de testing . . . . .	28
6.3. Clasificadores . . . . .	30
6.4. Tests estadísticos . . . . .	31
6.4.1. Wilcox Test . . . . .	31
6.4.2. Análisis Shapiro-Wilk Test . . . . .	32
6.4.3. Student Test . . . . .	32
6.5. Resultados . . . . .	33
6.5.1. Wilcox y Student Test . . . . .	36
6.5.2. Clasificadores encontrados . . . . .	37
6.6. Selección de atributos de forma automática . . . . .	37
6.7. Combinando clases de atributos . . . . .	38
<b>7. Conclusiones</b>	<b>40</b>

# 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 que también posee características del hablante. Estudiar estas características del habla nos permite conocer mejor la cultura de las personas. Nos permite identificar a los hablantes para saber el lugar donde pertenecen.

Identificar y extraer características del habla es una tarea muy difícil de realizar. No solo se debe 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 debe 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.

La objetivo de esta tesis es realizar un sistema que pueda facilitar 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 puede 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 las diferencias en el habla entre Córdoba y Buenos Aires. El primero de estos grupos se encuentran uno 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* [4] de Beatriz Fontanella de Weinberg y *Español en la Argentina* [3] 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/ y la /i/ y en la aspiración de la /s/. También describe el estiramiento de



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

Extrayendo el análisis de estos libros pude 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 realizan esto.** 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 para Buenos Aires.** Para las palabras terminadas en /s/ se acorta su duración en el hablante de Córdoba. Ejemplo: ‘Pájaros’ posee el fonema /s/ al final. Utilizando la dialéctica de Córdoba, la /s/ final sería mas 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.** La sílaba /s/, que precede a /c/ o /t/, suena más suave en cordobeces que en porteños. Ejemplo: ‘Mosca’ en la variante de Córdoba posee una sílaba más suave en el fonema /s/ que en Buenos Aires.
- **Regla 4: La ‘c’ antes de la ‘t’ se pronuncia con menor frecuencia para hablantes de Córdoba que para hablantes de Buenos Aires.** La sílaba /c/, que precede a /t/, no se debe pronunciar. Ejemplo: ‘Doctor’ no 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. Ejemplo: ‘Espárrago’ debe ser suave en comparación de 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 cumple.

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 cuales frases utilizamos y el medio empleado para grabar.

# Capítulo 2

## Diseño del experimento

Utilizando estudios previos de ambas variantes del español pudimos obtener las reglas definidas en el capítulo anterior. Recordemos que estas describen la diferencia entre cada uno de los dos grupos. En este capítulo proponemos realizar un experimento para poder extraer la información fonética de los mismos. La idea será realizar una serie de actividades donde el hablante sea grabado y 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 lo diga de forma lo 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 mas corta 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 [6] y lo veremos en detalle más adelante.

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. Utiliza 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 [6] donde utilizan un esquema similar. Este trabajo estudió los acentos del español Argentino utilizando todas sus combinaciones. En nuestro trabajo tenemos un problema similar por ello lo tomamos como referencia.

Para el esquema AMPER se fija un patrón de estructura de frases 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, o sea pasa por 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 estos dos grupos. 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 la Figura 2.1.

### 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 conocidas utilizadas se pueden ver en la Figura 2.2.

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

			1 - Localice la sílaba acentuada en la palabra y estirar la sílaba anterior		
			Aguda	Grave	Esdrújula
El Canapé	salió	espectacular	espectacular, canapé		
El Canapé	salió	delicioso	canapé	delicioso	
El Canapé	salió	riquísimo	canapé		riquísimo
El Repollo	salió	espectacular	espectacular	repollo	
El Repollo	salió	delicioso		delicioso, repollo	
El Repollo	salió	riquísimo		repollo	riquísimo
El Espárrago	salió	espectacular	espectacular		
El Espárrago	salió	delicioso		delicioso	
El Espárrago	salió	riquísimo			riquísimo

Figura 2.1: Frases AMPER

Tarea \ Categoría	2 - Aspiración y elisión de /s/	3 - La 's' antes de la 'c' o 't' suenan	4 Nueva - La 'c' antes de la 't' no	5 - La 'y' y 'll' se pasa a 'i'	6 - La 'r' no debe sonar. No debe
No hay dos sin tres	X (dos, tres)				
La tercera es la vencida	X (es)				
Perro que ladra no muerde					X (perro)
El pez por la boca muere	X (pes)				
En boca cerrada no entran moscas	X (moscas)	X (moscas)			X (cerrada)
Más vale pájaro en mano que 100 volando	X (mas)				
La curiosidad mató al gato					
Río revuelto, ganancia de pescadores	X (pescadores)	X (pescadores)			X (río, revuelto)
No hay que pedirle peras al olmo	X (peras)				
Más difícil que encontrar una aguja en un pajar	X (mas)				
Más perdido que turco en la neblina	X (mas)				
No le busques la quinta pata al gato	X (busqueS)	X (buSkas)			
Todo bicho que camina va al asador					
Caminante no hay camino, se hace camino al andar					
Se te escapó la tortuga		X (escapó)			
Todos los caminos conducen a Roma	X (todos, los, caminos)				X (Roma)
No hay mal que dure 100 años	X (años)				
Siempre que llovió paró				X (llovió)	
Cría cuervos, que te sacarán los ojos	X (cuervos, los, ojos)				
Calavera no chilla				X (chilla)	
La gota que rebasó el vaso					X (rebasó)
La suegra y el doctor, cuanto más lejos, mejor.	X (más, lejos)		X (doctor)		
A la mujer picaresca, cualquiera la pesca.		X (picaresca)			
Quien siembra vientos recoge tempestades	X (vientos)				X (recoge)
Un grano no hace granero, pero ayuda a su compañero					
La arquitectura es el arte de organizar el espacio	X (es)		X (arquitectura)		
El amor actúa con el corazón y no con la cabeza.			X (actúa)		
No dudes, actúa.	X (dudes)		X (actúa)		
El niño es realista; el muchacho, idealista; el hombre, escéptico, y el viejo, místico					
La música es sinónimo de libertad, de tocar lo que quieras y como quieras	X (es, quieras)				
La belleza que atrae rara vez coincide con la belleza que enamora.				X (belleza)	
No está mal ser bella; lo que está mal es la obligación de serlo.				X (bella)	
La batalla más difícil la tengo todos los días conmigo mismo.	X (más)			X (batalla)	
El que no llora, no mama.				X (llora)	
En la pelea, se conoce al soldado; sólo en la victoria, se conoce al caballero.			X (victoria)	X (caballero)	
La lectura es a la mente lo que el ejercicio al cuerpo.	X (es)		X (lectura)		

Figura 2.2: Frases conocidas

con la palabra ‘caballero’ para la regla 5: el fonema /ll/ se pasa a /i/ cambiando su duración y sonido. De esta forma cada frase tiene el mayor cubrimiento posible. En la Figura 2.2 podemos ver que es despareja la cantidad de frases utilizadas con respecto a sus reglas. 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 cuales van a ser las frases, ahora debemos definir qué frases y en qué orden se deben decir durante el experimento. Sucede que el orden que utilicemos va a ser crucial para obtener muestras: no es lo mismo empezar por una frase que solo cubre una sola regla que varias. Si elegimos primero las frases que cubren varias reglas a la vez, en un solo audio podremos obtener más cubrimiento de reglas.

¿Porqué quisiéramos cubrir mas 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 y solo con 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
3  Output: listaFrases
4  listaFrases = {}
5  DicPct <- Diccionario de porcentajes de cada regla
6  Mientras Frases != {}:
7      regla <- ObtenerReglaConMejorPorcentaje(DicPct)
8      frase <- Frases.ObtenerLaMasPonderada(regla)
9      listaFrases.agregar(frase)
10     RecalcularPorcentajes(DicPct)
11 Devolver listaFrases

```

La idea del algoritmo es la siguiente: vamos a utilizar un contador que nos va a decir cuántas muestras tenemos por cada regla. En cada paso vamos a ver ese contador y vamos a elegir la próxima frase teniéndolo en cuenta. Esta elección la lleva a cabo la función *ObtenerLaMasPonderada*. Esta se encarga de elegir la frase que haga referencia a la regla menos grabada y además que represente a más de una regla. De esa forma intentamos obtener la mayor cantidad de información posible con pocas grabaciones y ponderamos las frases que referencien a más reglas.

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 2.3 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 de alrededor del 40% en todas las reglas.

### 2.1.3. Combinando los dos tipos utilizando trazas

Definimos ambos grupos de frases a grabar. Ahora debemos definir como 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 ó 3 frases comunes extraídas del orden definido en *OrdenDe-*

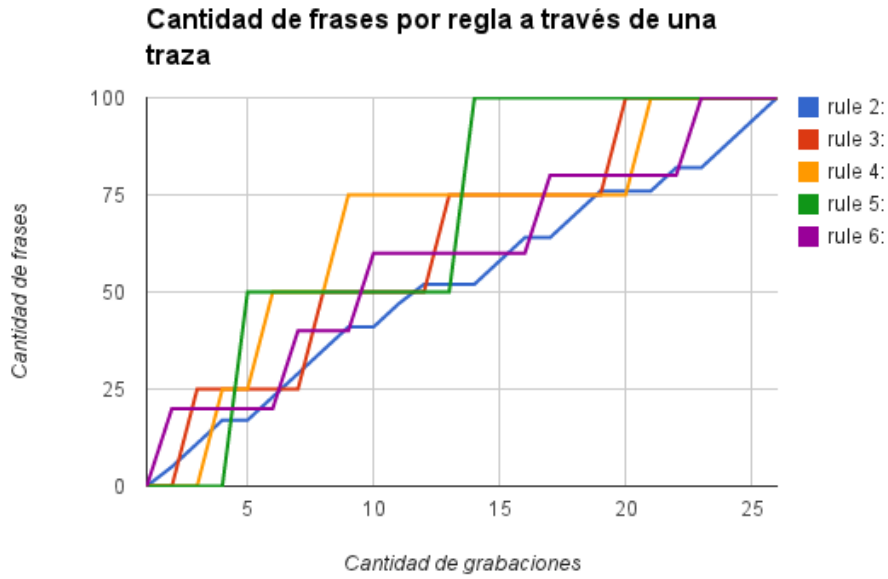


Figura 2.3: Cantidad de frases por traza

*FrasesConocidas*, 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 que frase va a tener que grabar.

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 como 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 nos da mucha ventaja 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 guardó la información de cada hablante, las frases a grabar y las trazas. La base de datos elegida fue PostgreSQL versión 9.1 y se eligió esta ya que es de código abierto. Los archivos de audio se guardaron en archivos *wav* por separado y se guardó una referencia al nombre del archivo generado en la base de datos. Para el servidor HTTP se utilizó Apache versión 2.2.22. El servidor utilizó el sistema operativo Ubuntu 12.04.4 LTS.

### 3.1. Recolección de datos

Cuando un usuario visitó nuestra página, primero debió 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 debió confirmar tener acceso al micrófono que posee en su dispositivo como se puede apreciar en la Figura 3.2. Una vez hecho esto, se le explicó las instrucciones como se ve en la Figura 3.3 y luego puede empezar a grabar.

Cada nuevo experimento utilizó una nueva traza del conjunto de trazas descriptas en la capítulo anterior. La interfaz que vio el usuario al grabar se puede ver en la Figura 3.4. Las grabaciones pueden ser escuchadas antes de ser confirmadas por



**¡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

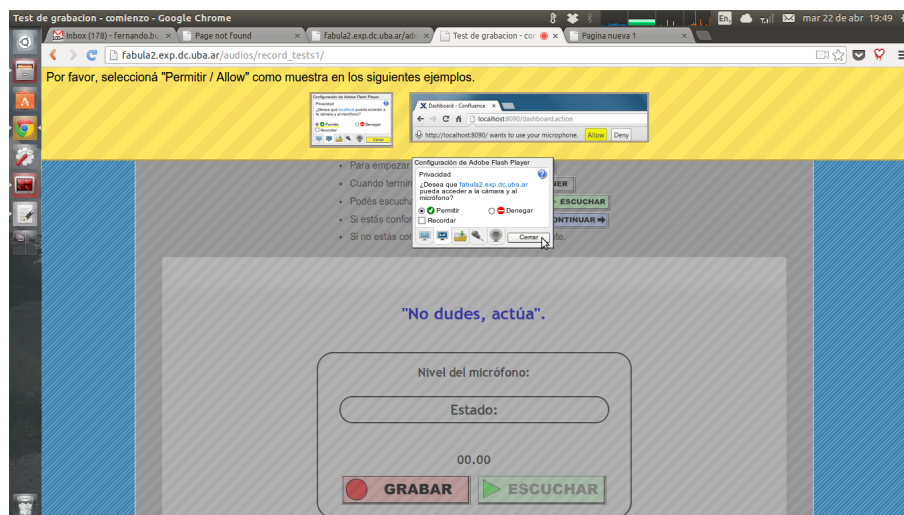


Figura 3.2: Se debe permitir micrófono para comenzar el experimento

el usuario. Lo importante es que la grabación se escuchen lo mejor posible. Para reproducir se aprieta en el botón *Reproducir* como se ve en Figura 3.5.

Una vez que el hablante chequeó que su grabación se escucha bien, la confirma. Cada vez que se graba un audio, esta 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 fueron grabando cada 5 frases. Una vez terminado estas 5 frases se le preguntó si quiere seguir grabando o terminar el experimento. De esta forma, aporta el tiempo que el hablante pudo disponer.



Figura 3.3: Inicio del experimento

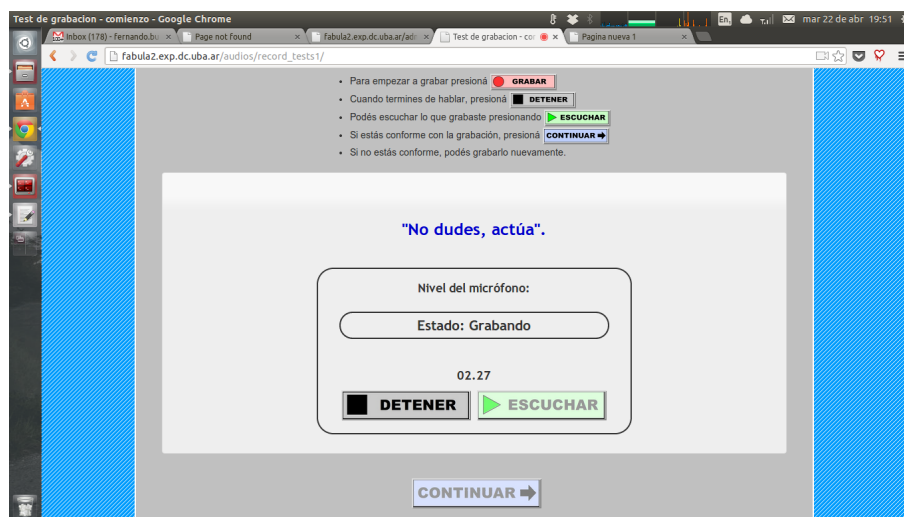


Figura 3.4: Grabando una frase

## 3.2. Grabación a través del browser

Los navegadores actuales no permiten acceder al micrófono directamente. Durante la tesis se desarrolló HTML5 que permitirá acceder al micrófono y a recursos similares de forma más fácil. No se eligió basarse en este porque sólo algunos browsers lo soportaban. 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 Interfaces <sup>1</sup> (WAMI). WAMI es una aplicación Flash que nos permite acceder al micrófono a través

---

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

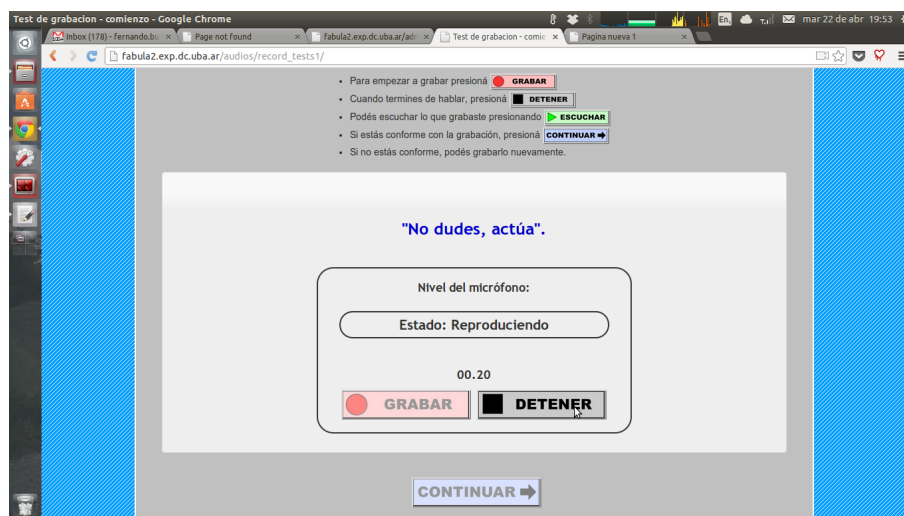


Figura 3.5: Reproduciendo la frase anteriormente grabada

de JavaScript. El proyecto WAMI es muy utilizado en proyectos similares 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 fueron básicos: 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 en la versión de Flash instalada. De todas formas el navegador Chrome posee preinstalado la última versión de Flash, quiere decir que este navegador puede correr perfectamente la aplicación sin importar el sistema operativo que se utilice.

## 3.3. Varias grabaciones por frase

Para tener la mejor grabación de cada frase, le dimos la opción al hablante que pueda escucharse 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 y para ver por qué el hablante se queda con 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. Este 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 a poder escuchar y marcar las grabaciones para utilizarlas como primer filtro de las grabaciones. Explicaremos esto en mas detalle a continuación.

### 3.4.1. Etiquetando audios

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 escucharon los audios y según su calidad los etiquetaron con alguna de las etiquetas definidas. Las etiquetadas utilizadas esta vez son: ‘Conservar’, ‘Sonido saturado’, ‘Mucho ruido de fondo’, ‘Problemas en el habla’. Esto se puede ver en Figura 3.6.

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 de tipo tar. Entonces si quisiéramos bajarnos todos los audios etiquetados 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 aquí.

## 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 los audios con una carpeta externa de backup.

**Audio 6**


**Id:** 6

**Speaker:** 2

**Word:** No está mal ser bella; lo que está mal es la obligación de serlo

**Attempt:** 1

**Filename:**



download: [bsas\\_u2\\_t32\\_a1](#)

**Labels:**

☐ Conservar

☒ Sonido saturado

☐ Mucho ruido de fondo

☐ Problema en el habla

Figura 3.6: Categorizando audios

### 3.6. Análisis del volumen

Un objetivo 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 en 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 de vuelta el experimento. De esta forma podemos filtrar audios 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 no logre un ambiente beneficioso para grabar y, aunque quiera, el filtro rechace todos sus audios. También notamos que había grabaciones que dieron mal el filtrado del volumen pero la grabación era buena. Esto no lo queremos como primer experimento del framework. Por eso elegimos aceptar todos sus audios.

A continuación veremos como 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 estos audios correctamente? Un archivo Wav, similar al que genera en cada una de las muestras, posee muchísima información. Es por esto que debemos seleccionar correctamente qué partes de la información nos sirve y qué partes podemos descartar.

### 4.1. Alineación forzada

Una grabación a partir de una frase posee muchísima información. Debemos seleccionar que parte de esta grabación nos interesa y que 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 básicamente 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.

Un dato muy importante es que este etiquetado idealmente no debe tener que ser realizado con intervención de un humano. Ya que 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. Luego de buscar bastante encontramos una llamada ProsodyLab Aligner [5]. Su función es realizar alineaciones automáticas en cada uno de los audios de forma fácil. Va analizar uno por uno cada audio y mediante un diccionario determina en que momento se dijo cada fonema y palabra.

Como dijimos, el formato de archivo utilizado para devolver estas marcas es TextGrid. El problema de la alineación automática es un caso particular de la alineación de audios.

Una particularidad que se destaca 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 íntegramente en lenguaje Python (versión 2.5). Utiliza fuertemente HTK que 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. Los Modelos Ocultos de Markov [9] (en inglés HMM) tratan de predecir qué fonemas aparecen en cada parte de los audios utilizando las diferentes muestras analizadas y la lista de fonemas pronunciada en cada grabación. Por ejemplo: mediante este modelo matemático, el programa analiza en cuales grabaciones, de la misma frase, se produce un mismo patrón de sonido. Si sucede esto en todas las 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 estas grabaciones. Entonces, a través de muestras va prediciendo los fonemas de las grabaciones.

Los requisitos para utilizar esta herramienta son: una hora de grabación 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*<sup>1</sup> que nos prestó su diccionario pudimos utilizar esta herramienta. Un diccionario fonético es básicamente un listado con las palabras que utilizamos y su división en fonemas. Es importante esto ya que va a ser usado por el alineador para describir los fonemas de cada palabra en cada frase.

Una vez terminada la alineación, ProsodyLab Aligner nos genera un archivo donde muestra como fueron esas alineaciones utilizando un puntaje. Más adelante veremos como utilizar este archivo. En definitiva, el primer paso para la extracción de información es alinear con esta herramienta.

## 4.2. Extracción de atributos

La extracción de datos fue realizado utilizando el lenguaje Python. Elegimos ese 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 librería se utiliza para realizar cálculos matemáticos. Nosotros la utilizamos para tener buena precisión en el cálculo de los atributos.

Como dijimos, el primer paso antes de extraer atributos es alinear con el ProsodyLab Aligner. Este al finalizar una alineación nos devuelve un archivo donde se encuentra como fueron realizadas esas alineaciones. Este archivo se llama ‘SCO-

---

<sup>1</sup>Página web: <http://www.lis.secyt.gov.ar/>

*RES'* y en él se encuentra una lista de todos los audios seguidos de un valor. Este valor nos permite ver 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 muy distintos. Este puede ser el primer filtro para el extractor.

Ordenando los audios utilizando esta numeración notamos que los menores poseen alineaciones malas, entonces definimos un umbral arbitrario para el cual aceptar la alineación si esta se supera. Si bien este procedimiento es efectivo, notamos que se encuentran algunos falsos positivos, o sea archivos que tienen un buen punto de score pero la alineación es mala. Al tener pocas grabaciones no pudimos aceptar estos casos, debimos corregirlos uno por uno.

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 4.1).

La rutina principal del programa toma de a uno las grabaciones y les aplica un conjunto de funciones. Cada una de estas funciones calculan un atributo. Los atributos se pueden dividir en dos tipos: uno correspondiente a *atributos temporales* o calculados solamente utilizando el TextGrid; y otro a *atributos acústicos* utilizando no sólo el TextGrid sino también el calculo de MFCC, que veremos más adelante. Si el atributo esta presente en la grabación tendremos ese dato en la extracción, sino se dejará como nulo. Luego juntamos todos los resultados de estas grabaciones y generamos el archivo Arff.

El archivo Arff tiene por cada linea una grabación y seguido todos los resultados del calculo de los atributos separado por comas. Necesitamos utilizar este formato ya que es el 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:

### 4.2.1. Atributos temporales

Los atributos temporales corresponden a atributos sobre la duración de los fonemas y la 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 teniendo en cuenta la unidad en que se contabiliza: mientras uno va a tener en cuenta fonemas, el otro tiene en cuenta sílabas. Se realizan dos normalizaciones. La primera será utilizando la forma:

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



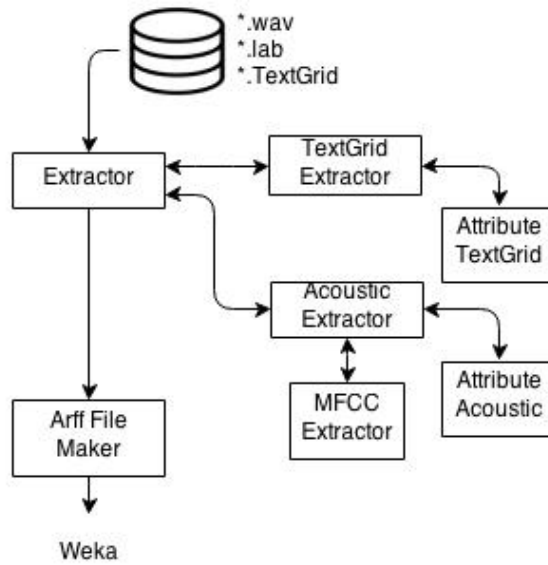


Figura 4.1: Diagrama workflow

- $\bar{X}$  es el promedio de duración del patrón buscado. Puede ser fonemas o sílabas.
- $\mu$  es el promedio de duración de la unidad utilizada.
- $\sigma$  es el desvío estándar de la unidad utilizada.

y luego la segunda asumiendo que  $\mu = 0$ .

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

Esta ultima tiene el nombre de Half-normal Distribution.

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 pierde esta relación ya que varía con respecto a la velocidad de la voz.

A continuación veamos cada uno de los atributos en particular para cada uno de los grupos y cómo se realiza su cálculo.

### Atributos fonéticos

Los atributos que contabilizan fonemas usados son:

- **Duración de la ‘kt’:** este atributo buscamos el patrón /kt/ en los TextGrids y luego a medir la duración del fonema /k/ en ese intervalo. Este atributo intenta medir la diferencia explicada en la regla 4 que nos indica medir dicho fonema.
- **Duración de la ‘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 lo medimos. Este atributo hace referencia a la regla 5 que mide dicho fonema.
- **Duración de la ‘rr’:** ídem para /r/ fuerte. Referencia a la regla 6 que hace hincapié en este fonema.
- **Duración de la ‘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 cantidad de fonemas y realiza un promedio. Este no se realiza normalización.
- **Duración de cada vocal:** contabilizamos cada vocal 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 la ‘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 tiempo 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 4.2 se puede ver una representación gráfica del TextGrid marcando los tiempo para cada fonema<sup>2</sup>. Se marca con rojo la duración donde aparece la ‘kt’.

Los valores de la normalización serán: para  $\mu$  se calculará como el promedio de duración de los fonemas en la frase en cuestión. Viendo la Figura 4.2 será el promedio de los tiempos marcados para todos los fonemas. Para  $\sigma$  será el desvío estándar de la duración de todos los fonemas de la frase. Y para  $\bar{X}$  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 rojo. La misma idea se aplica en el calculo de los demás atributos.

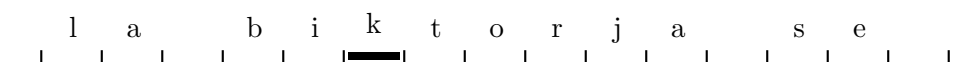


Figura 4.2: Ejemplo de cálculo de atributo

<sup>2</sup>Aclaración: la duración de los fonemas varía muchísimo. En el ejemplo se simplificó marcando todos los tiempos con el mismo tamaño para ser más simple la figura.

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, mediremos su duración y normalizaremos con las demás sílabas.
- **Duración de la sílaba anterior a la acentuada:** realizamos el mismo calculo anterior pero con la sílaba previa a la acentuada.

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.  $\mu$  representará el promedio de duración de las sílabas en la frase.  $\sigma$  será el desvío estándar de la duración de estas sílabas en la frase. Y finalmente  $\bar{X}$  será el promedio de duración de las sílabas anteriores a las acentuadas. Para cada uno de estos valores se calcula los dos tipos de normalización. En la Figura 4.3 podemos ver este ejemplo gráficamente<sup>3</sup>. No pudimos escribir toda la frase y todos sus acentos por cuestiones de espacio en la hoja.

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 4.3: Ejemplo de cálculo de atributo

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

### 4.2.2. Atributos acústicos

Los atributos acústicos utilizan las propiedades de los Wavs grabados. Para ello debimos extraer información con algún método que permita medirlos. Elegimos el calculo de MFCC ya que tiene relación directa con la percepción auditiva humana.

<sup>3</sup>Aclaración: al igual al caso anterior, la duración de las sílabas varia muchísimo. En el ejemplo se simplificó marcando todos los tiempos con el mismo tamaño para ser más simple la figura.

## Mel Frequency Cepstral Coefficients

La forma en que hablamos se produce por varias articulaciones. Algunas de ellas pueden ser: dientes, lengua, traquea etc. Estas articulaciones trabajan de forma tal para producir el sonido. Pero también funcionan para darle forma y aplicarle un filtro al sonido producido. Si sabemos correctamente que filtro se le aplica, podremos saber que sonido produce. La forma y el filtro asociado nos muestra donde esta la fuerza en el fonema. Este filtro es muy importante para entender la percepción humana. Los coeficientes MFCC se encargan de representar estos filtros. Veamos como se calculan.

Las señales de audio poseen muchas variaciones continuamente. En periodos cortos de tiempo, estas variaciones se reducen. Supongamos que dividimos cada audio en pequeños frames para calcular en ellos los coeficientes. El tamaño de cada frame esta entre 20-40 ms. Si la variación es menor que esta duración la descartaremos.

Luego para cada frame se calcula el espectro de frecuencia. Esto se viene motivado por un órgano que se encuentra en la oreja llamado Cóclea. Este vibra de diferente forma al llegarle cada frecuencia del sonido. Al vibrar, activa nervios que representan las distintas frecuencias que escuchamos. Dividir el sonido en períodos intenta mostrar que frecuencias están activas.

La Cóclea no reconoce diferencias entre dos frecuencias muy cercanas. Esto se incrementa mientras más alejada esta esa frecuencia. Para representar esta idea se utiliza un filtrado por escala de Mel. Esta escala es una aproximación de nuestra percepción. A frecuencias menores a los 1 Khz el filtro se comporta de forma lineal. A partir de ese valor, se comporta de forma logarítmica.

Mientras mas aumentamos la frecuencia, mas anchos son los filtros aplicados. Por ejemplo, ya en 4 Khz se aplican 20 filtros. Lo importante es ver cuanta energía hay en las frecuencias involucradas en el filtro. Luego que tenemos la energía de estos tramos le aplicamos la función logaritmo. De esta forma, para valores grandes de frecuencias su valor se decrementarán y no será igual que las pequeñas que poseen forma lineal. Esto se ajunta mejor a como escucha el oído. Para finalizar se computa DCT de las energías filtradas.

El siguiente pseudocódigo explica paso a paso como se calcula los coeficientes:

```

1 MFCC (Mel frequency cepstral coefficient):
2 1) Aplicar la derivada de Fourier de la se~nal. -> Espectro
3 2) Mapear las amplitudes del espectro a la escala mel.
4 3) Calcular el logaritmo.
5 4) Aplicar la transformada de coseno discreta (DCT).
6 5) Los MFCC son las amplitudes del espectro resultante.

```

Este algoritmo se calcula para un segmento del audio. Como dijimos, el audio se debe dividir en frames de 20 o 30 milisegundos pero avanzando 10 o 15 milisegundos. Es por ello que hay superposiciones en cada segmento. Al finalizar el algoritmo obtenemos 13 atributos acústicos de ese segmento. Podemos realizar la derivada de estos atributos y la segunda derivada para obtener más atributos. Estos atributos

corresponden a el estiramiento de los fonemas a través del tiempo. En total derivando dos veces llegan a 33 atributos acústicos.

Debemos extraer estas métricas para cada uno de los Wavs grabados. Este análisis tiene en cuenta como lo percibe un humano. Las frecuencias de Mel ayudan a describir la percepción humana del lado de las frecuencias que escucha.

## Implementación

Para realizar el calculo de estos coeficientes se utilizó un script en Matlab. El creador del script es Kamil Wojcicki y utiliza los 33 atributos utilizando 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 calcular el script y luego continuar con la extracción.

### 4.2.3. Nomenclatura utilizada

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

$$TIPO + \text{"_"} + ATRIBUTO + \text{"_"} + NORMALIZACIÓN$$

- *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  $(Min \mid Max \mid Avg) + (KT \mid LL \mid SC \mid RR)$ . Para hacer referencia a las reglas sobre atributos silábicos utilizamos los nombres de *syllableAccent* y *prevSyllableAccent* para la duración de la silaba acentuada y su anterior.
- *NORMALIZACIÓN* corresponde al tipo de normalización realizada. Estas pueden ser *norm* haciendo alusión a normalización o *normhd* haciendo alusión a normalización tomando  $\mu = 0$ .

Por ejemplo: definimos *SIL\_prevSyllableAccent\_normhd* como la duración de la silaba anterior a la acentuada aplicando normalización con  $\mu = 0$ . Todos los nombres y a que atributo se refieren se puede ver en el apéndice de atributos.

A continuación veamos 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. Los audios recolectados tuvieron algunos problemas al grabarse. El principal problema fue que el ambiente que utilizó cada hablante no estaba completamente en silencio como para hacer una buena grabación. Muchos errores surgieron en esa dirección. Otros errores comunes pero no tan frecuentes fueron: interpretaciones erróneas de la consigna, errores de volumen del micrófono, saturación etc..

### 5.0.4. Mediciones

Escuchamos los audios para determinar si se realizaron correctamente a través de la herramienta de administración que vimos en el capítulo 3. Los fuimos clasificando en: Conservar, Sonido saturado, Mucho ruido de fondo, Problema en el habla. Esta clasificación fue empírica, o sea no realizando ningún análisis sino que escuchando manualmente cada una. La cantidad de cada clase fue la siguiente:

	<b>Bs.As.</b>	<b>Cba.</b>	<b>Total</b>
<b>Conservar</b>	222	105	327
<b>Problemas en el habla</b>	33	15	48
<b>Mucho ruido de fondo</b>	2	12	14
<b>Sonido saturado</b>	2	0	2

Algo importante de ver es que los datos obtenidos están desbalanceados. No pudimos obtener la misma cantidad de audios para los dos grupos. Esto se va a reflejar en la clasificación y en el análisis posterior.

### 5.0.5. Errores comunes

Las categorías establecidas anteriormente describen los errores comunes mas frecuentes. Podemos observar que del total de 391 grabaciones, 64 tuvo algún problema.

Este es alrededor del 16 % de los audios grabados. Es un número alto para ser un experimento guiado.

La gran causa de este número es la faltante de un chequeo en el mismo momento que va grabando cada uno de los audios. Un trabajo futuro podría ser analizar el audio grabado y rechazarlo si no supera un nivel aceptable auditivo. Esto puede implementarse de varias formas. Una posible sería cuando esta grabando medir el volumen del micrófono cada una cierta cantidad de tiempo, por ejemplo 1 segundo. Si en esa medición el volumen no se encuentra entre rango máximo y mínimo de volumen, descartar el audio y pedirle al hablante que vuelva a grabar.

También se le podría dar mas información al hablante. Sabiendo que el micrófono tuvo un pico de volumen se podría pedirle al hablante que no hable tan fuerte. Ídem si habla muy bajo. Otras posibles soluciones a este problema es analizar antes de empezar el experimento si el sonido ambiente es muy alto o no. Y luego de ello aceptar una grabación nueva. Todas esas soluciones se pueden realizar en la aplicación web sin intervenir en el servidor.

Para análisis mas precisos se puede aplicar mejores filtros cuando llega la grabación del lado del servidor. Cuando llega el mensaje del audio al servidor, este ya puede obtener el wav y realizarle todo tipo de análisis mas precisos. Recordemos que el servidor esta implementado en Python que posee muchas librerías útiles para el análisis de audios. Al momento de terminar el análisis del audio en cuestión, deberá enviar la respuesta al hablante informándole si se debe realizar devuelta la grabación o si fue exitosa. Es importante notar que esta solución necesita buena conexión para el server.

Vamos a continuar con el análisis basándonos en los audios clasificados como Conservados. Luego trataremos de realizar algún análisis para reutilizar algunos de estos que tuvieron algún problema.

## 5.1. Alineación forzada

El alineador automático no realiza su función de forma perfecta. Sucede que muchas veces alinea mal. Es muy importante descartar los audios mal alineados ya que sino cuando los procese el extractor nos darían información errónea. Fuimos chequeando cada audio etiquetado como conservado y analizando si la alineación fue correcta con la aplicación Praat [1]. Los errores mas comunes fueron:

- **Ruido de fondo:** los casos donde el alineador se comporta de peor manera son los que se escuchan ruido de fondo. En esos casos las alineaciones resultan muy malas. Lamentablemente en nuestro caso esto es muy común.
- **Mouse click al finalizar:** sucede que el ambiente donde los hablantes realizaban las grabaciones no estaba bien aislado. Pasó en muchas oportunidades que el click de finalizar del mouse se grabó como parte final de la grabación

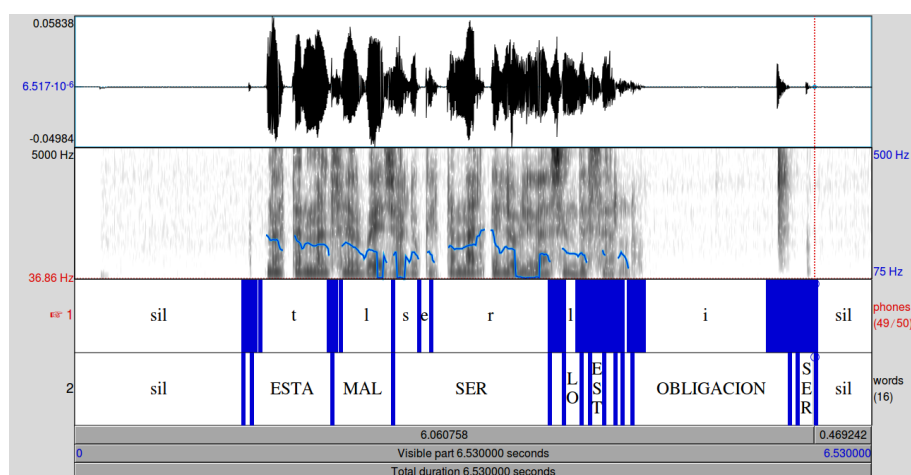


Figura 5.1: Ejemplo de alineación mala por ruido

(Ver Fig. 5.2). Ese sonido se grabó y afectó la alineación de forma tal que se tomaba como habla.

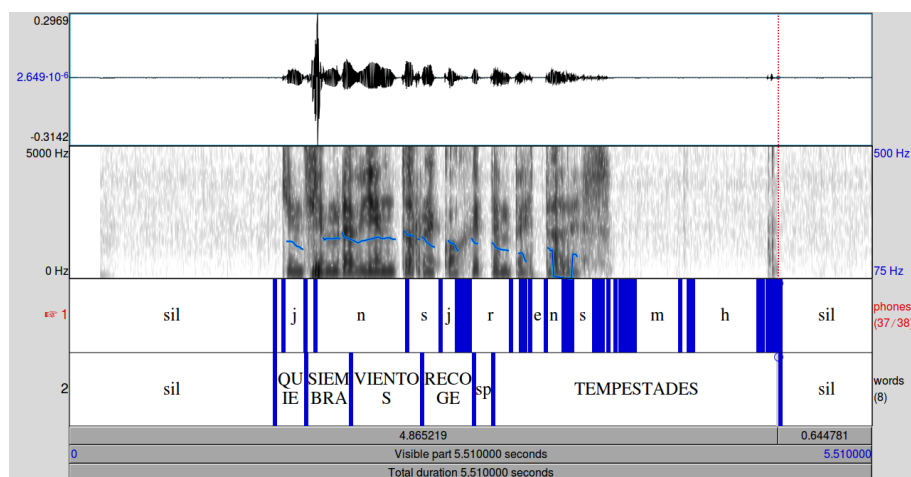


Figura 5.2: Click al final

- **Saturación del micrófono:** el volumen del micrófono es configurado por el hablante. Es por ello que debemos confiar en su buena voluntad. Sucede que muchas veces la grabación fue buena pero al final tuvo una entonación mas fuerte que las demás, haciendo que posteriormente la alineación no sea precisa.
- **Estiramiento de la /s/ final:** en varias oportunidades se quiso exagerar la entonación. Las frases finalizadas en /s/ fueron grabadas en muchos casos sosteniendo ese fonema por tiempo prolongado. A pesar de que fue alineado correctamente en toda su duración, este fue llevado a una duración entendible. El problema que surgió en estos casos fue que el hablante no supo pronunciar



la frase de la forma mas natural posible. Este fue el motivo por el cual se modificó.

## 5.2. Corrección de errores

Para corregir los errores descriptos debimos chequear cada uno de los TextGrids. Los resultados de la cantidad de TextGrid corregidos son:

	<b>Bs.As.</b>	<b>Cba.</b>	<b>Total</b>
<b>Modificados</b>	101	88	189
<b>Correctos sin modificación</b>	119	2	121
<b>Total</b>	220	90	310

Esta forma se realizó ya que eran pocos audios pero en un experimento con más hablantes se debe utilizar la opción de '.SCORES' antes descripta. De estos 310 debemos sacar las grabaciones que se realizaron varias veces. Recordemos que cada hablante tenía la posibilidad de grabar varias veces la misma grabación con la idea de que la última va a ser la mejor. Debemos quitar estos casos para no contabilizar más de una vez la misma grabación. Los audios repetidos son alrededor de 50. Quitando estos podemos empezar con el análisis.

# Capítulo 6

## Análisis

En esta sección mostraremos los resultados que obtuvimos luego de realizar la extracción. Primero presentaremos el baseline que consideramos. Luego describiremos el modelo de testing utilizando los datos recolectados. Explicaremos los clasificadores utilizados y en base a test estadísticos notaremos si aportan datos significativos. Por último, analizaremos los atributos más descriptivos de cada grupo.

### 6.1. Baseline

No hay ningún trabajo que trate de distinguir entre porteños y cordobéces a partir de su habla. Es por eso que el baseline se va a determinar a través del algoritmo majority class.

Imaginemos que tenemos un algoritmo que siempre elige un mismo grupo, por ejemplo Buenos Aires. Este algoritmo utilizando nuestros datos tendría una performance buena. En nuestro caso estará llegando al 45 % de efectividad aproximadamente. Este es el porcentaje a superar.

Cabe aclarar que si nuestro set de datos estuviera debidamente balanceado este porcentaje no sería tan alto. Lo ideal sería poder tener muestras balanceadas. Al tener este desbalance, puede suceder que al clasificar a un hablante en un test se obtenga mejores resultados para características de Buenos Aires que de Córdoba. Lamentablemente eso es una problemática de los datos obtenidos.

Utilizamos la herramienta de Machine Learning Weka para poder hacer el análisis. Esta nos provee un clasificador dummy descripto anteriormente. Este se llama ZeroR que es el algoritmo que siempre elige el grupo mayoritario.

### 6.2. Modelo de testing

La complejidad del problema y la forma en que fue realizado el experimento nos lleva a tener que descartar un modelo de testing común. Si utilizamos un modelo

estándar deberíamos dividir los audios en 2 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 en el de test. En ese caso el test sería erróneo ya que estaríamos entrenando con datos que luego serían testeados.

Para evitar este inconveniente debimos tomar en cuenta los hablantes a la hora de dividir los grupos. Dividimos a los hablantes en los 2 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 es muy variable. Un hablante pudo llegar a aportar 30 grabaciones mientras que otro sólo pudo aportar 5. Tomando en cuenta esto, la cantidad de audios de un grupo con respecto a otro puede quedar muy desbalanceada. Descartamos estos casos y tomamos que el conjunto de train tenga el 70 % de las instancias mientras que el restante 30 % sean destinadas para test. En definitiva, creamos 5 conjuntos de test de la forma  $\langle \text{train}, \text{test} \rangle$  con las características antes descriptas. A esto lo llamamos cross-validation test.

Otro problema a tener en cuenta es que los grupos de test generados sean lo más distintos posibles. Para solucionar eso el test de cross-validation será de la siguiente forma: Generamos conjuntos de 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 conjuntos utilizados.

A continuación el código generador del test cross-validation:

---

```

1  GeneradorDeTest:
2  Input: conjunto audios
3  Output: conjunto de  $\langle \text{train}, \text{test} \rangle$ 
4  resultado  $\leftarrow \{\}$ 
5  train  $\leftarrow \{\}$ 
6  test  $\leftarrow \{\}$ 
7  hablantesEnTests  $\leftarrow \{\}$ 
8  hablantes  $\leftarrow \text{ObtenerHablantes}(\text{audios})$ 
9  tamTest  $\leftarrow \text{tam}(\text{hablantes}) * 0.3$ 
10 Repetir 5 veces:
11     hablantesUsadosEnTest  $\leftarrow \text{ElegirRandom}(\text{hablantesEnTest}, \text{tamTest}$ 
         $\hookrightarrow * 0.2)$ 
12     hablantesNoUsadosEnTest  $\leftarrow \text{ElegirRandom}(\text{hablantes} -$ 
         $\hookrightarrow \text{hablantesEnTest}, \text{tamTest} * 0.8)$ 
13     hablantesTest  $\leftarrow \text{hablantesEnTest} + \text{hablantesNoUsadosEnTest}$ 
14
15     test  $\leftarrow \text{ObtenerAudios}(\text{hablantesTest}, \text{audios})$ 
16
17     train  $\leftarrow \text{ObtenerAudios}(\text{hablantes} - \text{hablantesTest}, \text{audios})$ 
18
19     Si  $\langle \text{train}, \text{test} \rangle$  no esta en resultado y
20         porcentajeMaximoDeSimilitud(test, resultado, 0.2) y
21         checkBalance(train) y checkBalance(test):
```

```

22         Agregar <train , test> a resultado
23         Agregar hablantesTest a hablantesEnTests
24     Devolver resultado
25
26     porcentajeMaximoDeSimilitud(conj , resultado , 0.2) :
27     Input: conj , conjunto de <train , test> , pct
28     Output: booleano
29     Recorrer test de conjunto de <train , test> :
30         pct ← Maximo(pct , porcentajeSimilitud(conj , test))
31     Devolver pct < 0.2
32
33     checkBalance:
34     Input: conj , pct
35     Output: booleano
36     bool_bsas ← #(conj) * 0.55 < #bsas(conj) < #(conj) * 0.65
37     bool_cba ← #(conj) * 0.45 < #cba(conj) < #(conj) * 0.45
38     Devolver bool_bsas y bool_cba

```

---

Vemos que en las líneas 11 y 12 agregamos los hablantes usados en los test y los nuevos hablantes todavía no utilizados. Luego de esos hablantes obtenemos los datos extraídos de sus audios en las líneas 15 y 17. Finalmente chequeamos las últimas características: que el par generado no sea uno ya previamente generado, que el conjunto test sólo tenga como máximo 20 % de elementos repetidos de los demás tests (línea 20 y 26) y que el balance de Córdoba y de Buenos Aires sea aproximadamente de un 40 % y 60 % respectivamente.

A continuación veremos los clasificadores que utilizaremos para predecir de qué lugar pertenece cada hablante.

### 6.3. Clasificadores

Vamos a entrenar varios clasificadores para poder determinar si el análisis de atributos que realizamos aporta mayor información a la hora de detectar un hablante. Los clasificadores propuestos son:

#### **Repeated Incremental Pruning to Produce Error Reduction (RIPPER)**

[2] - **Implementación JRip:** Este algoritmo divide el conjunto de entrada en pequeños grupos. Para cada uno va generando reglas que lo describan de forma golosa incremental. Luego cuando no puede describir más a ese grupo lo extrae y sigue con otro. Cuando estos conjuntos superan una cierta dimensión el algoritmo se detiene. Este algoritmo sirve mucho para datos no balanceados.

**C4.5 [8] - Implementación J48:** Este algoritmo genera un árbol de decisión que es muy usado para clasificación. Dada una serie de muestras con varios atributos: Para cada atributo calcula la ganancia de información. Elige el que tenga mejor

ganancia entre todos los atributos y con el crea un nodo en el árbol. Aplica recursivamente por cada rama. Si las muestras pertenecen a la misma clase o los atributos no proveen información se crea solo una hoja.

**Support Vector Machines [7] - Implementación Function SMO:** Support vector machines es un problema por el cual se intenta separar muestras en dos clasificaciones distintas a través de un hiperplano. Este hiperplano se construye utilizando los datos de entrada y sirve para clasificar las muestras en los dos grupos de forma óptima. Entonces utilizando este hiperplano, se puede etiquetar cada dato de entrada con su clasificación observando de qué lado del hiperplano se encuentra.

**Naive Bayes [10] - Implementación homónima:** Un clasificador de este tipo asume que cada atributo muestra una característica en particular de su clase pero no está relacionado con otro atributo. Cada uno de estos atributos contribuye de manera independiente a la clasificación de su clase. Utiliza fuertemente teorema de Bayes y que cada uno de los atributos es independiente.

## 6.4. Tests estadísticos

Utilizamos los resultados de cada clasificador para ver si son significativamente relevantes en la predicción de cada hablante. Los resultados que utilizamos son el vector resultante de entrenar con train y clasificar con test para los 5 grupos de test generados. Los clasificadores utilizados son ZeroR, o sea el baseline, comparado con algún clasificador más sofisticado, por ejemplo: JRip, J48, Function SMO, NaiveBayes. Para estos resultados vamos a realizar dos principales tests: Wilcoxon signed-rank y Test de Student.

### 6.4.1. Wilcoxon Test

Primero realizaremos Wilcoxon Test ya que necesita menos presunciones. Para realizar este test debemos cumplir que:

- Los datos son presentados de a pares y vienen de la misma población: esto sucede gracias a como generamos los tests. La población también siempre es la misma.
- Cada par es elegido al azar y independiente del resto: cada grupo generado para testing está armado de forma azarosa ya que la elección de cada hablante se realiza de esta forma.
- Los datos están medidos sobre una escala ordinal y no necesariamente debe provenir de una distribución Normal: esta característica es fundamental ya que no estamos seguros que nuestros datos provengan de una distribución Normal.

El input del mismo va a ser el vector resultante del test baseline ZeroR con el vector de los demás clasificadores. Las hipótesis van a ser:

Ho: Clasificador alternativo no es mejor que ZeroR

H1: Clasificador alternativo es mejor que ZeroR

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

Luego chequearemos si nuestra muestra es de distribución Normal. Si es ese el caso haremos el Test de Student. Para chequear Normalidad vamos a utilizar el test de Shapiro-Wilk.

#### 6.4.2. Análisis Shapiro-Wilk Test

El Test de Shapiro-Wilk lo utilizamos para poder afirmar si conjunto de datos proviene de una distribución Normal. Una característica importante es que posee un buen desempeño en pequeñas muestras.

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 se puede rechazar Ho y por ende se afirma que los datos siguen una distribución Normal.

Este test se realiza individualmente para cada vector resultado. O sea, debemos chequear 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 Shapiro-Wilk test un p-valor mayor 0,05 se puede realizar el Student Test.

#### 6.4.3. Student Test

Para los vectores que poseen una distribución Normal vamos a aplicarle este test. Este nos provee una forma de determinar si dos conjuntos de test son significativamente distintos. De la misma forma que planteamos la hipótesis de Wilcox test, este va a tener las mismas hipótesis. O sea:

Ho: Clasificador alternativo no es mejor que ZeroR

H1: Clasificador alternativo es mejor que ZeroR

La ventaja de usarlo es que, al saber que distribución representa, vamos a tener

resultados mas precisos. Aplicando el estadístico vamos a obtener 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, si hay evidencia y rechazamos  $H_0$ .

## 6.5. Resultados

Utilizamos un conjunto de test generado por el algoritmo anterior. 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:

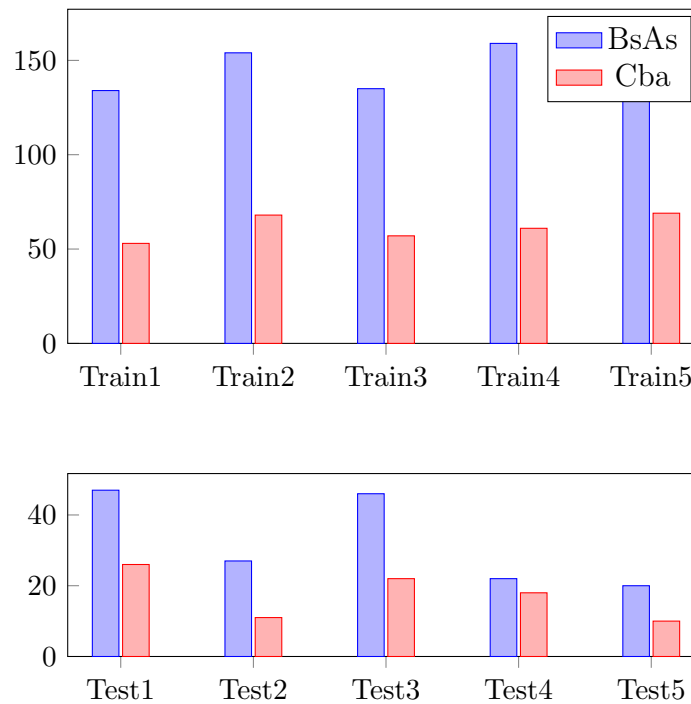


Figura 6.1: Cantidad de instancias de Buenos Aires y Córdoba según cada grupo de Train y Tests

Los resultados en porcentaje para los distintos clasificadores fueron:

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

Donde Test 1 corresponde al primer par  $\langle \text{train}, \text{test} \rangle$  y así sucesivamente. En la figura 6.2 se puede ver el porcentaje de cada tests en comparación. Excluimos a JRip y J48 por dar muy parecido a ZeroR.

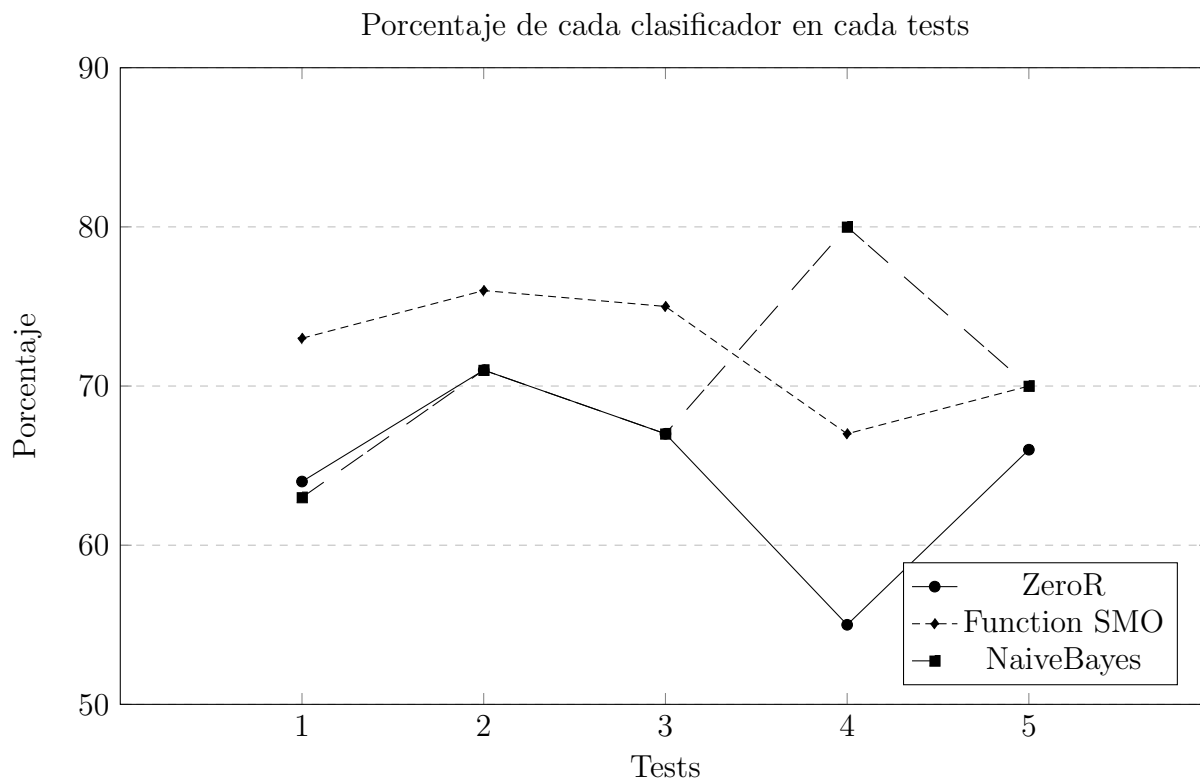


Figura 6.2: Porcentaje de ZeroR, Function SMO y NaiveBayes

Recordemos que el clasificador ZeroR elige siempre la clase mayoritaria en su grupo de test. Viendo la figura 6.2 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 Function SMO: se comporta muy similar a ZeroR pero en el test 4 posee mucha mejor performance. Viendo los grupos generados en la figura 6.1, el test 4 es el que tiene menos diferencia entre cantidad de hablantes de Buenos Aires y de Córdoba. Al tener mitad instancias de cada grupo y como ZeroR debe elegir en uno de esos dos, va a poseer un porcentaje de acierto de alrededor del 50 % como sucede.

El porcentaje de exactitud en la clasificación se puede apreciar con la métrica *Precision* y *Recall*. *Precision* se define como la cantidad de verdaderos positivos sobre la cantidad de verdaderos y falsos positivos. Tomamos como verdaderos positivos a la condición de que fue clasificado como Buenos Aires y efectivamente es de ahí. *Recall* se define como la cantidad de verdaderos negativos sobre la cantidad de verdaderos y falsos negativos. Estos valores 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

**Precision =  $22/40 = 0.55$ ; Recall = 1**  
**Instancias correctas = 55 %**

**Function SMO:**

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

**Precision =  $22/35 = 0.63$ ; Recall = 1**  
**Instancias correctas = 67 %**

**NaiveBayes:**

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

**Precision =  $20/26 = 0.77$ ; Recall =  $20/22 = 0.9$**   
**Instancias correctas = 80 %**

Viendo estas matrices de confusión y sus métricas podemos observar cual 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 no lo es). Esto sucede ya que elige solo una categoría siempre. En los demás clasificadores se intenta realmente predecir y por eso los Errores de tipo I y II están mas 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 test 0. Veamos para los clasificadores ZeroR y NaiveBayes.

**ZeroR:**

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

**Precision =  $47/73 = 0.64$ ; Recall =  $47/47 = 1$**   
**Instancias correctas = 64 %**

**NaiveBayes:**

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

**Precision =  $33/46 = 0.7$ ; Recall =  $33/47 = 0.7$**   
**Instancias correctas = 63 %**

En este caso podemos notar como a pesar de que den valores cercanos en instancias correctas, uno posee todo su error en un tipo solo mientras que el otro lo distribuye entre los dos tipos.

### 6.5.1. Wilcox y Student Test

Vamos a mostrar los resultados de estos tests estadísticos. Todos los tests estadísticos realizados fueron realizados utilizando R versión 3.0.1. Recordemos que los resultados surgen de realizar los test de Wilcox y Student para el vector resultado de cada clasificador con respecto a ZeroR.

	Student Test	Wilcox Test
<b>ZeroR y JRip</b>	0.8438	0.87
<b>ZeroR y J48</b>	0.9772	0.813
<b>ZeroR y NaiveBayes</b>	0.2113	0.1692
<b>ZeroR y Function SMO</b>	0.03125	0.004545

Todos los clasificadores pasaron el test Shapiro-Wilk, entonces podemos afirmar que los resultados de cada clasificador corresponden a una distribución Normal. Analizando estos resultados notamos que para el clasificador Function SMO poseen p-valor menor a 0,05 en ambas columnas. Esto quiere decir que **Function SMO tiene evidencia suficiente para ser mejor que ZeroR**. Por otro lado, los demás no pudieron lograr este cometido.

### 6.5.2. Clasificadores encontrados

Veamos los clasificadores obtenidos para analizar que atributos tuvieron en cuenta. Analizamos el clasificador JRip ya que de los clasificadores elegidos es el que utiliza una cantidad manejable de atributos. Los demás clasificadores utilizaron muchos más atributos y por ende debimos descartarlos ya que el análisis se tornaba engorroso.

Cada conjunto de test va a clasificar distinto y por ende utilizará una serie de reglas distinta. Estos datos corresponden a la clasificación *train0* y *test0*.

#### Clasificador JRip:

- $(FON_{ll\_norm} \leq -11,08) \text{ and } (ACU\_AverageLL\_6 \leq 4,308) \Rightarrow place = cba(12,0/0,0)$
- $(FON_{Sfinal\_normhd} \leq 27,874) \text{ and } (SIL\_prevSyllableAccent\_norm \geq -4,265) \Rightarrow place = cba(11,0/1,0)$
- $(FON_{rr\_normhd} \leq 31,355) \Rightarrow place = cba(10,0/2,0)$
- $else \Rightarrow place = bsas(154,0/23,0)$

Podemos notar en el anterior árbol de decisión que 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 su árbol de decisión nos permite pensar cuales atributos tienen mayor importancia a la hora de la clasificación.

Veamos con otro método cuales de los atributos aportan mayor información para la clasificación de los hablantes.

## 6.6. Selección de atributos de forma automática

En esta sección se aplicará a los distintos atributos evaluadores para analizar cual posee mayor importancia. El evaluador utilizado fue InfoGain.

#### Attribute Evaluator: InfoGain

Vamos a utilizar InfoGain para analizar la importancia de cada atributo y utilizaremos 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 que ese atributo se cumple. La ganancia de información de ese atributo es la resta de esos dos resultados. Esto se puede expresar como:  $InfoGain(Class, Attribute) = H(Class) - H(Class|Attribute)$ . De esta

forma, cuanto menor sea la entropía sabiendo ese atributo mayor será la ganancia de información para ese 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

Analizando los resultados vemos que los más preponderantes se refieren a la duración de consonantes, vocales, duración de la sílaba acentuada y su sílaba anterior. 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 primordial para distinguir los dos grupos. 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 sobre consonantes primordialmente. 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 de consonantes va a seguir respetándose. Son variables fuertemente correlacionadas.

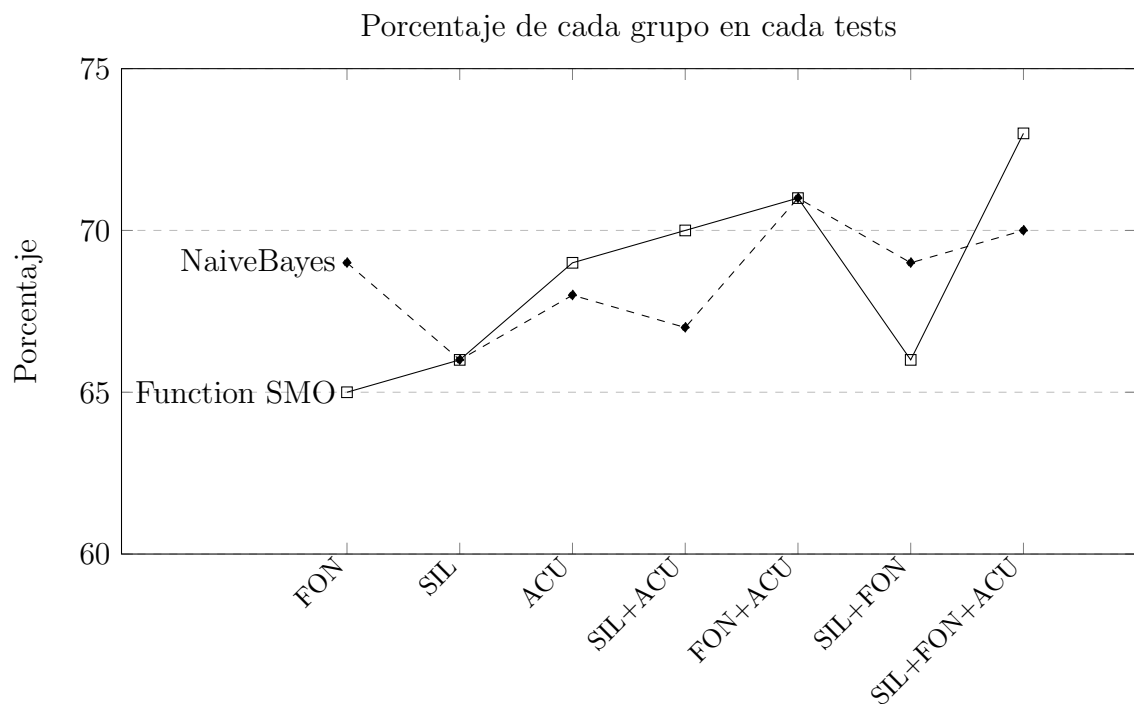
También algo que se desprende de este análisis es que todas las reglas del tipo fonéticas (empezadas con FON) y silábicas (empezadas con SIL) son sobre duración de tiempos. Sumar estos atributos y dividirlos en dos grupos es lo que logra los atributos sobre vocales y consonantes. Esta suma de atributos 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 o grabó pocas.

## 6.7. Combinando clases de atributos

Combinando los tipos de atributos definidos se puede apreciar cuanto 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 siguiente tabla se puede apreciar los resultados. Cabe aclarar que los valores de la tabla surgen 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



Lo esperable es que aumentando los atributos se aumenta el porcentaje de clasificación. 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 como en tipos de atributo el que posee mayor porcentaje son los atributos acústicos ya que se encuentran en los tres primeros grupos que obtuvieron mejor porcentaje. Quizás individualmente no tengan un atributo con mucha información predominante, pero la combinación de todos los atributos de esa clase hace que los clasificadores tengan buenas métricas.

## Capítulo 7

### Conclusiones

# Apéndice: marcas prosódicas

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

Frase	Prosodia
'no hay dos sin tres'	[[ 'no' ], [ 'aj' ], [ 'dos*' ], [ 'sin' ], [ 'tres*' ]]
'mas difícil que encontrar una aguja en un pajar'	[[ 'mas' ], [ 'di' ], [ 'fi*' ], [ 'sil' ], [ 'ke' ], [ 'eN' ], [ 'kon' ], [ 'trar*' ], [ 'una' ], [ 'a' ], [ 'Gu*' ], [ 'xa' ], [ 'en' ], [ 'un' ], [ 'pa' ], [ 'xar*' ]]
'mas perdido que turco en la neblina'	[[ 'mas' ], [ 'per' ], [ 'Di*' ], [ 'Do' ], [ 'ke' ], [ 'tur*' ], [ 'ko' ], [ 'en' ], [ 'la' ], [ 'ne' ], [ 'Bli*' ], [ 'na' ]]
'no le busques la quinta pata al gato'	[[ 'no' ], [ 'le' ], [ 'buh*' ], [ 'kes' ], [ 'la' ], [ 'kin*' ], [ 'ta' ], [ 'pa*' ], [ 'ta' ], [ 'al' ], [ 'ga*' ], [ 'to' ]]
'todo bicho que camina va al asador'	[[ 'to' ], [ 'do' ], [ 'bi*' ], [ 'cho' ], [ 'ke' ], [ 'ka' ], [ 'mi*' ], [ 'na' ], [ 'va' ], [ 'al' ], [ 'a' ], [ 'sa' ], [ 'dor*' ]]
'caminante no hay camino se hace camino al andar'	[[ 'ka' ], [ 'mi' ], [ 'nan*' ], [ 'te' ], [ 'no' ], [ 'aj' ], [ 'ka' ], [ 'mi*' ], [ 'no' ], [ 'se' ], [ 'ha*' ], [ 'ce' ], [ 'ka' ], [ 'mi*' ], [ 'no' ], [ 'al' ], [ 'an' ], [ 'dar*' ]]
'se te escapó la tortuga'	[[ 'se' ], [ 'te' ], [ 'eh' ], [ 'ka' ], [ 'po*' ], [ 'la' ], [ 'tor' ], [ 'tu*' ], [ 'Ga' ]]
'todos los caminos conducen a roma'	[[ 'to' ], [ 'Dos' ], [ 'los' ], [ 'ka' ], [ 'mi*' ], [ 'nos' ], [ 'kon' ], [ 'du*' ], [ 'cen' ], [ 'a' ], [ 'Ro*' ], [ 'ma' ]]
'no hay mal que dure cien años'	[[ 'no' ], [ 'aj' ], [ 'mal*' ], [ 'ke' ], [ 'du*' ], [ 're' ], [ 'cien*' ], [ 'a*' ], [ 'nos' ]]
'siempre que llovio paro'	[[ 'sjem*' ], [ 'pre' ], [ 'ke' ], [ 'Zo' ], [ 'Bjo*' ], [ 'pa' ], [ 'ro*' ]]
'cria cuervos que te sacaran los ojos'	[[ 'krj*' ], [ 'a' ], [ 'kwer*' ], [ 'Bos' ], [ 'ke' ], [ 'te' ], [ 'sa' ], [ 'ka' ], [ 'ran*' ], [ 'los' ], [ 'o*' ], [ 'xos' ]]
'la tercera es la vencida'	[[ 'la' ], [ 'ter' ], [ 'se*' ], [ 'ra' ], [ 'es' ], [ 'la' ], [ 'ben' ], [ 'si*' ], [ 'Da' ]]
'calavera no chilla'	[[ 'ka' ], [ 'la' ], [ 'Be*' ], [ 'ra' ], [ 'no' ], [ 'Hi*' ], [ 'Za' ]]
'la gota que rebalsó el vaso'	[[ 'la' ], [ 'go*' ], [ 'ta' ], [ 'ke' ], [ 'Re' ], [ 'Bal' ], [ 'so*' ], [ 'el' ], [ 'ba*' ], [ 'so' ]]
'la suegra y el doctor cuanto mas lejos mejor'	[[ 'la' ], [ 'swe*' ], [ 'Gra' ], [ 'y' ], [ 'el' ], [ 'dok' ], [ 'tor*' ], [ 'kwan' ], [ 'to' ], [ 'mas' ], [ 'le*' ], [ 'xos' ], [ 'me' ], [ 'xor*' ]]
'a la mujer picaresca cualquiera la pesca'	[[ 'a' ], [ 'la' ], [ 'mu' ], [ 'Cer*' ], [ 'pi' ], [ 'ka' ], [ 'reh*' ], [ 'ka' ], [ 'kwal' ], [ 'kje*' ], [ 'ra' ], [ 'la' ], [ 'peh*' ], [ 'ka' ]]

'quien siembra vientos recoge tempestades'	[[ 'kj*', 'en'], [ 'sjem*', 'bra'], [ 'bjen*', 'tos'], [ 'Re', 'ko*', 'Ce'], [ 'tem', 'peh', 'ta*', 'Des']]
'un grano no hace granero pero ayuda a su compañero'	[[ 'un'], [ 'gra*', 'no'], [ 'no'], [ 'ha*', 'ce'], [ 'gra', 'ne*', 'ro'], [ 'pe', 'ro'], [ 'a', 'yu*', 'da'], [ 'a'], [ 'su'], [ 'com', 'pa', 'ne*', 'ro']]
'la arquitectura es el arte de organizar el espacio'	[[ 'la'], [ 'ar', 'ki', 'tek', 'tu*', 'ra'], [ 'es'], [ 'el'], [ 'ar*', 'te'], [ 'de'], [ 'or', 'Ga', 'ni', 'sar*'], [ 'el'], [ 'eh', 'pa*', 'sjo']]
'el amor actua con el corazon y no con la cabeza'	[[ 'el'], [ 'a', 'mor*'], [ 'ak', 'tw*', 'a'], [ 'kon'], [ 'el'], [ 'ko', 'ra', 'son*'], [ 'i'], [ 'no'], [ 'kon'], [ 'la'], [ 'ka', 'Be*', 'sa']]
'no dudes actua'	[[ 'no'], [ 'du*', 'Des'], [ 'ak', 'tw*', 'a']]
'el nino es realista el muchacho idealista el hombre esceptico y el viejo mistico'	[[ 'el'], [ 'ni*', 'no'], [ 'es'], [ 're', 'a', 'lis*', 'ta'], [ 'el'], [ 'mu', 'cha*', 'cho'], [ 'i', 'de', 'a', 'lis*', 'ta'], [ 'el'], [ 'hom*', 'bre'], [ 'es', 'cep*', 'ti', 'co'], [ 'y'], [ 'el'], [ 'vie*', 'jo'], [ 'mis*', 'ti', 'co']]
'perro que ladra no muerde'	[[ 'pe*', 'Ro'], [ 'ke'], [ 'la*', 'Dra'], [ 'no'], [ 'mwer*', 'De']]
'la musica es sinonimo de libertad de tocar lo que quieras y como quieras'	[[ 'la'], [ 'mu*', 'si', 'ka'], [ 'es'], [ 'si', 'no*', 'ni', 'mo'], [ 'de'], [ 'li', 'Ber', 'taD*'], [ 'de'], [ 'to', 'kar*'], [ 'lo'], [ 'ke'], [ 'kje*', 'ras'], [ 'i'], [ 'ko', 'mo'], [ 'kje*', 'ras']]
'la belleza que atrae rara vez coincide con la belleza que enamora'	[[ 'la'], [ 'be', 'Ze*', 'sa'], [ 'ke'], [ 'a', 'tra*', 'e'], [ 'Ra*', 'ra'], [ 'Bes*'], [ 'kojn', 'si*', 'De'], [ 'kon'], [ 'la'], [ 'be', 'Ze*', 'sa'], [ 'ke'], [ 'e', 'na', 'mo*', 'ra']]
'no esta mal ser bella lo que esta mal es la obligacion de serlo'	[[ 'no'], [ 'eh', 'ta*'], [ 'mal*'], [ 'ser'], [ 'be*', 'Za'], [ 'lo'], [ 'ke'], [ 'eh', 'ta*'], [ 'mal*'], [ 'es'], [ 'la'], [ 'o', 'Bli', 'Ga', 'sjon*'], [ 'de'], [ 'ser*', 'lo']]
'la batalla mas dificil la tengo todos los dias conmigo mismo'	[[ 'la'], [ 'ba', 'ta*', 'Za'], [ 'mas'], [ 'di', 'fi*', 'sil'], [ 'la'], [ 'teN*', 'go'], [ 'to', 'Dos'], [ 'los'], [ 'dj*', 'as'], [ 'kon', 'mi*', 'Go'], [ 'mis*', 'mo']]
'el que no llora no mama'	[[ 'el'], [ 'ke'], [ 'no'], [ 'Zo*', 'ra'], [ 'no'], [ 'ma*', 'ma']]
'en la pelea se conoce al soldado solo en la victoria se conoce al caballero'	[[ 'en'], [ 'la'], [ 'pe', 'le*', 'a'], [ 'se'], [ 'ko', 'no*', 'se'], [ 'al'], [ 'sol', 'da*', 'Do'], [ 'so*', 'lo'], [ 'en'], [ 'la'], [ 'bik', 'to*', 'rja'], [ 'se'], [ 'ko', 'no*', 'se'], [ 'al'], [ 'ka', 'Ba', 'Ze*', 'ro']]
'la lectura es a la mente lo que el ejercicio al cuerpo'	[[ 'la'], [ 'lek', 'tu*', 'ra'], [ 'es'], [ 'a'], [ 'la'], [ 'men*', 'te'], [ 'lo'], [ 'ke'], [ 'el'], [ 'e', 'Cer', 'si*', 'sjo'], [ 'al'], [ 'kwer*', 'po']]
'el pez por la boca muere'	[[ 'el'], [ 'pes*'], [ 'por'], [ 'la'], [ 'bo*', 'ka'], [ 'mwe*', 're']]
'el canape salio espectacular'	[[ 'el'], [ 'ka', 'na', 'pe*'], [ 'sa', 'ljo*'], [ 'eh', 'pek', 'ta', 'ku', 'lar*']]



'el canape salio delicioso'	[[el'], [ka', na', pe*'], [sa', ljo*'], [de', li', sjo*', so']]]
'el canape salio riquisi- mo'	[[el'], [ka', na', pe*'], [sa', ljo*'], [Ri', ki*', si', mo']]]
'el repollo salio especta- cular'	[[el'], [Re', po*', Zo'], [sa', ljo*'], [eh', pek', ta', ku', lar*']]]
'el repollo salio delicioso'	[[el'], [Re', po*', Zo'], [sa', ljo*'], [de', li', sjo*', so']]]
'el repollo salio riquisi- mo'	[[el'], [Re', po*', Zo'], [sa', ljo*'], [Ri', ki*', si', mo']]]
'el esparrago salio espec- tacular'	[[el'], [eh', pa*', Ra', Go'], [sa', ljo*'], [eh', pek', ta', ku', lar*']]]
'el esparrago salio deli- cioso'	[[el'], [eh', pa*', Ra', Go'], [sa', ljo*'], [de', li', sjo*', so']]]
'el esparrago salio riquisi- mo'	[[el'], [eh', pa*', Ra', Go'], [sa', ljo*'], [Ri', ki*', si', mo']]]
'en boca cerrada no en- tran moscas'	[[en'], [bo*', ka'], [se', Ra*', Da'], [no'], [en*', tran'], [moh*', kas']]]
'mas vale pajaro en mano que cien volando'	[[mas'], [ba*', le'], [pa*', xa', ro'], [en'], [ma*', no'], [ke'], [sjen*'], [bo', lan*', do']]]
'la curiosidad mato al ga- to'	[[la'], [ku', rjo', si', DaD*'], [ma', to*'], [al'], [ga*', to']]]
'rio revuelto ganancia de pescadores'	[[Rj*', o'], [Re', Bwel*', to'], [ga', nan*', sja'], [de'], [peh', ka', Do*', res']]]
'no hay que pedirle peras al olmo'	[[no'], [aj'], [ke'], [pe', Dir*', le'], [pe*', ras'], [al'], [ol*', mo']]]

Cuadro 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

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

Cuadro 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 la 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

Cuadro 7.4: Atributos normalizados

# Bibliografía

- [1] Paul Boersma and David Weenink. Praat: doing phonetics by computer [computer program] version 5.3.51, retrieved 2 june 2013 from <http://www.praat.org/>.
- [2] William W. Cohen. Fast effective rule induction. In *Twelfth International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.
- [3] Elena Vidal de Battini. *Español en la Argentina*. Consejo Nacional de Educación, 1964.
- [4] Maria Beatriz Fontanella de Weinberg. *El español en la Argentina y sus variedades regionales*. Editorial Edicial S.A., Rivadavia 739 (1002) Buenos Aires, Argentina, 2000.
- [5] Kyle Gorman, Jonathan Howell, and Michael Wagner. Prosodylab-aligner: A tool for forced alignment of laboratory speech. In *Proceedings of Acoustics Week in Canada, Quebec City*, 2011.
- [6] Jorge A. Gurlekian, Reina Yanagida, Mónica Noemí Trípodí, and Guillermo Toledo. Amper-argentina: Variabilidad rítmica en dos corpus.
- [7] John C. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, ADVANCES IN KERNEL METHODS - SUPPORT VECTOR LEARNING, 1998.
- [8] Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.
- [9] L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, Feb 1989.
- [10] Harry Zhang. The optimality of naive bayes. In *FLAIRS Conference*, pages 562–567, 2004.