



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.

Palabras claves: Guerra, Rebelión, Wookie, Jedi, Fuerza, Imperio (no menos de 5).

AGRADECIMIENTOS

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce sapien ipsum, aliquet eget convallis at, adipiscing non odio. Donec porttitor tincidunt cursus. In tellus dui, varius sed scelerisque faucibus, sagittis non magna. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Mauris et luctus justo. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Mauris sit amet purus massa, sed sodales justo. Mauris id mi sed orci porttitor dictum. Donec vitae mi non leo consectetur tempus vel et sapien. Curabitur enim quam, sollicitudin id iaculis id, congue euismod diam. Sed in eros nec urna lacinia porttitor ut vitae nulla. Ut mattis, erat et laoreet feugiat, lacus urna hendrerit nisi, at tincidunt dui justo at felis. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Ut iaculis euismod magna et consequat. Mauris eu augue in ipsum elementum dictum. Sed accumsan, velit vel vehicula dignissim, nibh tellus consequat metus, vel fringilla neque dolor in dolor. Aliquam ac justo ut lectus iaculis pharetra vitae sed turpis. Aliquam pulvinar lorem vel ipsum auctor et hendrerit nisl molestie. Donec id felis nec ante placerat vehicula. Sed lacus risus, aliquet vel facilisis eu, placerat vitae augue.

A mi viejo.

Índice general

1..	Introducción	1
2..	Diseño del experimento	5
2.0.1.	Frases conocidas	5
2.1.	Diseño teórico	5
2.1.1.	Frases utilizadas	5
2.1.2.	Utilizar AMPER	5
2.1.3.	Utilizar frases conocidas	7
2.2.	Generación de trazas	7
3..	Arquitectura del sistema	9
3.1.	Recolección de datos	9
3.2.	Grabación a través del browser	9
3.3.	Requerimientos	10
3.4.	Sistema de administración	10
3.4.1.	Etiquetando audios	10
3.5.	Filtrado de audios	11
3.6.	Varias grabaciones por frase	11
4..	Extracción de datos	13
4.1.	Alineación forzada	13
4.1.1.	Prosodylab Aligner	13
4.1.2.	Diccionario TranscriptorFonetico2 del LIS	13
4.2.	Extracción de atributos	14
4.2.1.	Implementación	14
4.2.2.	Atributos temporales	15
4.2.3.	Atributos acústicos	16
5..	Datos obtenidos	19
5.0.4.	Mediciones	19
5.0.5.	Errores comunes	19
5.1.	Alineación forzada	20
5.1.1.	Errores comunes	20
5.2.	Corrección de errores	21

5.2.1. Disminución del ruido con Sox	21
6.. Análisis	23
6.1. Baseline	23
6.2. Modelo de testing	23
6.3. Clasificadores	24
6.4. Tests estadísticos	25
6.4.1. Wilcox Test	25
6.4.2. Análisis Shapiro-Wilk Test	25
6.4.3. Student Test	26
6.5. Resultados	26
6.5.1. Clasificadores	26
6.5.2. Wilcox y Student Test	27
6.6. Selección de atributos de forma automática	27
7.. Conclusiones	29
8.. Trabajos futuros	31
9.. Apéndice: marcas prosódicas	33

1. INTRODUCCIÓN

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. 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 y conocer mejor su cultura.

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 y 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 afecta.

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 para lugares alejados puede ser muy contraproducente. De cualquier forma, pensamos 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 corroborar las ventajas y desventajas de utilizarlo.

El experimento que tomamos como caso particular es las diferencias en el habla entre Córdoba y Buenos Aires. Ya se encuentran estudios que explican estas diferencias. Algunos de ellos son: Fontanella y Vidal (contar que dicen.)

En el libro *El español en la Argentina*, Beatriz Fontanella de Weinberg recompila varios trabajos de colegas donde 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 hace hincapié en los sonidos más suaves y cortos de la /r/ y /y/ y en la aspiración de la /s/. También afirma el estiramiento de la sílaba anterior a la acentuada en cada palabra como distintivo del acento.

En el libro *Español en la Argentina*, Elena Vidal analiza región por región el uso de los fonemas importantes. Destaca la diferencia entre las dos regiones de la /r/, /s/ y de la

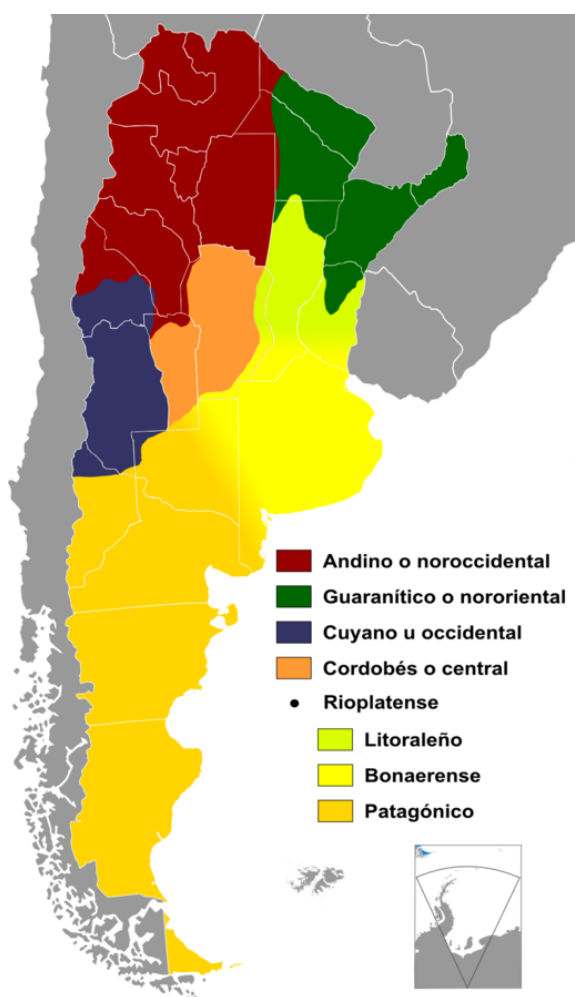


Fig. 1.1: Dialectos del idioma español en Argentina

/ll/. También referencia a la pronunciación de la /s/.

Extrayendo el análisis de estos libros pude definir las reglas que describen a cada grupo. Estas son:

Regla 1: Localice la sílaba acentuada en la palabra y estirar la sílaba anterior

Cada palabra posee su sílaba acentuada. 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: Aspiración y elisión de /s/

Para las palabras terminadas en /s/ se debe acortar su duración para 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: La 's' antes de la 'c' o 't' suenan suaves

La sílaba /s/, que precede a /c/ o /t/, debe sonar suave.

Ejemplo: 'Mosca' en el dialéctico de Córdoba posee una sílaba más suave en el fonema /s/ que en Buenos Aires.

Regla 4: La 'c' antes de la 't' no se pronuncia

La sílaba /c/, que precede a /t/, no se debe pronunciar.

Ejemplo: 'Doctor' no debe sonar el fonema /c/.

Regla 5: La 'y' y 'll' se pasa a 'i'

Palabras con el fonema /y/ o /ll/ se pronuncian /i/.

Ejemplo: 'lluvia' se debe pronunciar utilizando el fonema /i/

Regla 6: La 'r' no debe sonar. No debe vibrar

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 habla espontánea. No surgen de habla leída.

Vamos a tomar distintas frases donde aparecen estas reglas y compararlas con la dialéctica de Buenos Aires.

2. DISEÑO DEL EXPERIMENTO

Utilizando estudios previos de ambos dialectos, pudimos 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 estos grupos. El experimento va a poner foco primordialmente en las diferencias antes dichas. La idea sera realizar una serie de actividades donde el hablante sea grabado y esas actividades hagan incapié en las diferencias. Como el acento se potencia cuando se realiza habla espontánea, vamos a querer que el hablante lo diga de forma lo más natural posible.

A continuación vamos a describir el experimento en más detalle.

2.0.1. Frases conocidas

Debemos obtener habla espontánea de los hablantes. Es por ello que se nos ocurrió como actividad pronunciar frases popularmente conocidas. Pensamos que al ser muy conocidas el hablante, al haberla pronunciado tanto, vamos a obtener 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. De esta forma se quiere obtener habla espontánea.

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 grupos Vamos a tener dos formas de grabaciones: frases comunes que tratan de cubrir la espontaneidad (cubriendo las reglas 2 al 5) y frases AMPER que tratan de cubrir el acento barriendo por cada tipo de acentuación (regla 1). A continuación vemos las reglas en sus dos conjuntos.

2.1.1. Frases utilizadas

2.1.2. Utilizar AMPER

Utilizamos este esquema para analizar todas las variantes posibles de la Regla 1. Recordemos que la regla 1 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 el esquema AMPER fijamos un patrón y fuimos cambiando las palabras que utiliza. El esquema AMPER utilizado es:

Objeto+” salió “+Adjetivo

			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

Fig. 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 (buSkes)			
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 chillaba				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)		

Fig. 2.2: Frases conocidas

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

Por ejemplo: *El canape salio delicioso*". Canape 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.

2.1.3. Utilizar frases conocidas

Como afirmamos antes, utilizo frases comunes para poder obtener los acentos de cada grupo de forma natural. Pensé que si se graba una frase popular, el hablante al estar acostumbrado a decirla no iba a poder evitar impregnarle el su acento. 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 *ç* antes de la *t*". Sucede igual con la palabra *çaballero*" para la regla 5. Esta nos dice medir la duración de la *ll*". De esta forma cada frase extrae los atributos lo mas posible.

Intercalando los dos tipos Ahora debemos intercalar las trazas de frases comunes con las de Amper. Vamos a agregar 4-5 frases de la traza de frases comunes, luego agregar 1-2 frases del esquema de amper y así sucesivamente. 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 minima cantidad de grabaciones que puede realizar un hablante son 5 grabaciones. Luego se le pregunta si quiere seguir grabando, si dice que si se le agregan otras 5 grabaciones así sucesivamente hasta llegar a las 40 que es el total de grabaciones.

2.2. Generación de trazas

Debemos definir qué frases y en qué orden se debe decir durante el experimento. Una traza es una lista de las frases que va a grabar un hablante. Sucede que el orden que utilizemos 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.

Optamos por tener precalculada las trazas para evitar cálculos innecesarios a la hora de empezar el experimento. Es por eso que guardamos 10000 trazas generadas. Un hablante al empezar se le dará una de estas y grabará sucesivamente en ese orden.

La generación de trazas sigue el siguiente algoritmo:

```

1  Genrador de trazas :
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

Vamos a ponderar las frases que referencien a más reglas. 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 tomándolo en cuenta. Elegiremos 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.

Lo bueno de esto es que para un hablante grabando 10 frases tenemos muestras de todas las reglas. Este es un dato importante ya que maximizamos la cantidad de información de cada frase y le hacemos perder menos tiempo realizando el experimento. Esto se puede ver en el siguiente gráfico.

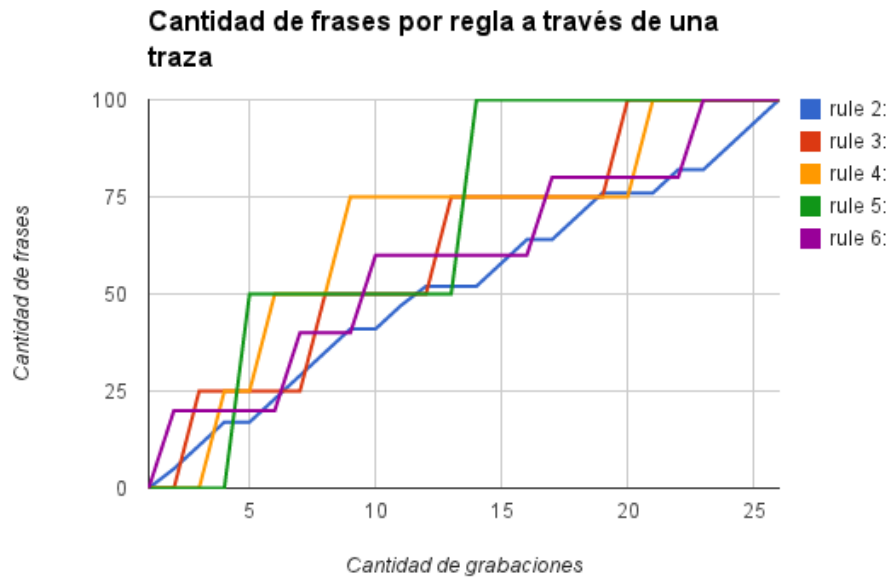


Fig. 2.3: Cantidad de frases por traza

En conclusión, se grabarán de a 5 frases. Teniendo en cuenta que las últimas 2 corresponderán a el esquema AMPER.

3. ARQUITECTURA DEL SISTEMA

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.

La página web esta desarrollada en Django que es un framework para la creación de paginas web. Esta misma necesita una base de datos para guardar cada clase de dominio. En esta base vamos a guardar los datos de cada hablante, las frases a grabar y las trazas principalmente.

3.1. 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. Luego se procede a realizar las grabaciones.

En la pantalla de grabación el usuario deberá confirmar tener acceso al micrófono que posee en su dispositivo. Una vez hecho esto, 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. Las grabaciones pueden ser escuchadas antes de ser confirmadas por el usuario. Lo importante es que la grabación se escuchen lo mejor posible.

Cada vez que se confirma una grabación, esta se graba en un archivo wav en el servidor.

3.2. 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 y 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 (<http://code.google.com/p/wami-recorder/>) 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 meterse en la implementación. Utilizando dicha interfaz uno configura 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.

Cabe aclarar que 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. 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.

3.3. Requerimientos

Los requerimientos son básicos: micrófono, conexión a internet. Tuvimos problemas sobre el browser que utilizaba. Wami necesita Flash versión 11.04 (chequear) 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 podía correr perfectamente la aplicación sin importar el sistema operativo.

3.4. Sistema de administración

El sistema debe tener datos cargados para estan online recolectando audios. Los minimos 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. Una vez cargado esos datos, los hablantes van a acceder a una url donde van a poder realizar el experimento grabando los distintos audios.

3.4.1. 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.

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 sincronizacion 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

Fig. 3.1: Categorizando audios

3.5. 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 optamos por 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.

3.6. 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 cortas entonces no es necesario 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.

4. EXTRACCIÓN DE DATOS

4.1. Alineación forzada

Gracias a 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. Y además, un dato muy relevante, 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.

Las partes que debemos extraer de los audios son las partes donde se encuentran la diferencias de cada regla descrita anteriormente. Debemos tener una herramienta que nos permita obtener esas pequeños pedazos de audios para analizar sus diferencias.

4.1.1. Prosodylab Aligner

Luego de buscar bastantes, encontramos con una librería llamada ProsodyLab Aligner (<http://prosodylab.org/tools/aligner/>). Su función es realizar alineaciones automáticas en cada uno de los audios de forma fácil. O sea, va analizar cada audio y mediante un diccionario determina cada fonema en que tiempos se dijo. El formato utilizado para devolver estas marcas es el TextGrid.

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 esta 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. Básicamente trata de predecir que fonemas aparecen en cada parte del los audios utilizando las diferentes muestras.

Vamos a analizar si cumplimos con los requerimientos. 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 - INIGEM que nos prestó su diccionario pudimos utilizar esta herramienta. El diccionario fonético nos provee para cada palabra los distintos fonemas que la componen. 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.

4.2. Extracción de atributos

En esta sección vamos a analizar como extraer cada uno de los atributos que planteamos en cada regla.

4.2.1. Implementación

La extracción de datos fue realizado en 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. 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.

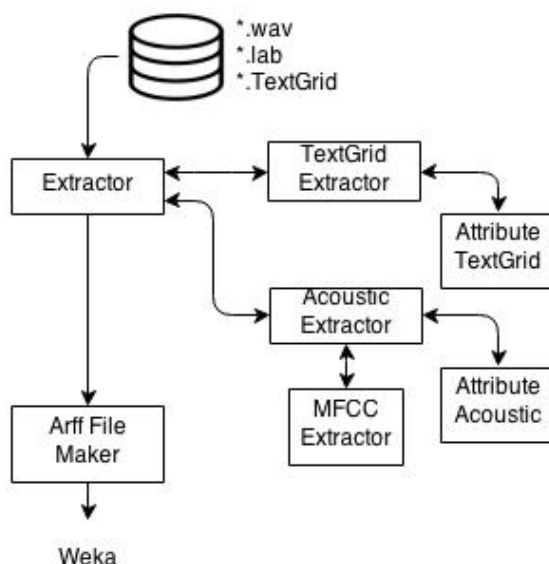


Fig. 4.1: Diagrama workflow

El ProsodyLab-Algner al finalizar una alineación nos devuelve un archivo donde se encuentra como fueron realizadas esas alineaciones. Este archivo se llama '.SCORES' y en el son lista 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.

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:

4.2.2. 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 normal utilizando:

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

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

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

Esta ultima tiene el nombre de half normal distribution.

Vamos a dividir el grupo de los atributos temporales 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':** con este atributo vamos a buscar el patrón 'kt' en las frases y luego a medir normalizando de las dos formas el fonema 'k'.
- **Duración de la 'sc':** ídem con 'sc' y midiendo el fonema 's'.
- **Duración de la 'll':** buscamos el patrón 'll' y lo medimos normalizandolo.
- **Duración de la 'rr':** ídem para 'rr'.
- **Duración de la 's' final:** Ídem para las 's' de final de palabra.
- **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 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.

4.2.3. 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 cambian constantemente poseen muchas variaciones. 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.

El 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 el logaritmo. 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 señal. -> 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 BLA 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.

5. DATOS OBTENIDOS

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

Escuché los audios para determinar si se realizaron correctamente. 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:

	Bs.As.	Cba.	Total
Conservar	222	105	327
Problemas en el habla	33	15	48
Mucho ruido de fondo	2	12	14
Sonido saturado	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 sería 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 e 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 Conser-vados. 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.

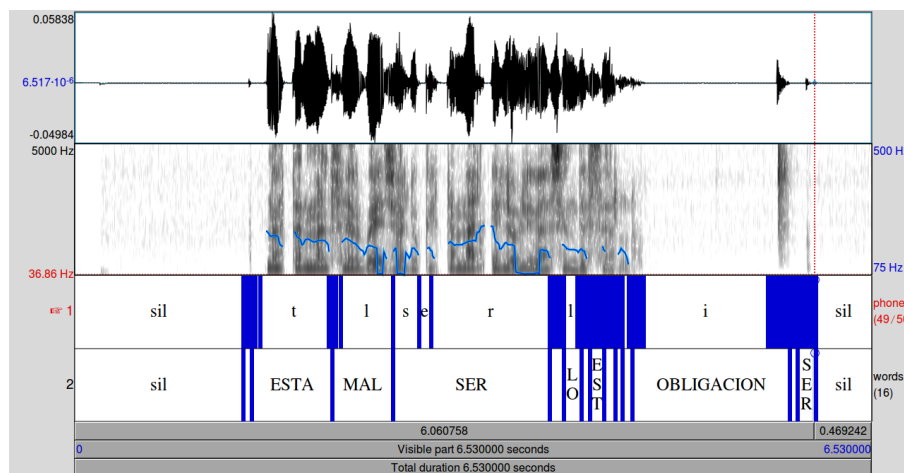


Fig. 5.1: Alineación mala

5.1.1. Errores comunes

Es muy importante descartar los audios mal alineados ya que sino cuando los procese el extractor nos darían información errónea. Fuimos chequeando cada audio y analizando si la alineación fue correcta. Para realizar esta tarea utilizamos el programa Praat. 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. Ese sonido se grabó y afectó la alineación de forma tal que se tomaba como habla.

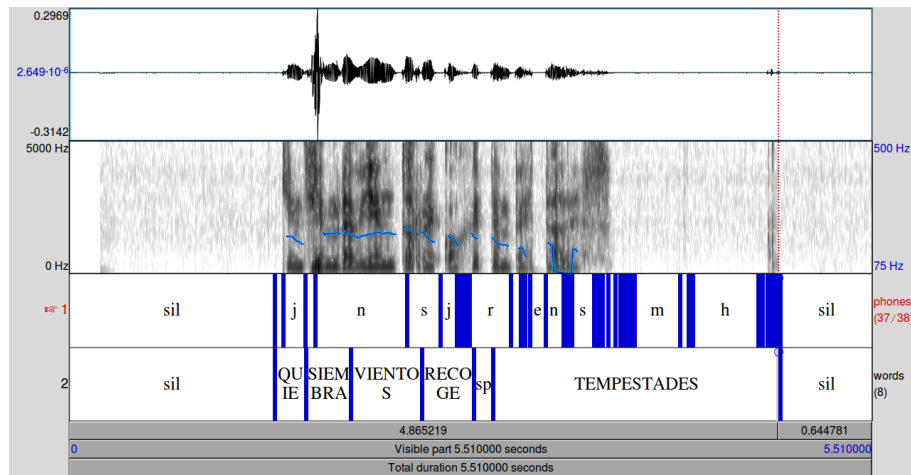


Fig. 5.2: Click al final

- **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 mucho 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:

	Bs.As.	Cba.	Total
Modificados	101	88	189
Correctos sin modificación	119	2	121

Esta forma se realizó ya que eran pocos audios. Otro modo de hacer esto pero de forma automática puede ser de la siguiente forma. Cuando se realiza la alineación, se crea un archivo llamado .SCORES. Este archivo posee los valores de verosimilitud de cada alineación. Entonces este archivo podría servir para definir si un audio esta bien alineado. Si la alineación superó un umbral pre-configurado, el audio estará bien alineado y se podrá utilizar para el estudio, sino el audio se descartará. Vale aclarar que esto puede tener falsos positivos. Esta idea podría realizarse en trabajos futuros.

5.2.1. Disminución del ruido con Sox

6. ANÁLISIS

Vamos a mostrar los resultados que obtuvimos luego de realizar la extracción.

6.1. Baseline

No hay ningún trabajo que trate de distinguir entre porteños y cordobéses 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. Al tener más hablantes de Buenos Aires que de Córdoba en nuestro set de datos acertaría más del 50 % de las veces. En promedio va a tender a (Completame XX) % de efectividad. 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.

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. Podría surgir el problema de que un hablante tenga audios en el conjunto de training y en el de testing. 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 los grupos de train y test eligiendo al azar hablantes. Cuando tomemos un hablante al azar, debemos agregar todos sus audios grabados. En el siguiente pseudocódigo se explica la metodología.

```
1  Generador de Test :
2  Input: conjunto audios
3  Output: < conjunto train , conjunto test >
4  train <- audios
5  test <- {}
6  mientras percentage < 0.1:
7      hablante_BsAs <- Elegir al azar(audios)
8      grabaciones_BsAs <- ObtenerGrabaciones(hablante_BsAs , audios)
```

```

9
10     hablante_Cba <- Elegir_al_azar(audios)
11     grabaciones_Cba <- ObtenerGrabaciones(hablante_Cba, audios)
12
13     tests <- tests U grabaciones_BsAs U grabaciones_Cba
14     train <- train - grabaciones_BsAs - grabaciones_Cba
15
16     percentage <- Tam(test) / Tam(train)
17
18     Devolver <train, test>

```

Este proceso lo debemos realizar 10 veces para poder evitar que hayamos tenido suerte al generar el test. Luego de generado estos 10 pares de grupos de train y test vamos a utilizarlos para entrenar clasificadores.

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) - 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 para. Este algoritmo sirve mucho para datos no balanceados.

C45 - Implementación J48: Este algoritmo genera un árbol de decisión y es muy usado para clasificación. Dada una serie de muestras con varios atributos cada una se realiza: Para cada atributo calcula la ganancia de información si se divide las muestras en a. 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, este es el caso base.

Sequential minimal optimization - Implementación Function SMO: Este algoritmo intenta resolver un problema de optimización para los problemas tipo Support vector machines. Support vector machines es un problema por el cual se intenta separar muestras en dos clasificaciones distintas a través de un hiperplano. Este hiperplano posee la máxima distancia a los datos de entrenamiento mas cercanos para cada clase. Sequential minimal optimization es una implementación que intenta resolver este problema. Lo que hace es: divide el problema en una serie de pequeños problemas que pueden ser resueltos facilmente. Cada pequeño problema es encontrar multiplicadores en una ecuación de 2 variables.

Naive Bayes - Implementación ídem: Un clasificador de este tipo asume que cada atributo de cada clase muestra una característica en particular de su clase pero no esta relacionado con otro atributo. Cada una de estos atributos contribuye de manera independiente a la clasificación de su clase.

Más adelante describiremos los parámetros utilizados para cada uno.

6.4. 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 10 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.

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 armamos los tests. La población también siempre es la misma.
- Cada par es elegido de forma azarosa y independiente del resto: cada grupo generado para testing esta armado de forma azarosa ya que la elección de cada hablante se realiza de esta forma.
- Los datos estan 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

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 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 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 $\geq 0,05$, por ejemplo ZeroR y J48, 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 significativa mente 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

H_1 : 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 .

6.5. Resultados

Todos tests fueron realizados utilizando R versión 3.0.1.

6.5.1. Clasificadores

Los resultados para los distintos clasificadores fueron:

	ZeroR	JRip	J48	Function SMO	NaiveBayes
Test 1	0	0	0	0	0
Test 2	0	0	0	0	0
Test 3	0	0	0	0	0
Test 4	0	0	0	0	0
Test 5	0	0	0	0	0
Test 6	0	0	0	0	0
Test 7	0	0	0	0	0
Test 8	0	0	0	0	0
Test 9	0	0	0	0	0
Test 10	0	0	0	0	0

donde Test 1 corresponde al primer par {conjunto train, conjunto test } y así sucesivamente.

6.5.2. Wilcoxon y Student Test

Vamos a mostrar los resultados de estos tests estadísticos.

	Student Test	Wilcoxon Test
ZeroR y JRip	0	0
ZeroR y J48	0	0
ZeroR y NaiveBayes	0	0
ZeroR y Function SMO	0	0

6.6. Selección de atributos de forma automática

Attribute Evaluator: InfoGain

Utilizando todos los atributos

0	attr1
0	attr2
0	attr3
0	attr4
0	attr5

Utilizando solo los atributos PHO

Utilizando solo los atributos SIL

Utilizando solo los atributos ACO

Attribute Evaluator: ClassifierSubsetEvaluator

Utilizando todos los atributos

Utilizando solo los atributos PHO

Utilizando solo los atributos SIL

Utilizando solo los atributos ACO

7. CONCLUSIONES

8. TRABAJOS FUTUROS

9. APÉNDICE: MARCAS PROSÓDICAS

@inproceedingsCohen1995, author = William W. Cohen, booktitle = Twelfth International Conference on Machine Learning, pages = 115-123, publisher = Morgan Kaufmann, title = Fast Effective Rule Induction, year = 1995

@bookQuinlan1993, address = San Mateo, CA, author = Ross Quinlan, publisher = Morgan Kaufmann Publishers, title = C4.5: Programs for Machine Learning, year = 1993