



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

1. Introducción	1
2. Diseño del experimento	4
2.0.1. Frases conocidas	4
2.1. Diseño teórico	4
2.1.1. Frases utilizadas	5
2.1.2. Utilizar frases con esquema AMPER	5
2.1.3. Utilizar frases conocidas	5
2.2. Generación de trazas	6
3. Sistema de grabación online	8
3.1. Requerimientos	8
3.2. Recolección de datos	9
3.3. Grabación a través del browser	9
3.4. Varias grabaciones por frase	10
3.5. Sistema de administración	11
3.5.1. Etiquetando audios	11
3.6. Filtrado de audios	11
4. Extracción de información	13
4.1. Alineación forzada	13
4.1.1. Prosodylab Aligner	13
4.1.2. Diccionario TranscriptorFonetico2 del LIS	14
4.2. Extracción de atributos	14
4.2.1. Atributos temporales	15
4.2.2. Atributos acústicos	16
4.2.3. Nomenclatura utilizada	18

5. Análisis	19
5.1. Baseline	19
5.2. Modelo de testing	19
5.3. Clasificadores	21
5.4. Tests estadísticos	22
5.4.1. Wilcox Test	22
5.4.2. Análisis Shapiro-Wilk Test	23
5.4.3. Student Test	23
5.5. Resultados	24
5.5.1. Wilcox y Student Test	27
5.5.2. Clasificadores encontrados	27
5.6. Selección de atributos de forma automática	28
5.7. Combinando clases de atributos	29
6. Conclusiones	31

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 al 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, por sobre todo, costoso. Además de estudiar cada grupo se debe utilizar muchos recursos: por ejemplo, se deben utilizar soporte para grabar en buena calidad las muestras, se debe realizar varios viajes para buscar los diferentes hablantes, se debe analizar cada uno de los audios de manera individual, entre otras cosas.

La motivación 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 descriptos el problema 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 posee muy baja calidad. De hecho, se utiliza en algunos experimentos donde esta característica no es un inconveniente.

El sistema desarrollado utiliza 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 carenciados que no posean una conexión este sistema no será útil. De cualquier forma, pensamos que su utilización soluciona muchos inconvenientes. Otra ventaja 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. Vamos a realizar un sistema para mejorar este proceso y diseñaremos un experimento para

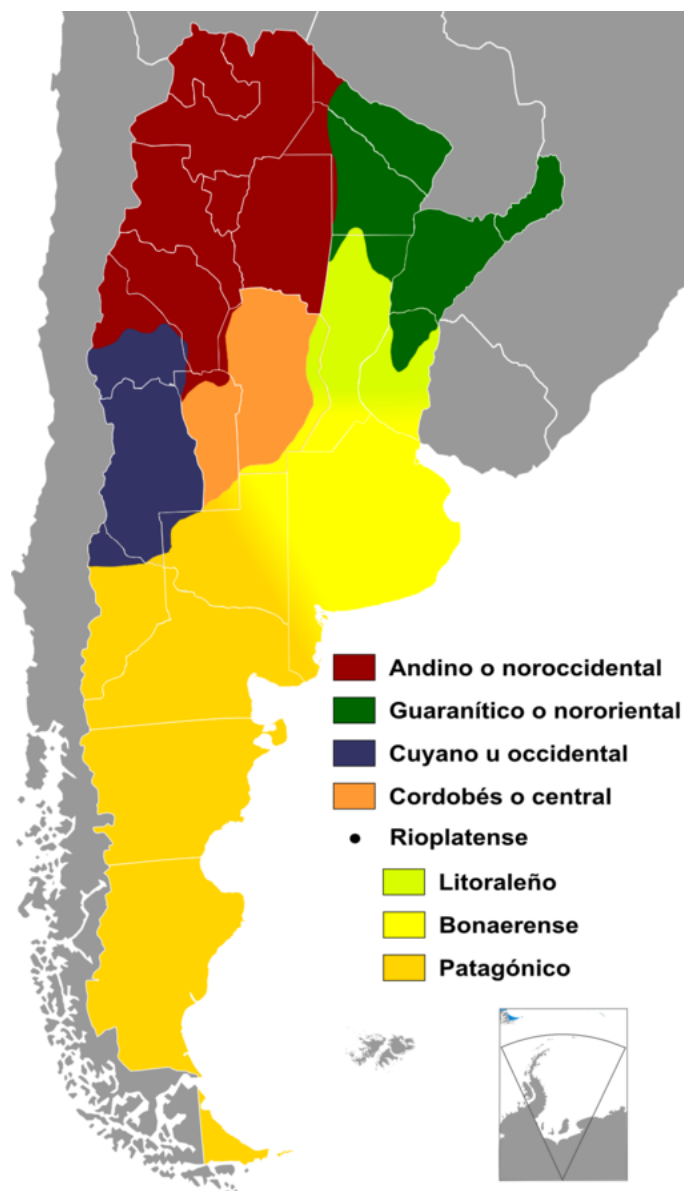


Figura 1.1: Dialectos del idioma español en Argentina

corroborar las ventajas y desventajas del mismo.

El experimento que tomamos como caso particular es las diferencias en el habla entre Córdoba y Buenos Aires. Estos dos grupos se encuentran uno en la zona central de nuestro país y el otro 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* [2] de Beatriz Fontanella de Weinberg y *Español en la Argentina* de Elena Vidal de Battini.

Fontanella de Weinberg recompila varios trabajos de colegas que analizan el español de cada región de Argentina. Cada región tiene un capítulo y entre ellas se encuentra una para Buenos Aires y otra para Córdoba. En la descripción de estos

capítulos las diferencias hacen hincapié en los sonidos más suaves y cortos de la /r/ y la /y/ 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.
- **Regla 2: Los hablantes de Córdoba aspiran y elisionan la /s/ al finalizar una palabra. No 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 en Buenos Aires** Palabras con el fonema /y/ o /ll/ se pronuncian /i/. Ejemplo: ‘lluvia’ se debe pronunciar utilizando el fonema /i/
- **Regla 6: En hablantes cordobeces la /r/ no debe vibrar 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.

Cabe destacar que estas reglas se producen en el habla espontánea. Tienden a no surgir en el 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 por de Buenos Aires. También describiremos cuales frases utilizamos y el medio empleado.

Capítulo 2

Diseño del experimento

Utilizando estudios previos de ambos dialectos, pude extraer dichas reglas que describen la diferencia entre cada uno de los dos grupos. Vamos a proponer realizar un experimento para poder extraer la información fonética de los mismos. El experimento va a poner foco primordialmente en las diferencias antes descritas. La idea sera realizar una serie de actividades donde el hablante sea grabado y esas actividades hagan hincapié en las diferencias. A continuación vamos a describir el experimento en más detalle.

2.0.1. Frases conocidas

Como el acento se potencia cuando se realiza habla espontánea, vamos a intentar que el hablante lo diga de forma lo más natural posible. Es por ello que se nos ocurrió como actividad pronunciar frases popularmente conocidas. La intención es que si el hablante conoce la frase y es frecuente de utilizarla entonces es más fácil que su pronunciación sea utilizando habla espontánea.

Vamos a querer utilizar frases lo mas conocidas posibles, para que el hablante ya tenga registrada como decirla sin poderle eliminar el acento.

2.1. Diseño teórico

Para realizar las grabaciones debemos darle a cada hablante una serie de frases a grabar. A esa serie de frases las llamaremos trazas.

Descripción de los 2 tipos de frases Vamos a tener dos formas de grabaciones:

- 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.

- Frases AMPER que tratan de cubrir el acento barriendo por cada tipo de acentuación. Estas corresponden a la regla 1 que hace incapié en la longitud de la sílaba anterior a la acentuada.

A continuación vemos las reglas en sus dos conjuntos.

2.1.1. Frases utilizadas

			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

2.1.2. Utilizar frases con 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 es la más conocida y puede aparecer de varias formas. Es por eso que este esquema nos va a resultar muy útil. Para tomar este esquema nos basamos en el trabajo de [4]

Para el esquema AMPER se fija un patrón de estructura de frases y se va cambiando las palabras que utiliza. El esquema AMPER utilizado es:

Objeto + "salió" + Adjetivo

donde Objeto puede ser *Canapé*, *Repollo*, *Espárrago*. Adjetivo puede ser *espectacular*, *delicioso*, *riquísimo*. Utilizamos estas palabras ya que cubren por la acentuación de cada tipo de palabra, o sea pasa por agudas grave y esdrújula.

Por ejemplo: *El canapé salio 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. Es importante armar la mayoría de las combinaciones para obtener muchas variantes de donde se encuentra el acento. De esta forma poder obtener muchísimas variantes con respecto a la Regla 1, que nos decía estirar la sílaba anterior a la acentuada. Todas las combinaciones se pueden ver en la Figura 2.1.

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, 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

2.1.3. Utilizar frases conocidas

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 el 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* posee atributo para 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. Esta nos dice medir la duración de la *ll*. De esta forma cada frase extrae los atributos lo mas posible. En la Figura 2.2 podemos ver el desbalance entr la cantidad de frases utilizadas con respecto a sus reglas aplicadas. Más adelante veremos como impacta esto en las frases que vamos a pedir para grabar.

Intercalando los dos tipos: Intercalaremos las frases comunes con las frases de AMPER para obtener las trazas de frases a grabar. Vamos a agregar 4 o 5 frases comunes, luego agregar 1 o 2 frases del esquema de AMPER y así sucesivamente hasta completar todas. La idea es no cansar al hablante con frases repetitivas y evitar que sepa de antemano que frase va a tener que grabar.

Siempre cada 5 grabaciones: 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 a las 40 que es el total de grabaciones.

2.2. Generación de trazas

Definimos como va a ser el esquema general de las trazas, ahora debemos definir qué frases y en que orden se debe decir durante el experimento. Sucede que el orden que utilicemos va a ser crucial para tener muestras. No es lo mismo empezar por una frase que sólo referencia a una regla que a varias. Si referencia a varias reglas a la vez, en un sólo audio podremos sacarle más información.

Una traza es una lista de las frases que puede grabar un hablante. Un hablante al empezar se le dará una de estas y grabará sucesivamente en ese orden. Elegimos tener precalculada las trazas para evitar cálculos innecesarios a la hora de empezar el experimento. Es por eso que guardamos 10.000 trazas generadas. Entonces al ingresar un hablante nuevo al experimento se le dará una de estas trazas para que grabe.

La generación de trazas sigue el siguiente algoritmo:

```

1  GenradorDeTrazas:
2  Input: Frases
3  Output: listaFrases
4  listaFrases = {}
5  DicPct <- Diccionario de porcentajes de cada regla
6  Mientras Frases != {}:
7      regla <- Obtener regla con mejor porcentaje
8      CtoFrases <- frases.ObtenerDeLaReglaLasMasPonderadas(regla)
9      listaFrases.agregar(CtoFrases)
10     RecalcularPorcentajes(DicPct)
11 Devolver listaFrases

```

La idea del algoritmo es la siguiente: Al generar las trazas vamos a utilizar un contador que nos va a decir cuantas 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 lo lleva a cabo la función *ObtenerDeLaReglaLasMasPonderadas*. Esta se encarga de elegir la frase que haga referencia a la regla menos grabada en el contador y además que represente a mas 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 maximizamos la cantidad de información de cada frase y al hablante le hacemos perder menos tiempo realizando el experimento. Esto se puede ver en el gráfico 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 todas las reglas. Excluimos del gráfico la regla 1 ya que esta utiliza con las frases AMPER.

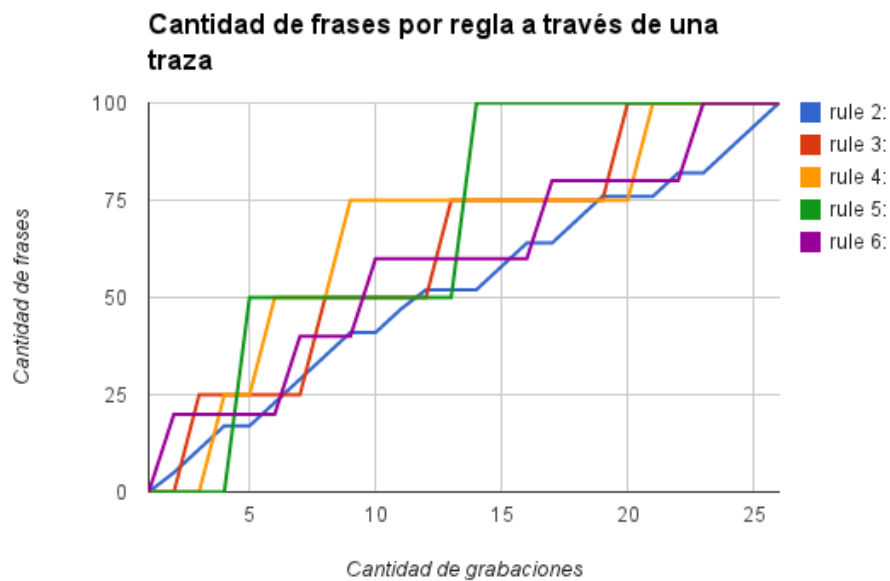


Figura 2.3: Cantidad de frases por traza

En conclusión, se grabarán de a 5 frases. Teniendo en cuenta que las primeras serán frases conocidas y las últimas 2 corresponderán al esquema AMPER.

Capítulo 3

Sistema de grabación online

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

La página web esta desarrollada en Django versión 1.4.2 que es un framework para la creación de paginas web. Se eligió este framework por su facilidad a la hora de salvar objetos a la base de datos y también por la cantidad importante de librerías que posee Python. La versión utilizada de Python es 2.7.3. Esta misma necesita una base de datos para guardar cada clase de dominio. En esta base de datos vamos a guardar la información de cada hablante, las frases a grabar y las trazas principalmente. La base de datos utilizada fue PostgreSQL versión 9.1 y se eligió esta ya que es de código abierto. Los archivos de audio se van a grabar en archivos *.wav* por separado y vamos a guardar 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 corre en el sistema operativo Ubuntu 12.04.4 LTS.

3.1. Requerimientos

Los requerimientos son básicos: micrófono, conexión a internet. Tuvimos problemas sobre el browser que 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 IOs, no tienen problemas en la versión de Flash instalada. De todas formas el navegador Chrome posee preinstalado dicha versión de Flash, entonces este navegador puede correr perfectamente la aplicación sin importar el sistema operativo.

3.2. Recolección de datos

Cuando un usuario visita nuestra página, primero le hace llenar un formulario. Este le pregunta: género, fecha de nacimiento y de qué lugar es oriundo. Al confirmar los datos, 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.



¡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 cumples estos requisitos, por favor completá los siguientes datos para comenzar:

Sexo:

Lugar donde te criaste:

Lugar donde vivís actualmente:

Mes de nacimiento: 01-1990

Figura 3.1: Encuesta inicial del sistema

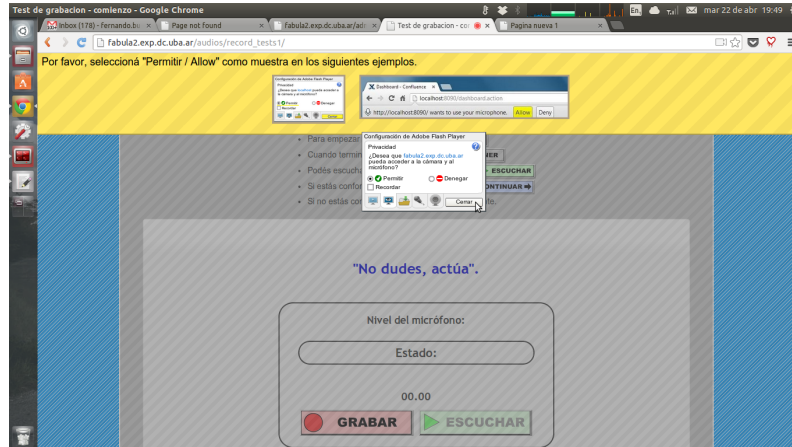


Figura 3.2: Se debe permitir micrófono para comenzar el experimento

En la pantalla de grabación el usuario deberá confirmar tener acceso al micrófono que posee en su dispositivo, como se puede ver en la Figura 3.2. Una vez hecho esto, se le explica las instrucciones (Fig. 3.3) y puede empezar a grabar. Cada nuevo experimento utiliza una nueva traza. El experimento en total consiste en realizar 40 grabaciones. El hablante va grabando de a 5 frases, cada vez que las termina de grabar ofrece la opción de seguir grabando otras 5 más. De esta forma, aporta el tiempo que puede. La interfaz que aprecia 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 cumples estos requisitos, por favor completá los siguientes datos para comenzar:

Sexo:

Lugar donde te criaste:

Lugar donde vives actualmente:

Mes de nacimiento: 01-1990

¡Empezar!

Figura 3.3: Inicio del experimento

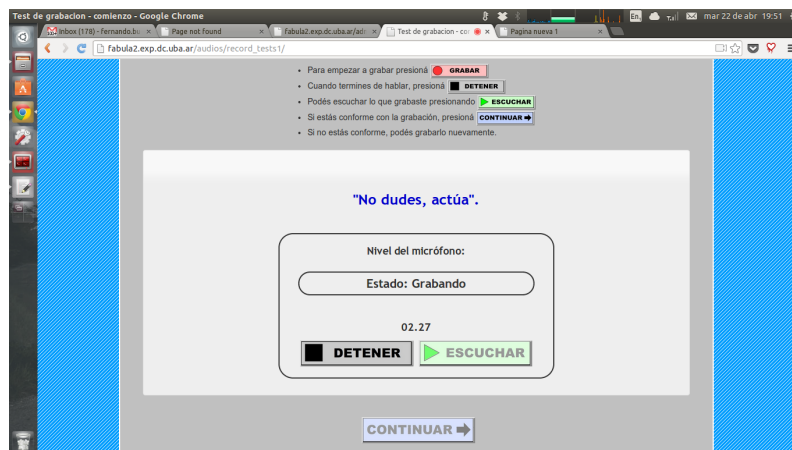


Figura 3.4: Grabando una frase



Figura 3.5: Reproduciendo la frase anteriormente grabada

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.

Cada vez que se confirma una grabación, esta se graba en un archivo wav en el servidor. El archivo wav que se genera tiene una frecuencia de muestreo de 22050 Hz y posee un sólo canal. Con estas características pudimos obtener un audio de calidad buena.

3.3. Grabación a través del browser

Los navegadores actuales no pueden soportar acceder al micrófono directamente. Durante la tesis, se desarrolla HTML5 que podrá soportar acceder al micrófono 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 eso excluiría gente. Es por eso que debimos utilizar alguna tecnología alternativa. Buscando encontré un proyecto llamado WAMI que es una aplicación Flash que nos permite acceder al micrófono a través de JavaScript.

El proyecto WAMI ¹ nos permite acceder al micrófono de la computadora utilizando Flash. Este posee una interfaz que permite utilizar la aplicación Flash desde Javascript. De esta forma, no es necesario analizar en la implementación. Utilizando dicha interfaz se puede configurar varias urls. Una de ellas es donde envía la información de lo grabado.

Cuando termina de grabar, se envía un mensaje POST al servidor. Este obtiene el paquete de información y lo guarda como archivo .wav. Cuando se quiere reproducir algún audio se envía un mensaje GET a través de la interfaz. El servidor lo responde con el audio requerido y se reproduce en el navegador.

Como se mencionó antes, para poder acceder al micrófono el usuario debe habilitar permisos para que la aplicación acceda al hardware de su computadora. Esto aparece como una ventana emergente al principio del experimento. Esto se puede ver en la Figura 3.2. También se puede configurar la calidad del audio grabado y analizar el nivel del volumen que posee. Se guarda un registro de nivel de volumen sobre la interfaz para utilizar esos datos en algún proyecto futuro.

3.4. Varias grabaciones por frase

Siguiendo con la idea de tener la mejor grabación de cada hablante, le dimos la opción a cada hablante de que después de grabar un audio de una frase puedan escucharse como quedó. Esto requiere un ida y vuelta entre el cliente (navegador) y el servidor. Al grabar, el cliente manda un mensaje HTTP POST para el servidor con los datos de grabación. Las frases son de corta longitud entonces no es necesario

¹<http://code.google.com/p/wami-recorder/>

preocuparse por la longitud del paquete. Cuando el cliente quiere escucharlo envía un mensaje GET a ese mismo audio anteriormente grabado. El servidor envía la grabación y es reproducida en el cliente. Esta 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 si es un punto débil del sistema.

A cada hablante le dimos la opción que pueda escuchar y volver a grabar la frase cuantas veces quisiera. Esto lo hicimos para poder detectar cual es el disparador que hace que diga mal una frase. Puede resultar interesante analizar los anteriores audios y porque se queda con el último.

3.5. Sistema de administración

El sistema debe tener datos precargados para estar activado recolectando hablantes. Los mínimos datos para poder tener la página funcionando en vivo son los datos de las trazas y la clasificación de los audios. Este cargo de datos se pueden guardar con fixtures del framework Django. Esta es una característica que nos permite guardar en un archivo datos para cargar antes de habilitar la web para estar online. Una vez cargado esos datos, los hablantes van a acceder a una url donde van a poder realizar el experimento grabando los distintos audios.

3.5.1. Etiquetando audios

Cuando varias personas terminan el experimento, los administradores pueden acceder a una url donde se puede escuchar cada audio que se va generando. Y si fue exitoso, o sea no esta saturado y no tiene ruidos, puede etiquetarlo con alguna de las etiquetas definidas. Las etiquetadas utilizadas esta vez son: ‘Conservar’, ‘Sonido saturado’, ‘Mucho ruido de fondo’, ‘Problemas en el habla’. Más adelante veremos como obtener estos audios. Esto se puede ver en Figura 3.6.

Backups automáticos El sistema posee backups automáticos generados a la noche automáticamente. Los backups consisten en un dump de la base de datos y de sincronización de los audios con una carpeta de backup. De esta forma, se guardan todos los datos cada día, y los audios quedan a salvo.

Export de wavs y csv Luego de obtener varios audios podemos exportar todos los datos a traves de urls. Utilizamos distintas urls que nos van a permitir bajarnos los audios etiquetados. Por ejemplo si accedemos a /conservados vamos a bajar los audios etiquetados de esa forma. Idem para las demas. Tambien se pueden bajar todos los audios sin etiquetas. Sobre la base de datos, se pueden bajar todo el modelo de datos a formato csv.

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. Filtrado de audios

Debemos evitar grabar audios saturados. Para ello se nos ocurrió medir el volumen de la grabación cuando sucede la misma. El resultado es una serie de valores entre 0 a 100. Sobre estos valores vamos a calcular el máximo y el mínimo. Si el primero es mayor a un cierto umbral (o sea mayor a 80) quiere decir que en la grabación se saturó en algún momento. Si el mínimo es menor a un cierto umbral (o sea menor a 20 por ejemplo) quiere decir que hay mucho sonido ambiente. En cualquiera de los dos casos podemos pedirle al usuario que grabe devuelta el experimento. De esta forma podemos filtrar audios que no nos servirán para reconocer el acento.

Si bien esta característica esta fue programada, no fue utilizada en la recolección de datos. El motivo fue que queríamos chequear cuan bien funcionaba la herramienta sin filtros y con completa participación de los usuarios. Otro motivo fue la paciencia de los hablantes. Puede suceder que al tratar de grabar no logre un ambiente beneficioso para grabar. Esto quiere decir que aunque quiera grabar 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.

Para los audios que salieron saturados, más adelante vamos a realizar un estudio para intentar sacarles el ruido y de esta forma intentar rescatarlos.

Capítulo 4

Extracción de información

Utilizando nuestra página web se van a poder obtener distintas muestras de Córdoba y Buenos Aires. Pero ¿cómo podemos analizar estos audios correctamente?. Un archivo wav, como los que captura la página cada una de las pruebas, posee muchísima información. Es por esto que debemos seleccionar correctamente que partes de la información nos sirve y que partes podemos descartar.

4.1. Alineación forzada

Un dato muy importante es que esta selección idealmente no debe tener que ser realizada con intervención de un humano. Ya que, si tenemos muchos audios tendríamos que hacerlo uno por uno y sería un trabajo muy arduo. De esto se encarga la alineación forzada. Las partes que debemos extraer de los audios son las partes donde se encuentran las diferencias de cada regla descripta anteriormente. Debemos tener una herramienta que nos permita obtener esos pequeños pedazos de audios para analizar sus diferencias.

4.1.1. Prosodylab Aligner

Luego de buscar bastantes, encontramos con una herramienta llamada ProsodyLab Aligner [3]. Su función es realizar alineaciones automáticas en cada uno de los audios de forma fácil. O sea, va a analizar cada audio y mediante un diccionario determina cada fonema en que momento se dijo. El formato utilizado para devolver estas marcas es el TextGrid. El problema de la alineación es un sub-problema de la alineación automática.

Algo que destaca esta herramienta es que no necesita datos de entrenamiento. Sólo con una hora de grabación es suficiente para correrlo y obtener resultados. Otra ventaja es que puede utilizarse para cualquier idioma.

Esta herramienta está hecha íntegramente en lenguaje Python (ver. 2.5) y script de Linux. Utiliza fuertemente de HTK (ver. 3.4) y SoX (ver. 14.1.0). Para realizar

la alineación utiliza Modelos Ocultos de Markov [7] (en ingles HMM). Básicamente trata de predecir que fonemas aparecen en cada parte de los audios utilizando las diferentes muestras y la lista de fonemas pronunciada en cada grabación. Por ejemplo: mediante un modelo matemático el programa analiza que en muchas grabaciones de la misma frase se produce un mismo fonema. Ese fonema va a ser marcado de igual forma en el TextGrid de las tres grabaciones.

La hora de grabación la debíamos cumplir recolectando grabaciones de la página web. Esta meta era posible de realizar. La creación de un diccionario era más complicado, ya que debía ser en español. Gracias a *Laboratorio de Investigaciones Sensoriales* que nos prestó su diccionario pudimos utilizar esta herramienta. El diccionario fonético nos provee para cada palabra los distintos fonemas que la componen. Esta herramienta se utiliza para generar la lista de fonemas que aparecen en cada grabación. De esta manera, se puede realizar una alineación acorde al español.

4.1.2. Diccionario TranscriptorFonetico2 del LIS

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 para cada grabación.

4.2. Extracción de atributos

La extracción de datos fue realizado utilizando el lenguaje Python. Elegimos ese ya que es un lenguaje de fácil de programar y tiene muchas librerías útiles para este tipo de casos. Utilizamos una muy conocida llamada Numpy versión 1.6.1. Esta es una librería para el calculo preciso en aplicaciones científicas.

El extractor posee como input los archivos .wav y los archivos textgrid que corresponden a las alineaciones temporales de cada fonema en cada audio. El extractor se debe ejecutar después de la alineación realizada por Prosodylab Aligner. El workflow del extractor se puede ver en 4.1).

El ProsodyLab-Aligner al finalizar una alineación nos devuelve un archivo donde se encuentra como fueron realizadas esas alineaciones. Este archivo se llama '.SCORES' y en el 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 va a ser el primer filtro para el extractor. Ordenando los audios en esta escala notamos que los menores poseen alineaciones malas, entonces definimos un umbral para el cual aceptar y rechazar la alineación. 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.

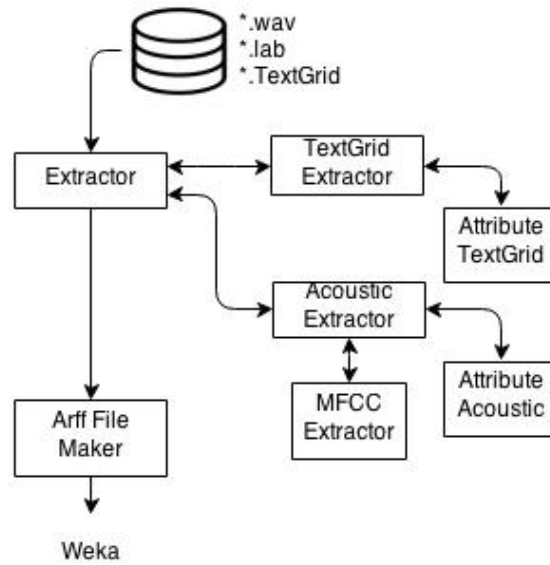


Figura 4.1: Diagrama workflow

Luego de esto, el extractor va a correr un conjunto de funciones que van a extraer cada uno de los atributos. Este conjunto de funciones se dividen en dos: las que computan atributos temporales y las que computan atributos acústicos. Veamos cada uno:

4.2.1. Atributos temporales

Corresponden a los atributos de duración en 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 lo miden.

Las mediciones son normalizadas de dos formas: primero una normalización utilizando:

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

y luego otra asumiendo que $\mu = 0$.

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

Esta ultima tiene el nombre de half normal distribution.

los atributos temporales se dividen en dos grupos: fonéticos y silábicos. Cada grupo calcula igual la normalización pero uno va a tener en cuenta fonemas y otro sílabas. Veamos cuales son:

Atributos fonéticos

- **Duración de la 'kt':** este atributo vamos a buscar el patrón /kt/ en los TextGrids y luego a medir la duración del fonema /k/. Este atributo intenta medir la diferencia explicada en la regla 4 que nos dice 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.

Atributos silábicos

Vamos a hacer un análisis de la sílabas. Los atributos que usamos son:

- **Duración de la sílaba acentuada:** en cada una de las frases vamos a buscar 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.

Estos atributos usamos para poder medir fuertemente la Regla 1, que esta es la más resaltada de la tonada cordobesa. 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.

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.

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.

Las señales de audio poseen muchas variaciones continuamente. En periodos cortos de tiempo estas variaciones se reducen. Vamos a dividir todo el 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 a este frame, habrá pocas pruebas para tenerla en cuenta y entonces 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. El audio se debe dividir en frames de 20 o 30 milisegundos pero avanzando 10 o 15 milisegundos. 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 datos de los wavs grabados. Para ello debemos analizarlos y que ese análisis nos de una medida que pueda compararse. El análisis debe tener 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:

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

donde:

- *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)*. Parahacer 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 ala acentuada aplicando normalización. Todos los nombres y a que atributo se refieren se puede ver en el apéndice de atributos.

Capítulo 5

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.

5.1. Baseline

No hay ningún trabajo que trate de distinguir entre porteños y cordobeces 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 mayor grupo.

5.2. 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. Vamos a dividir a los hablantes en los 2 conjuntos. Si bien esto evita el problema anterior, la cantidad de audios aportados por cada hablante es muy variable. Un hablante puede llegar a aportar 30 grabaciones mientras que otro sólo puede aportar 5. Tomando en cuenta esto la cantidad de audios de un grupo con respecto a otro puede quedar muy desbalanceada. Vamos a descartar estos casos y tomaremos que el conjunto de train va a tener el 70 % de las instancias mientras que el restante 30 % va a ser destinado para test.

Otro problema a tener en cuenta es que los grupos de test generados sean lo más distintos posibles. Para solucionar eso vamos a generar el test de cross validation de la siguiente forma: Iremos generando conjuntos de instancias para train y test, pero con una salvedad. Para armar el conjunto de test vamos a utilizar el 20 % de los elementos de las instancias ya utilizadas en los tests anteriores y el resto de instancias nuevas. De esta forma, vamos a regular la cantidad de instancias repetidas en los tests y nos aseguramos que sean todos distintos. Esto es para poder 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 <train , test>
4  resultado ← {}
5  train ← {}
6  test ← {}
7  hablantesEnTests ← {}
8  hablantes ← ObtenerHablantes(audios)
9  tamTest ← tam(hablantes) * 0.3
10 Repetir 5 veces:
11     hablantesUsadosEnTest ← ElegirRandom(hablantesEnTest , tamTest
        ↪ * 0.2)
12     hablantesNoUsadosEnTest ← ElergirRandom(hablantes -
        ↪ hablantesEnTest , tamTest * 0.8)
13     hablantesTest ← hablantesEnTest + hablantesNoUsadosEnTest
14
15     test ← ObtenerAudios(hablantesTest , audios)
16
17     train ← ObtenerAudios(hablantes - hablantesTest , audios)
18
19     Si <train , test> 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.

5.3. 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)

[1] - **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 [6] - 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 él 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, este es el caso base.

Support Vector Machines [5] - 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 que lado del hiperplano se encuentra.

Naive Bayes [8] - 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 el teorema de Bayes y que cada uno de los atributos es independiente.

Más adelante describiremos los parámetros utilizados para cada uno.

5.4. Tests estadísticos

Vamos a utilizar los resultados de cada clasificador para ver si los resultados son significativamente relevantes. Los resultados que vamos a utilizar van a ser el vector resultante de los 5 grupos de tests utilizando el clasificador ZeroR contrastado con algún clasificador más sofisticado, por ejemplo: JRip, J48, Function SMO, NaiveBayes. Para ello vamos a realizar dos principales tests: Wilcoxon signed-rank y Test de Student.

5.4.1. 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 armamos 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:

H_0 : Clasificador alternativo no es mejor que ZeroR

H1: Clasificador alternativo es mejor que ZeroR

donde 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 Si podemos rechazar H_0 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.

5.4.2. Análisis Shapiro-Wilk Test

El Test de Shapiro-Wilk lo utilizamos para notar la normalidad del conjunto de datos. Lo utilizamos ya que posee un buen desempeño en pequeñas muestras.

Planteamos como hipótesis nula que la población esta distribuida de forma normal, aplicamos el estadístico de este test y si el p-valor nos da mayor a 0,05 entonces la hipótesis nula es rechazada. Si en cambio es menor a 0,05 no se puede rechazar H_0 .

Este test se realiza individualmente para cada vector resultado. O sea, debemos chequear que los resultados de ZeroR para ver si su distribución se asemeja a la distribución Normal. Esto para cada resultado de los clasificadores. Si ambos tuvieron p-valor $\leq 0,05$, por ejemplo ZeroR y J48, se puede realizar el Student Test.

5.4.3. 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:

H_0 : 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 ser mas precisos a la hora de calcular su p-valor. 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 .

5.5. Resultados

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

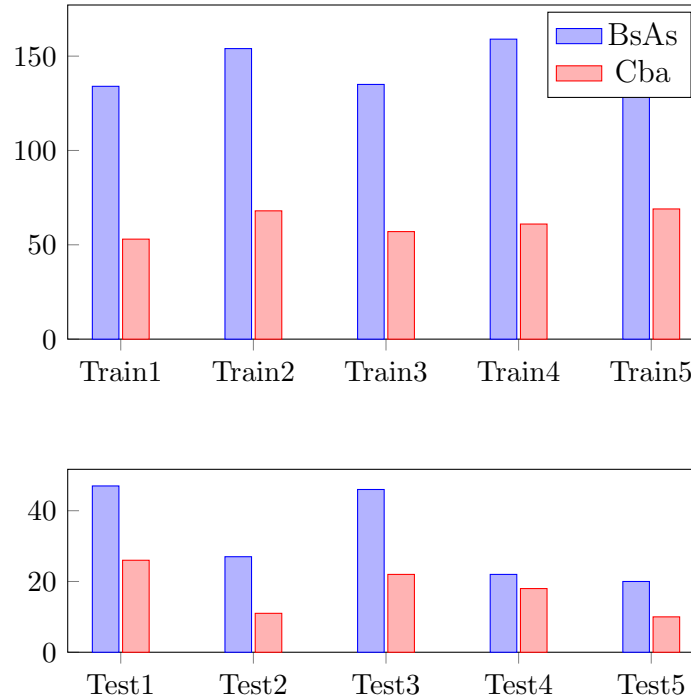


Figura 5.1: Cantidad de instancias de Buenos Aires y Córdoba según cada grupo de Train y Tests

Todos los tests realizados de aquí en mas fueron realizados utilizando R versión 3.0.1. Los resultados para los distintos clasificadores fueron:

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

Donde Test 1 corresponde al primer par (conjunto train, conjunto test) y así sucesivamente. En la figura 5.2 se puede ver el porcentaje de cada tests en comparación. Excluimos a JRip y J48 por dar muy parecido a ZeroR.

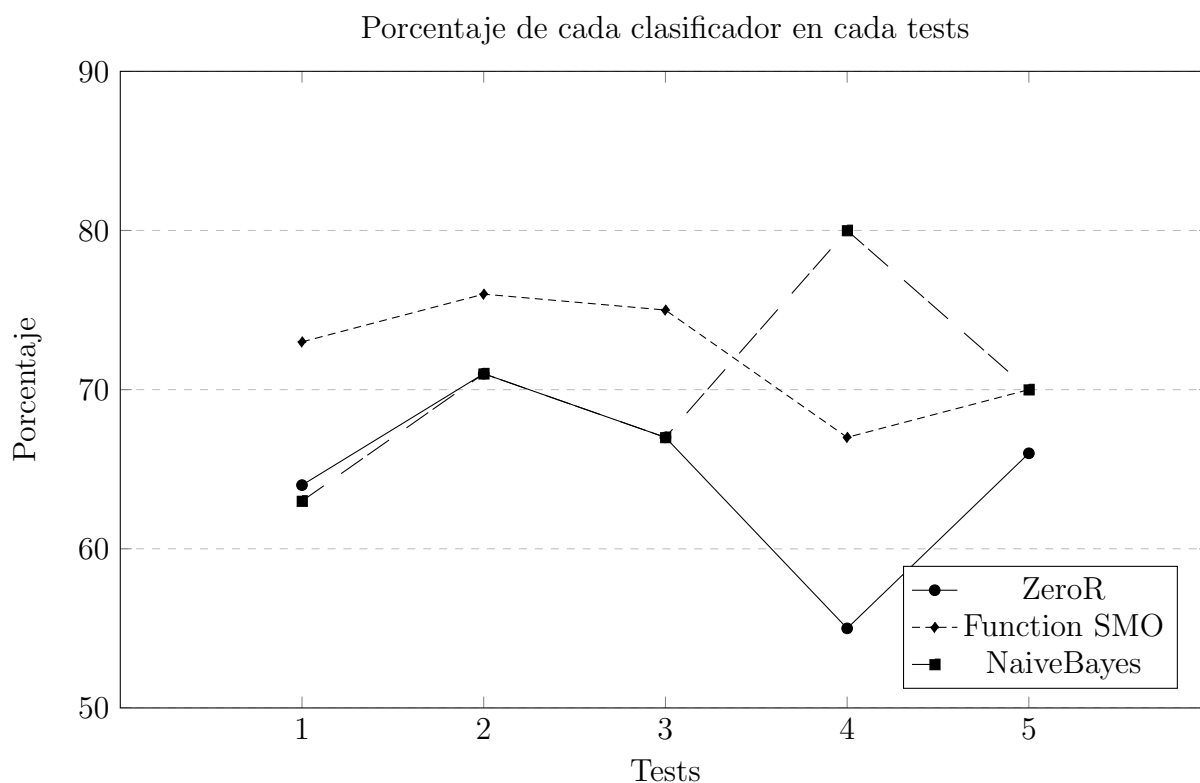


Figura 5.2: Porcentaje de ZeroR, Function SMO y NaiveBayes

Recordemos que el clasificador ZeroR elige siempre la clase mayoritaria en su grupo de test. Viendo la figura 5.2 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 5.1 es que de todos los tests 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.

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 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.

NaiveBayes:

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.

5.5.1. Wilcoxon y Student Test

Vamos a mostrar los resultados de estos tests estadísticos.

	Student Test	Wilcoxon Test
ZeroR y JRip	0.8438	0.87
ZeroR y J48	0.9772	0.813
ZeroR y NaiveBayes	0.2113	0.1692
ZeroR y Function SMO	0.03125	0.004545

Analizando estos resultados notamos que para el clasificador Function SMO poseen p – *valor* menor a 0,05. Esto quiere decir que el clasificador obtenido tiene evidencia suficiente para ser mejor que ZeroR. Por otro lado, los demás no pudo lograr este cometido.

5.5.2. Clasificadores encontrados

Veamos los clasificadores obtenidos para analizar que atributos tuvieron en cuenta. 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:

- $(FON_ll_norm \leq -11,08) \text{ and } (ACU_AverageLL_6 \leq 4,308) \Rightarrow place = cba(12,0/0,0)$
- $(FON_Sfinal_normhd \leq 27,874) \text{ and } (SIL_prevSyllableAccent_norm \geq -4,265) \Rightarrow place = cba(11,0/1,0)$
- $(FON_rr_normhd \leq 31,355) \Rightarrow place = cba(10,0/2,0)$
- $else \Rightarrow place = bsas(154,0/23,0)$

5.6. Selección de atributos de forma automática

En esta sección se aplicará a los distintos atributos evaluadores para analizar cual posee mayor importancia. Los evaluadores utilizados son InfoGain y ClassifierSubsetEvaluator.

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

Utilizando todos los atributos 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.

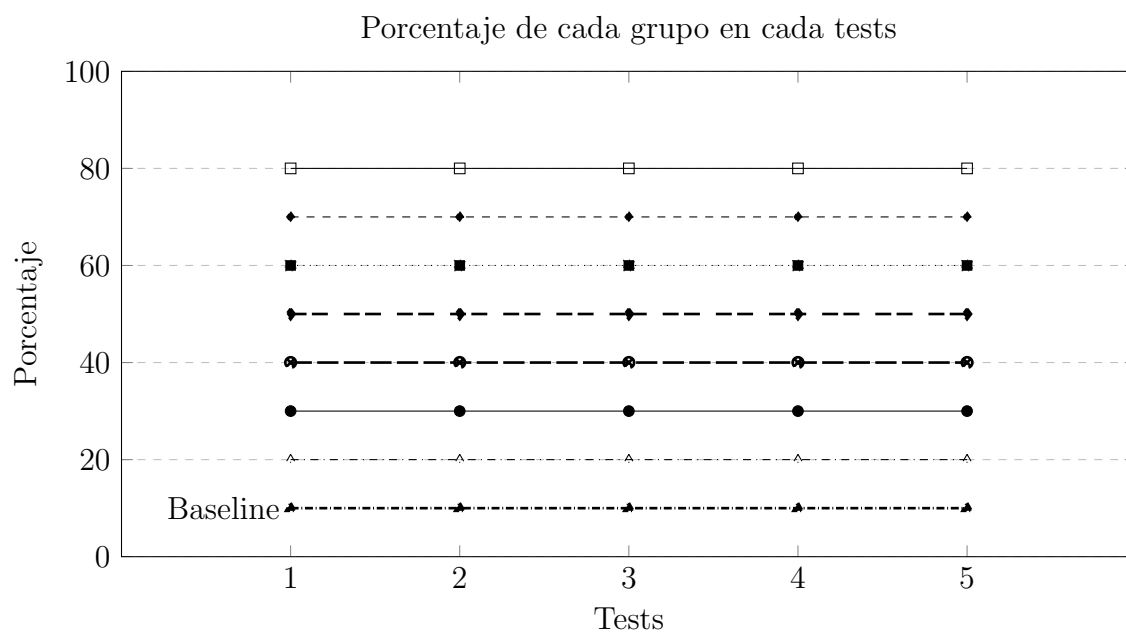
También algo que se desprende de este análisis es que todas las reglas del tipo fonéticas (empezadas con FON) y silábicas (empezadas con SIL) son sobre duración de tiempos. Sumar estos atributos y dividirlos en dos grupos es lo que logra los atributos sobre vocales y consonantes. Esta suma de atributos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

Los demás atributos son entendibles a la hora de analizar su ganancia de información.

5.7. Combinando clases de atributos

Combinando los tipos de atributos definidos se puede apreciar cuanto aporta cada clase de los mismos. Vamos a agregar los tipos de atributos y vamos a analizar como se incrementa el porcentaje de exactitud para la clasificación. El conjunto de datos utilizado es el *test0* y se utiliza *JRip* como clasificador.

	NaiveBayes
SIL + FON + ACU	59.5745
SIL + FON	57.4468
FON + ACU	61.7021
SIL + ACU	61.7021
SIL	53.1915
FON	57.4468
ACU	59.5745
Baseline	44.6809



Lo esperable es que aumentando los atributos se aumenta el porcentaje de clasificación. A grandes rasgos, aquí sucede pero dejando de lado el conjunto de atributos acústicos. Al agregar este conjunto la performance baja notablemente. Creemos que esto sucede por los distintos ambientes de grabación de cada hablante. Estos aportan mucho ruido y por ello no se puede extraer información útil para la clasificación.

Capítulo 6

Conclusiones

Apéndice: marcas prosódicas

En la figura 6.1 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 6.1: Marcas prosódicas

Apéndice: nomenclatura de atributos

En la tabla 6.2 se puede ver la nomenclatura de los atributos elegidos.

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

Atributos normalizados

En la tabla 6.3 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 6.3: Atributos normalizados

En la tabla 6.4 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 6.4: Atributos normalizados

Bibliografía

- [1] William W. Cohen. Fast effective rule induction. In *Twelfth International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.
- [2] 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.
- [3] 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.
- [4] Jorge A. Gurlekian, Reina Yanagida, Mónica Noemí Trípodi, and Guillermo Toledo. Amper-argentina: Variabilidad rítmica en dos corpus.
- [5] John C. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, ADVANCES IN KERNEL METHODS - SUPPORT VECTOR LEARNING, 1998.
- [6] Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.
- [7] L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, Feb 1989.
- [8] Harry Zhang. The optimality of naive bayes. In *FLAIRS Conference*, pages 562–567, 2004.