UNIVERSIDAD DE TALCA
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA CIVIL EN COMPUTACIÓN

API de Reconocimiento de Voz para Lenguaje Castellano usando Redes Neuronales

SERGIO FLORES LABRA

Profesor Guía: RODRIGO PAREDES

Memoria para optar al título de
Ingeniero Civil en Computación

Curicó – Chile
24 de Enero de 2018

CONSTANCIA

La Dirección del Sistema de Bibliotecas a través de su encargado Biblioteca Campus Curicó certifica que el autor del siguiente trabajo de titulación ha firmado su autorización para la reproducción en forma total o parcial e ilimitada del mismo.

Curicó, 2019

Dedicado a mis padres, ya que gracias a sus esfuerzos, he podido estudiar.

«No era más que un zorro semejante a cien mil otros. Pero yo le hice mi amigo y ahora es único en el mundo.» El principito.

AGRADECIMIENTOS

Agradezco a todos aquellos que han sido parte de este proceso, ya sean familiares, amigos, profesores o conocidos, que de una u otra forma han permitido que esto sea posible.

Ha sido un largo proceso Universitario, en donde he podido conocer a grandes personas a las cuales puedo llamar y considerar mis amigos. He vivido muchos momentos los cuales nunca olvidaré, como hay otros que simplemente quiero dejar atrás en el olvido. Aunque pensándolo bien, de todos esos momentos debo dar gracias a Dios, por que sin ellos no habría podido madurar y ser quién soy ahora.

Como no agradecer a aquella mujer que me ha acompañado fielmente desde 3^{ero} medio, y con quien me proyecto a pasar todo lo que dure mi vida. Gracias Cynthia, porque sin tu apoyo en las instancias más difíciles, quizás el resultado sería otro.

Nombrar a quién me soportó durante este proceso de Memoria de Título, ya que a veces puedo ser muy majadero y molestar hasta el cansancio para recibir una retroalimentación. Es por eso Rodrigo, que por escrito dejo mis más sinceros agradecimientos por disponer de tu tiempo, sé que era valioso y difícil dado el doloroso proceso que te tocó pasar. He aprendido a conocerte y valorarte por quién eres, ya que bien en el fondo tienes un gran corazón. Broma, no es tan en el fondo.

Daniel, Gonzalo y Matías, los amigos con los que pude compartir toda mi estadía en la U, ya que formamos una amistad desde primer año. Gracias por escuchar mis historias, incluso aunque fuesen fomes. Gracias por compartir sus tiempos conmigo y por poder contar con ustedes en todo momento.

Gracias a Hernando y Marcelita, porque más que funcionarios de la Universidad, son y serán parte de mi vida. Más que la relación laboral, pude encontrar grandes personas que siempre se preocupaban por mí. Un saludo, una comida o un simple buenos días son gestos que llevaré siempre en mi memoria.

RESUMEN

En los últimos años, el Reconocimiento de Voz ha jugado un papel importante dentro del mundo tecnológico. Esta tecnología permite, entre otras aplicaciones, la transcripción de audio a texto, sistemas de subtitulado en tiempo real, traducción de frases audibles y asistentes virtuales para dispositivos móviles, etc.

Existen muchos mecanismos para la solución de este tipo de problemas, tales como los clasificadores, los filtros de señales y las redes neuronales; de los cuales este último ha logrado llevarse los elogios por el porcentaje de exactitud de respuesta. En los últimos 17 años, con la incorporación de Deep Learning, se ha alcanzado exactitudes superiores a un 98 % según lo declarado por Google.

El modelo de Red Neuronal que ha permitido estos grandes avances es el modelo LSTM, el cual consiste en una red neuronal que posee una memoria a largo plazo y en base a su retroalimentación y un par de mejoras, permite entregar una respuesta más precisa al calcular la probabilidad de la palabra que se está detectando.

Las soluciones existentes requieren del acceso al centro de procesamiento del proveedor, afectando a los clientes que no cuentan con conexión a internet. Por lo anterior, en esta memoria se construye una API que incluye funcionalidades para los trabajos de reconocimiento de voz en el lenguaje castellano y que pueda trabajar desconectada de la internet. El ideal es poder contar con una herramienta que no sólo reconozca la frase, sino que también pueda corregir la sintaxis de la respuesta por medio de un diccionario de palabras.

Para evaluar si la API logra cumplir con las expectativas planteadas, se utilizan pruebas de exactitud del modelo y tiempo de respuesta. El corrector de palabras basado en diccionario responde en 0.5 segundos con una exactitud de un 93.8 %. Cosa distinta se da en el modelo de reconocimiento, ya que los resultados no pueden ser medidos de la misma forma por no contar con una base de datos de entrenamiento razonablemente grande. Esto se debe a que los modelos neurales para reconocimiento de voz basados en Deep Learning necesitan mucha información para su entrenamiento, la cual es muy difícil de conseguir, ya que estamos hablando del orden de los cientos de miles de datos. Por otra parte el hecho de trabajar con lenguaje Castellano dificulta aún más el problema, ya que los caracteres con tildes y diéresis causan dificultades en el procesamiento.

RESUMEN

In recent years, Voice Recognition has played an important role in the technological world. This technology allows, among other applications, audio-to-text transcription, real-time subtitling systems, translation of audible phrases and virtual assistants for mobile devices, etc. There are many mechanisms for solving this type of problem, such as classifiers, signal filters and neural networks, of which the latter has been praised for the percentage of response accuracy. In the last 17 years, with the incorporation of Deep Learning, accuracy has been achieved above 98 % as declared by Google. The Neuronal Network model that has allowed these breakthroughs is the LSTM model, which consists of a neural network that has a long term memory and based on its feedback and a couple of improvements, allows to provide a more precise response when calculating the probability of the word being detected. Existing solutions require access to the provider's processing center, affecting customers without an Internet connection. Therefore, an API is built in this memory that includes functionalities for speech recognition works in Spanish language and that can work disconnected from the Internet. The ideal is to have a tool that not only recognizes the phrase, but can also correct the syntax of the answer by means of a word dictionary. Tests of model accuracy and response time are used to assess whether the API is meeting expectations. The dictionary-based word corrector responds in 0.5 seconds with an accuracy of 93.8 %. This is different from the recognition model, as results cannot be measured in the same way because there is no reasonably large training database. This is because the neural models for speech recognition based on Deep Learning need a lot of information for their training, which is very difficult to achieve, since we are talking about the order of hundreds of thousands of data. On the other hand, the fact of working with Spanish language makes the problem even more difficult, since the characters with accents and dieresis cause difficulties in the processing.

TABLA DE CONTENIDOS

página

Anexos

ÍNDICE DE FIGURAS

página

ÍNDICE DE TABLAS

página

1. Introducción

La tecnología y el uso de Internet en los últimos años ha crecido de forma exorbitante. Según la Subsecretaría de Telecomunicaciones (SUBTEL), en los Resultados de la Encuesta Nacional de Acceso y Usos de Internet realizada el año 2015, se declara que el 70 % de los usuarios tienen acceso a Internet y que las razones por las cuales se hace uso de esta tecnología son: 92.3 % para tener acceso a información y en un 89.4 % para comunicarse con otras personas.

El rápido avance de las tecnologías de la comunicación se ve reflejado de manera mundial y esta alta demanda ha provocado que se busquen soluciones para simplificar las vidas de los usuarios. Aplicaciones como asistentes de voz, reconocimiento del habla, transcriptores de audio a texto y detectores texto en imágenes, entre otras, son sólo algunas de las propuestas utilizadas, creadas en pos del usuario con el objetivo de simplificar ciertas tareas puntuales de la cotidianeidad.

Existen ciertas actividades humanas complejas de llevar a cabo en sistemas computacionales. En respuesta a esto, nace la Inteligencia Artificial como una rama de la computación que busca enseñarle a la máquina o computador a realizar tareas que una persona puede hacer. Ejemplo de ello es enseñar a un computador a jugar ajedrez, identificar patentes de vehículos, reconocer a personas dentro de videos o fotografías y percibir los sentimientos de las personas en textos, entre otras.

En esta investigación se propone el uso de Inteligencia Artificial para reconocer un extracto de audio y transcribirlo a texto. Es necesario aclarar que no es un sistema nuevo. En efecto, existen varias empresas tales como Microsoft, Google o Apple, que incluyen esta funcionalidad en sus dispositivos móviles o de escritorio; pero no entregan una API para incluir estos beneficios al momento de crear una aplicación

externa o es necesario utilizar conexión a internet para utilizar sus servicios. Es por ello que se busca generar una solución basada en redes neuronales, para transcribir el audio a texto sin la necesidad de hacer uso de internet para dar una respuesta.

1.1. Descripción del Contexto

Esta investigación se lleva a cabo para la empresa *Exe Ingeniería*, ubicada en Santiago de Chile. Se busca generar un prototipo inicial de una API de reconocimiento de voz, para posteriormente integrar dicha API a un sistema que permita llenar formularios de forma automática.

1.2. Descripción del Problema

Exe Ingeniería ha detectado un problema en empresas dedicadas al rubro de la construcción, donde diariamente los usuarios necesitan solicitar herramientas y materiales. Actualmente, los usuarios entregan un documento escrito a mano difícil de comprender a simple vista, por lo que se propone como solución final el desarrollo de un software capaz de registrar los materiales requeridos de forma hablada, reduciendo los errores operacionales tales como, no entender la petición que se está realizando o reducir el tiempo que dedican en completar de la solicitud.

1.3. Objetivos

1.3.1. Objetivo General

- Desarrollar una API de reconocimiento de voz para el lenguaje Castellano.

1.3.2. Objetivos Específicos

- Generar una estructura escalable, que permita resolver el problema de reconocimiento de voz.
- Verificar de forma experimental el comportamiento del modelo.
- Desarrollar una mini aplicación de reconocimiento de voz que utilice la API producida en esta memoria.

1.4. Alcances del Proyecto

- La API se restringe a responder al conjunto de palabras con la cual fue entrenada.
- Soporta una cantidad limitada de palabras, debido al extenso tiempo necesario para su entrenamiento.
- Se restringen las entradas de voz a hombre o mujer mayores de edad, descartando a niños para reducir problemas de fonética.
- El tiempo de audio de entrada se restringe a un largo máximo de 10 segundos para reducir el tiempo de entrenamiento.
- Se considera la API un prototipo funcional escalable.

1.5. Estructura del Documento

A continuación se presenta la estructura del resto de este documento.

En el Capítulo 2, *Antecendentes*, se muestra una breve reseña teórica del problema, pasando por el origen de la voz, técnicas de reconocimiento de voz y algoritmos de búsqueda de palabras en espacios métricos.

En el Capítulo 3, *Análisis del Problema*, se describe los trabajos existentes que intentan solucionar el problema en cuestión y la propuesta de solución que se desarrolla durante este trabajo.

En el Capítulo 4, *Desarrollo*, se describe la arquitectura de la solución, la metodología utilizada, bajo qué condiciones se lleva a cabo la solución, y cómo se constituye internamente la solución.

En el Capítulo 5, *Pruebas*, se muestran las mediciones tomadas para validar que el trabajo realizado es correcto.

En el Capítulo 6, *Conclusiones y Trabajo Futuro*, se realiza una revisión global de los resultados con la finalidad de sintetizar el trabajo ejecutado. Por último, se entregan recomendaciones y propuestas de implementación para futuras mejoras en el sistema.

2. Antecedentes

2.1. Reconocimiento de Voz

El sonido humanamente audible consiste en ondas sonoras que producen oscilaciones de la presión del aire, que son convertidas en ondas mecánicas en el oído humano y percibidas por el cerebro. La propagación del sonido es similar en los fluidos, donde el sonido toma la forma de fluctuaciones de presión. En los cuerpos sólidos la propagación del sonido involucra variaciones del estado tensional del medio. En condiciones normales y a una temperatura ambiente de 15 grados Celcius, la velocidad de propagación del sonido en el aire es de 340 m/s lo que corresponde a 1224 km/h.

2.1.1. La Voz

La voz humana presenta numerosas características. Por ejemplo, el tono de voz masculino se mueve entre 50 y 200 Hz. En cambio, la voz femenina es un tanto más aguda, encontrándose entre 150 y 300 Hz [3]. Naturalmente, con análisis más detallado es posible determinar distintos patrones característicos, como lo son los indicados en el Cuadro 2.1.

Característica	Descripción
Amplitud	Es la fluctuación con que se mueve la voz.
Volume	Es la intensidad de la voz medida en decibeles, y se puede se categorizar en fuerte o despacio.
Tono	Es la frecuencia de la onda que genera la voz con la cual se mueve en el espacio. Es muy utilizado para rescatar las emociones con que se dicen las palabras.
Timbre	Es la característica que puede verse afectada ya sea por resfrío u otras enfermedades, y que genera que la voz se aprecie de forma más grave o aguda. Esta cualidad nos permite identificar dos fuentes sonoras diferentes.
Discurso	Contenido de la voz, es decir que es lo que se está diciendo.
Duración	Permite diferenciar entre un sonido corto y un sonido largo cuando el resto de sus cualidades o parámetros son idénticos.

Cuadro 2.1: Características de la Voz.

2.1.2. Proceso de Digitalización de la Voz

Digitalización es el proceso o acción de convertir información análoga a digital. En otras palabras, es convertir cualquier señal continua en una señal discreta numérica. Por lo general, la digitalización de señales es en binario (su codificación es en dos estados, 1 y 0), pero todo depende del tipo de señal o el propósito de la digitalización.

En el proceso de conversión de análogo a digital existen 3 subprocesos importantes: el muestreo, la codificación y la tasa/frecuecia de muestreo.

- Muestreo: Consiste en obtener medidas instantáneas, es decir, en lapsos de tiempo lo más breve posibles. A partir de estas muestras se puede recontruir la señal analógica original mediante interpolación. Cabe señalar que mientras más muestras se tiene, la representación de la señal es mejor.

- Codificación: Corresponde al valor que se le da a cada muestra. Es importante destacar que a mayor número de bits, el parecido a la señal análoga original es mayor.

- Tasa/Frecuencia de Muestreo: Es la cantidad de muestras por unidad de tiempo que se toman de una señal continua para producir una discreta. Este valor comúnmente suele expresarse en Hercios (Hz).

Hay que tener en consideración que la máxima audiofrecuencia perceptible por el oído humano es 20 kHz [25], por lo que con una frecuencia al doble bastaría para representar dicho audio de una manera apropiada. Sin embargo, el estándar introducido por el CD-Audio es de 44.1 kHz.

Para resolver el problema de cuántas muestras se debe seleccionar para replicar con exactitud una señal, Nyquist-Shannon propone un modelo matemático en 1924 [19] que permite volver a la señal original en su totalidad. La fórmula se basa en encontrar la Frecuencia más alta F_{max} contenida en la señal análoga $X_a(t)$. Con esto, la tasa de muestreo F_s debe ser mayor a $2F_{max}$, permitiendo recuperar totalmente la señal original usando la función de interpolación:

$$g(t) = \frac{\sin 2\pi F_{max} t}{2\pi F_{max} t} \tag{2.1}$$

Así, $X_a(t)$ se puede expresar como:

$$X_a(t) = \sum_{n=-\infty}^{\infty} X_a\left(\frac{n}{F_s}\right) g\left(t - \frac{n}{F_s}\right) \tag{2.2}$$

donde $X_a\left(\frac{n}{F_s}\right) = X_a(nT) \equiv X(n)$ son las muestras de $X_a(t)$.

2.1.3. Técnicas de Extracción de Caracteríticas

Existen varias técnicas de extracción de características, de las cuales MFCC es la más usada en el área del reconocimiento de voz, ya que integra variados mecanismos de extracción de características en una sola técnica.

- **MFCC**

  El algorítmo *Mel Frequency Cepstral Coefficients* [11] es una técnica que toma muestras de voz como entradas. Después del procesamiento, calcula coeficientes únicos para una muestra particular. MFCC toma la sensibilidad de la percepción humana con respecto a las frecuencias a considerar y por ello es lo mejor para el reconocimiento del habla.

Para obtener los coeficientes, la señal debe pasar por un proceso de varias etapas de filtrado [24] para llegar a un resultado, tal como se muestra en la Figura 2.1.

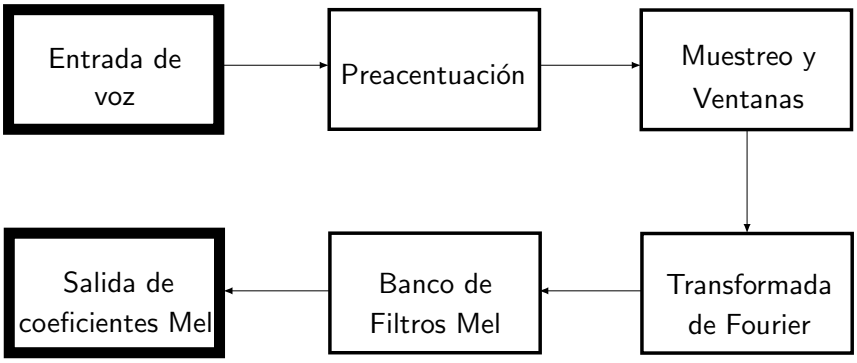

Figura 2.1: Diagrama en Bloques para la obtención de los MFCC.

Preacentuación: Esta etapa tiene como propósito amplificar las frecuencias altas. Esta etapa es usada por varios motivos, tales como balancear el espectro de frecuencias ya que las frecuencias altas suelen tienen menos magnitudes comparados con las frecuencias bajas, evita problemas numéricos al aplicar la transformada de Fourier y en algunas ocasiones mejora el rango de señal a ruido.

Muestreo y Ventanas: Esta etapa divide la señal en tiempos más pequeños. El sentido de esta división es porque la señal al transcurrir el tiempo posee cambios, que en la mayoría de los casos genera conflictos el aplicar la transformada de Fourier a la señal completa. Es por esto, que se asume de forma segura, que la señal es constante en un periodo corto de tiempo, de modo que al aplicar FFT en pequeños tramos logra generar una mejor aproximación de la curva de la señal. Esta división en tramas de audio, normalmente son hechas entre 20 ms a 40 ms con un ($\pm 10\,\%$) de solapamiento entre frames consecutivos. Junto con lo anterior, se aplica una función de Hamming para evitar discontinuidades al principio y al final de cada bloque. Para llevar esto a cabo se utiliza la Ecuación 2.3

$$w(n) = 0{,}54 - 0{,}46\cos\left(\frac{2\pi n}{N-1}\right),\ \text{con } 0 \leq n \leq N-1 \text{ y } N = |signal| \quad (2.3)$$

Transformada de Fourier: También conocida como *FFT*, es una función matemática que permite obtener el espectro de la frecuencia de una señal. La transformada de Fourier entrega una representación de la amplitud de la frecuencia usando la transformada y la inversa de la misma.

Banco de Filtros Mel: Es un banco de 40 filtros de onda triangular, que intenta imitar la percepción no lineal del oído humano, eliminando de forma discriminatoria las frecuencias más bajas. Para transformar la señal de Hertz a Mel y viceversa se utiliza la Fórmula 2.4. Una representación de la escala de Mel es como se muestra en la Figura 2.2 y puede ser obtenida usando la Ecuación 2.5.

$$\begin{aligned} m &= 2595 \log_{10}\left(1 + \frac{f}{700}\right) \\ f &= 700(10^{m/2595} - 1) \end{aligned} \tag{2.4}$$

$$H_m(k) = \begin{cases} 0 & k < f(m-1) \\ \dfrac{k - f(m-1)}{f(m) - f(m-1)} & f(m-1) \leq k < f(m) \\ 1 & k = f(m) \\ \dfrac{f(m+1) - k}{f(m+1) - f(m)} & f(m) < k \leq f(m+1) \\ 0 & k > f(m-1) \end{cases} \tag{2.5}$$

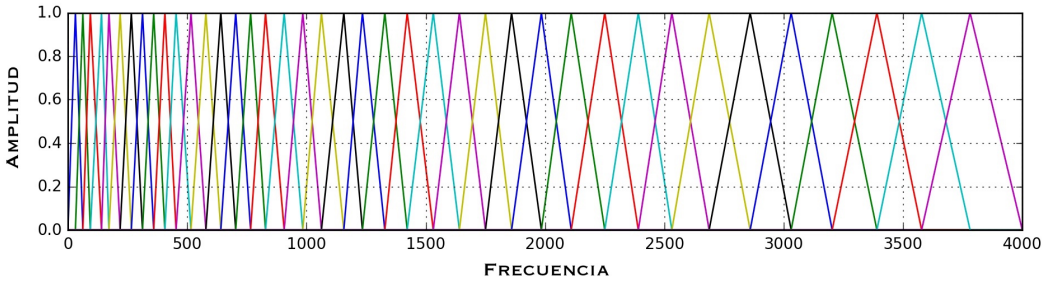

Figura 2.2: Banco de Filtros en Escala Mel.

Para aplicar dichos métodos de extracción de características, existen herramientas que proveen este y otros algoritmos, sin necesidad de implementarlos, de los cuales se destacan:

- SciPy: Librería de Python [14] que trabaja bajo la modalidad de Open Source, que entrega variadas funcionalidades para el desarrollo de sistemas basados en matemáticas, ciencias e ingeniería.
- Python speech features: Librería de Python que posee 5 funciones de extracción de características, a saber: Mel Frequency Cepstral Coefficients, Log Filterbank Energies, Spectral Subband Centroids, Fast Fourier Transform, Filterbank Energies.
- Open Smile: Es un conjunto de herramientas de código abierto para análisis de audio, procesamiento y clasificación especialmente dirigidos a aplicaciones de voz o música, ASR, reconocimiento de emociones, seguimiento de ritmos y detección de acordes. Este kit de herramientas se desarrolló en el Institute for Human-Machine Communication en la Technische Universitaet Muenchen de Munich, Alemania [4]. Este Kit puede ser utilizado con lenguajes como Java, C, C++, Python o Android.

2.2. Redes Neuronales Artificiales

2.2.1. Origen

La Inteligencia Artificial es la disciplina que estudia la forma de diseñar procesos que exhiban características que comúnmente se asocian con el comportamiento

humano inteligente [6]. La Inteligencia Artificial sintetiza y automatiza tareas intelectuales y es, por lo tanto, potencialmente relevante para cualquier ámbito de la actividad intelectual humana. Las redes neuronales artificiales son fundamentales en el contexto de la Inteligencia Artificial.

La red neuronal es un modelo matemático que está inspirado en la forma en que trabajan e interactúan las neuronas cerebrales de los seres humanos. El estudio de las redes de inteligencia artificial fué iniciado por biólogos, pero se ha vuelto un campo interdisciplinario, incluyendo a ciencias en computación, ingeniería eléctrica, matemáticas, física y psicología, entre otras.

El principal objetivo de la red neuronal es la construcción de sistemas capaces de presentar un cierto comportamiento inteligente. Esto implica la capacidad de aprender a realizar una determinada tarea. La base de una red neuronal es la neurona, la cual es una célula viva que posee características propias que permiten comunicarse entre ellas (sinápsis). La Figura 2.3 muestra la estructura típica de la neurona humana.

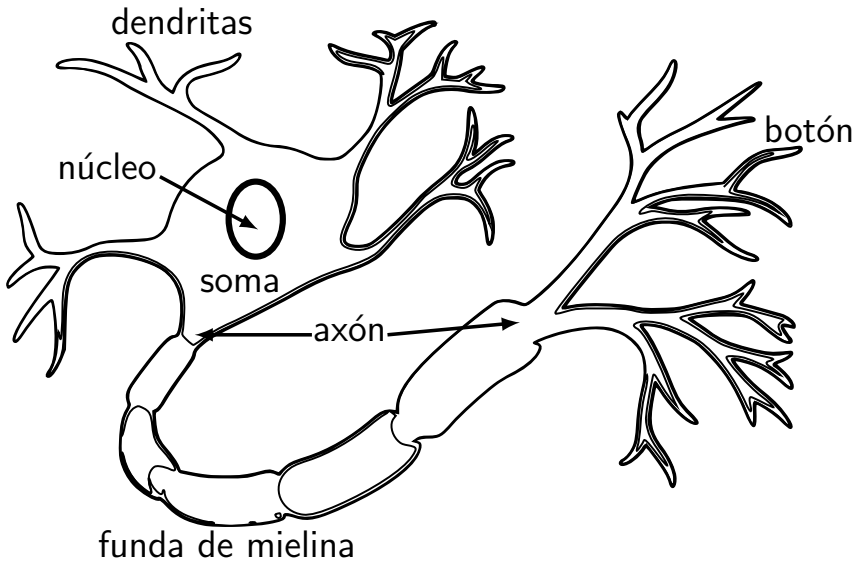

Figura 2.3: Neurona.

Para modelar la neurona humana, Frank Rosenblatt creó un modelo algorítmico llamado *perceptrón* en el año 1960 [15], que está compuesto por entradas, vías, pesos, función de entrada y función de activación. La Figura 2.4 muestra una representación

del modelo creado por Rosenblatt, donde se puede observar los conceptos nombrados anteriormente.

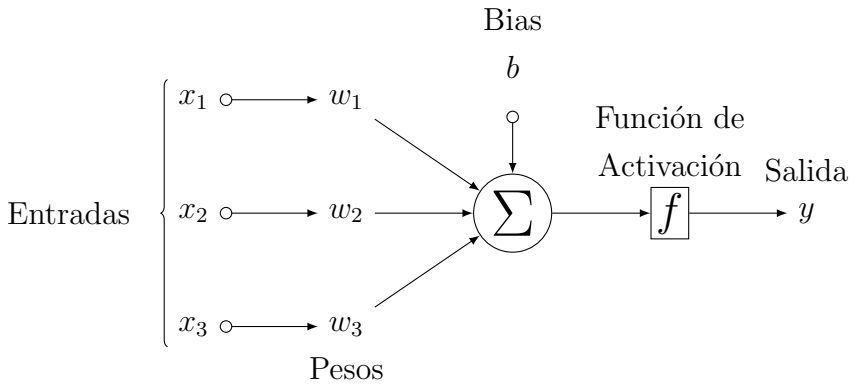

Figura 2.4: Representación del Perceptrón de Rosenblatt.

Como bien sabemos, los seres humanos poseen aproximadamente 100.000 millones de neuronas [23] que trabajan en conjunto y este mismo concepto fue tomado para construir un modelo de red neuronal. Dicho modelo recibe el nombre de perceptrón multicapa, ya que se divide en 3 grupos: las entradas, capas ocultas y salidas, tal como se muestra en la Figura 2.5, en donde cada capa posee neuronas representadas por un círculo, que para este caso es un perceptrón.

Cabe señalar que cada uno de estos componentes juegan un rol sumamente importante al momento de diseñar un modelo que represente una problemática en particular. Cada arista que conecta las neuronas con otra posee un peso, el cual representa el grado de comunicación que existe entre estas dos neuronas.

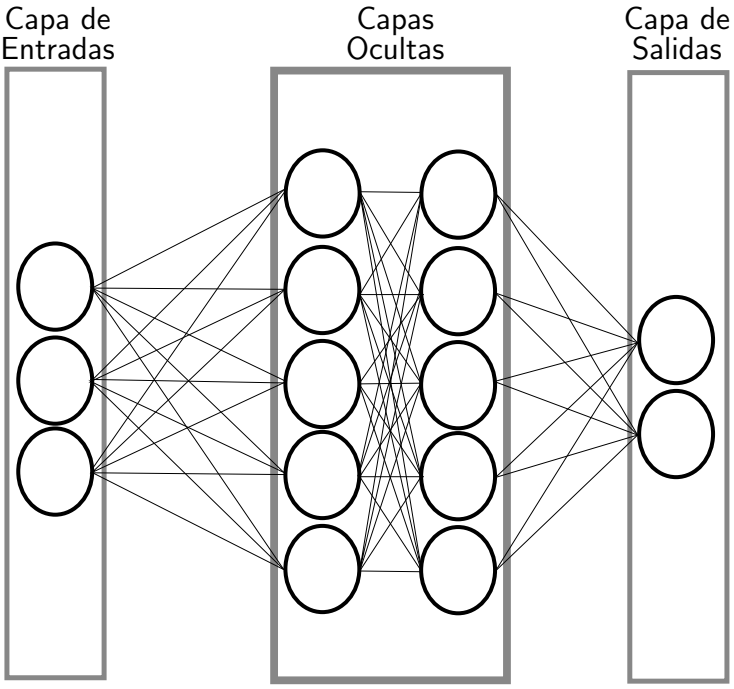

Figura 2.5: Modelo Estructural de una Red Neuronal Multicapas.

Una red neuronal posee una función de activación, la cual es una especie de excitador, que despierta a una neurona para que pueda trabajar. Hay varias funciones que se pueden utilizar con este objetivo, de las cuales se destacan:

Escalón: $$y = \begin{cases} 1 & \text{si } x \geq 0 \\ 0 & \text{si } x < 0 \end{cases}$$

Lineal a tramos: $$y = \begin{cases} x & \text{si } -1 \leq x \leq 1 \\ 1 & \text{si } x > 1 \\ -1 & \text{si} x < -1 \end{cases}$$

Sinusoidal: $$y = \sin(\omega x + \varphi) \quad \text{si } -1 \leq x \leq 1$$

La Función Sigmoide, por otra parte, cuenta con dos posibles formas de uso, todo depende del rango de x. Si x presenta un rango $0 \leq x \leq 1$, comúnmente se utiliza

la función matemática $\frac{1}{1+e^{-x}}$. Para un rango entre $-1 \leq x \leq 1$, se suele utilizar la función $\tanh(x)$.

El aprendizaje o entrenamiento de una red neuronal se basa en el ajuste de los pesos, encontrando los valores que dado cualquier entrada pueda entregar una salida válida y correcta. Donald Hebb en 1949 [9] postuló lo que se conoce como el inicio del aprendizaje de las redes neuronales. Hebb proveyó un algoritmo que permite actualizar los pesos en las conexiones de la red neuronal, concibiendo la base de los mecanismos de aprendizajes que se conocen hoy en día.

Existen 3 formas de aprendizaje para las redes neuronales, y son:

- **Supervisado:** Se presenta a la red un conjunto de patrones de entrada junto con la salida esperada. Los pesos se van modificando de manera proporcional al error que se produce entre la salida real de la red y la salida esperada.
- **No supervisado:** Se presenta a la red un conjunto de patrones de entrada. No hay información disponible sobre la salida esperada. El proceso de entrenamiento en este caso debe ajustar sus pesos en base a la correlación existente entre los datos de entrada.
- **Por refuerzo:** Este tipo de aprendizaje une ambos tipos de aprendizajes nombrados anteriormente. Se le presenta a la red un conjunto de patrones de entrada y se le indica a la red si la salida obtenida es o no correcta. Sin embargo, no se le proporciona el valor de la salida esperada. Este tipo de aprendizaje es muy útil en aquellos casos en que se desconoce cual es la salida exacta que debe proporcionar la red.

2.2.2. Tipos de Redes Neuronales

Las principales aplicaciones de las Redes Neuronales son:

- Clasificación de Patrones.
- Agrupamiento o Categorización.
- Aproximación de funciones.
- Memoria direccionable por contenido.

Para dar solución a otros problemas de mayor complejidad, se han creado distintos tipos de redes neuronales, basados en su estructura y diseño particular. Entre los más destacados, se puede apreciar la siguiente comparativa mostrada en el Cuadro 2.2.

Red Neuronal	Características	Ventajas	Desventajas
Adaline / Madaline	Red neuronal de una sola capa. Esta red es adaptativa, ya que permite configurarse para dada una entrada tener muchas salidas. Comúnmente es utilizada para sistemas de comunicación o control.	Facilidad de implementación y uso.	Es muy simple para problemas relacionados al reconocimiento de voz.
Perceptrón de multicapa	Es un tipo de red que presenta retroalimentación, el cual genera un mapeo del conjunto de entradas con la(s) distintas salidas. Esta función es utilizada para la aproximación de funciones, clasificación, pronóstico y control.	Es un modelo padre para muchos otros tipos de redes neuronales. Fácil de implementar y usar.	Es un modelo muy genérico que no logra dar solución a problemas específicos. Para problemas específicos es más apropiado usar variaciones de este.
Deep Learning	Basado en un conjunto de algoritmos que permite abstracción de alto nivel para problemas complejos. Es uno de los modelos que se está utilizando bastante por variadas empresas, Google o Apple, para resolver problemas del área de reconocimiento de imagen o voz.	Permite el modelamiento de problemas complejos utilizando estructuras abstractas como matrices, vectores, etc.	Requiere de mucho tiempo de entrenamiento para alcanzar buenos resultados.
Recurrent Neural Network	Es un tipo de red inteligente que posee ciclos entre sus unidades internas, generando un estado de comportamiento temporal dinámico. Este tipo de red neuronal ha sido usado en problemas de reconocimiento de escritura a mano o de reconocimiento de voz.	Ya que posee ciclos en su interior, esto genera mejoras al intentar dar soluciones a problemas que requieran de un contexto en particular.	Es un modelo complejo de implementación, pero que puede ser solucionado por herramientas que facilitan su implementación.
Convolution Neural Network	Es un tipo de red neuronal feed-forward, que trabaja de forma muy similar al comportamiento de las neuronas de la corteza visual del cerebro humano. Es una variación del perceptrón multicapa, que hace mucho más efectiva la tarea de tratamiento de imágenes por computadora.	Es una red que está diseñada para trabajos de clasificación tanto de datos como imágenes.	La desventaja de esta red es su tiempo extenso de entrenamiento, ya que requiere mucho tiempo para ajustar sus pesos para clasificar los datos.

Cuadro 2.2: Comparativa de los tipos de redes neuronales.

2.2.3. Redes Neuronales para Reconocimiento de Voz

Hay que tener en consideración que una red neuronal debe ser alimentada por entradas, que contextualizado a la problemática, estas entradas deberían ser características de la voz extraídas de un audio.

El problema de Reconocimiento de la Voz no es un problema nuevo. Existen varias soluciones y su respuesta y porcentaje de error depende exclusivamente del diseño de la red y del entrenamiento.

Como se habló anteriormente, existen distintos tipos de redes neuronales, de las cuales la más utilizadas en el área de reconocimiento de voz son Deep Learning, Convolution Neural Network y Recurrent Neural Network. Cada uno de estos tipos entregan enormes beneficios, pero el mayor problema de estos es su entrenamiento.

Tanto Deep Learning como Convolution Neural Network requieren un gran tiempo de entrenamiento. En efecto, han sido utilizados por *Microsoft* y *Google*, y declaran haber utilizado entre 3 a 8 meses de entrenamiento [10]. Recurrent Neural Network (RNN) es una buena opción para trabajar con problemas supervisados. El mayor inconveniente es que para reconocimiento de voz es necesario añadir un contexto a lo que se está diciendo. Esto se vé solucionado con LSTM (Long Short-Term Memory), que es una variación de RNN permitiendo agregar una memoria de corto a largo plazo.

Long Short-Term Memory, es una variación de una RNN publicada por Hochreiter y Schmidhuber en 1997 [12]. Es un tipo de neurona que tiene un estado interno que evoluciona con el tiempo, permitiéndole aprender relaciones entre datos presentados a la neurona en el pasado y los del presente. Así es que su respuesta está condicionada al contexto de su pasado, y no sólo a las entradas presentes.

Una Red LSTM posee, como componente básico, celdas de memoria para dar el contexto a la respuesta. Dicha celda es una unidad lineal auto-recurrente que está rodeada de una serie de compuertas. Esta celda puede guardar información por largos periodos de tiempo. El modelo matemático que representa a una celda está dado por la Fórmula 2.6. Dicho modelo matemático, puede ser representado de forma gráfica como se muestra en la Figura 2.6.

$$
\begin{aligned}
\mathbf{g}^u &= \sigma(\mathbf{W}^u \cdot \mathbf{h}_{t-1} + \mathbf{I}^u \cdot \mathbf{x}_t) \\
\mathbf{g}^f &= \sigma(\mathbf{W}^f \cdot \mathbf{h}_{t-1} + \mathbf{I}^f \cdot \mathbf{x}_t) \\
\mathbf{g}^o &= \sigma(\mathbf{W}^o \cdot \mathbf{h}_{t-1} + \mathbf{I}^o \cdot \mathbf{x}_t) \\
\mathbf{g}^c &= \tanh(\mathbf{W}^c \cdot \mathbf{h}_{t-1} + \mathbf{I}^c \cdot \mathbf{x}_t) \\
\mathbf{m}_t &= \mathbf{g}^f \odot + \mathbf{g}^u \odot \mathbf{g}^c \\
\mathbf{h}_t &= \tanh(\mathbf{g}^o \odot \mathbf{m}_{t-1})
\end{aligned}
\tag{2.6}
$$

Donde σ es la función sigmoide logística, $\mathbf{W}^u, \mathbf{W}^f, \mathbf{W}^o, \mathbf{W}^c$ son matrices de peso recurrentes y $\mathbf{I}^u, \mathbf{I}^f, \mathbf{I}^o, \mathbf{I}^c$ son matrices de proyección.

El Producto de Hadarmard [26], para dos matrices de igual tamaño es definido como $\odot$. Este producto declara que si $A(ij)$ y $B(ij)$ son matrices de igual tamaño, entonces $A(ij) \odot B(ij)$ corresponde a $C(ij)$, donde $C_{ij} = A_{ij} \cdot B_{ij}$

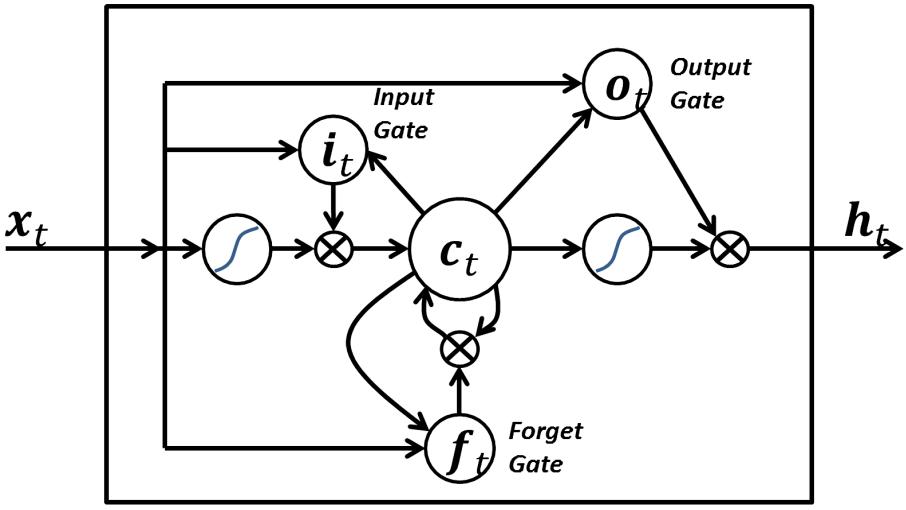

Figura 2.6: Celda de Memoria LSTM.

La capa oculta de una red LSTM está compuesta de muchas celdas de memoria y tanto las capas de entradas como de salida son convencionales. Una red LSTM se representa como se muestra en la Figura 2.7, en donde se muestra una entrada, una capa oculta de dos celdas y una salida. Cabe señalar que es sólo una representación, en efecto las redes LSTM pueden contener muchas entradas, salidas y celdas, dependiendo del problema en particular. El entrenamiento de una red LSTM es una mezcla

entre un aprendizaje recurrente en el tiempo y un aprendizaje retro-propagado en el tiempo.

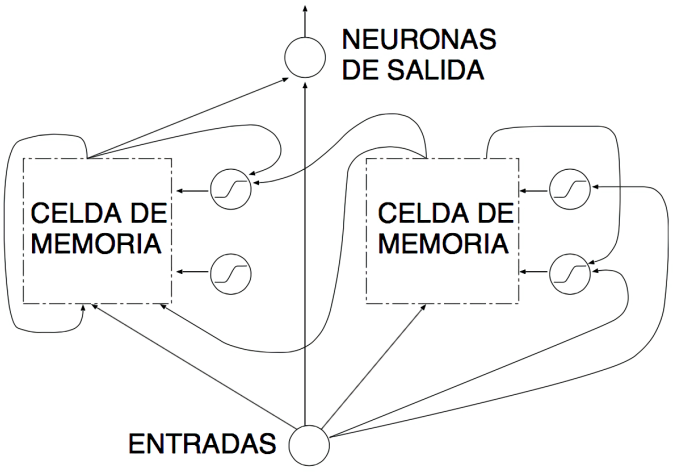

Figura 2.7: Red LSTM.

2.3. Búsqueda en Espacios Métricos

Según Papadopoulos [20], un espacio métrico es un conjunto no vacío M junto a una función de distancia con valor real no negativa $d(x, y)$ definida para cualquier par de objetos $x, y \in M$. Dicha función de distancia debe cumplir ciertas reglas, las cuales se enuncian a continuación:

- Positividad: $d(x, y) \geq 0$
- Simetría: $d(x, y) = d(y, x)$
- Reflexividad: $d(x, y) = 0$, si $x = y$
- Desigualdad triangular: $d(x, y) + d(y, z) \geq d(x, z)$

Al analizar esto, se puede percibir que dado dos objetos se obtiene la distancia que existen entre ellos como un valor mayor o igual a 0. El problema de trabajar con Espacios Métricos es que las búsquedas suelen tardar tiempo en generar un respuesta, ya que para obtener un objeto es necesario recorrerlos todos pues no existe orden total. Este problema de búsqueda no es nuevo, por lo que existen variadas soluciones que se basan en el uso de índices.

Los índices corresponden a una estructura de datos que permite realizar búsquedas más rápidas. Un ejemplo de índice es la libreta de direcciones telefónicas, en que cada número de teléfono está organizado y separado por abecedario con una viñeta correspondiente a la letra con que clasifica su nombre. Este tipo de solución suele utilizarse en muchos ámbitos, como los sistemas de Bases de Datos por ejemplo, pero siempre y cuando pertenezcan a un orden total, cosa tal que no se presenta en los espacios métricos.

Búsqueda por similitud o proximidad es un tipo de consulta que se realiza cuando el objeto en sí a consultar, no se encuentra exactamente en la base de datos. El objetivo de un índice para estas consultas es disminuir la cantidad de comparaciones de distancia o similitud, generando una respuesta en un menor tiempo.

2.3.1. Levenshtein

Levenshtein [16] es un algoritmo creado por Vladimir Levenshtein en 1966, que permite encontrar la distancia entre dos palabras, considerando inserciones, eliminaciones y cambios de caracteres para convertir una palabra en otra. Por medio de este algoritmo es posible detectar si dos palabras son iguales o qué tan diferentes son. Este algoritmo puede ser utilizado como función de distancia en problemas de espacios métricos basados en palabras. Una implementación de este algoritmo en formato de pseudocódigo es el que se muestra en el Algoritmo 1.

Algorithm 1 Levenshtein con Programación dinámica

```
1: int Levenshtein(palabra1, palabra2):
2:     if palabra1 == palabra2 return 0
3:     (s, t) = (|palabra1|, |palabra2|)
4:     m←[s+1][t+1]
5:     for i ← 0 to |s+1|
6:         m[i][0]←i
7:     for j ← 0 to |t+1|
8:         m[0][j]←j
9:     for i ← 0 to |s|
10:        for j ← 0 to |t+1|
11:            cost ← palabra1[i] == palabra2[j] ? 0:1
12:            m[i+1][j+1] ← min(m[i][j+1]+1, m[i+1][j]+1, m[i][j]+cost)
13:    return m[s][t]
```

2.3.2. Índices Basados en Pivotes

Para resolver el problema de búsqueda en espacios métricos, existe una amplia gama de algoritmos para indexar espacios métricos, de los cuales nos concentramos en los llamados *índices basados en pivotes*. Estos pivotes se construyen seleccionando k elementos $p_1, p_2, \cdots, p_k$ de M e identificando cada elemento $u \in M$ como un punto k-dimensional $((d(u, p_1), \cdots, d(u, p_k))$. Con esta información, es posible filtrar los elementos, u tales que $|d(q, p_i) - d(u, p_i)| > r$ para algún pivote p_i, dado que, en virtud de la desigualdad triangular, sabemos que $d(q, u) > r$ sin necesidad de evaluar realmente $d(q, u)$. Es interesante destacar, que los algoritmos basados en pivotes pueden reducir el número final de evaluaciones de distancia, aumentando el número de pivotes. Dado que el objeto se compara contra todos los pivotes, se puede estimar una cota inferior de la distancia entre el objeto u y la query q usando $D_k(q, u) = max_{1 \leq j \leq k} |d(q, p_j) - d(u, p_j)|$. Hacer uso de los pivotes, es equivalente a descartar los elementos u tales que $D_k(q, u) > r$.

2.3.3. Búsqueda de los Cercanos

La Búsqueda de los Cercanos [5] es un algoritmo de búsqueda por similitud en espacios métricos que entrega los objetos que son iguales o similares al buscado. Este tipo de algoritmo es uno de los más comunes en esta área, y sufre sus propias modificaciones dependiendo del ámbito a utilizarse. Cabe señalar, que antes de ejecutar este algoritmo, es necesario tener precargado la base de datos de los objetos, y tener precalculado el índice de búsquedas. En este caso corresponde al conjunto de pivotes y la distancia entre los objetos y pivotes, ya que el mayor costo se paga al calcular esta información, pero las consultas por cercanos son mucho más rápidas que al consultarlas de forma iterativa.

Para concretar este tipo búsqueda, típicamente se utiliza el Algorítmo 2, el cuál entrega el valor que se encuentra más cercano al de consulta. Si bien es importante destacar que esta variación no sólo responde el más cercano, si no que en caso de haber más de un objeto a la misma distancia mínima, este algoritmo entrega todos los que empaten. Esto es posible de llevar a cabo ya que las distancias a evaluar son discretas, puesto que las distancias de palabras se miden basadas en el número de operaciones de colición de caracteres (inserciones, eliminaciones o cambios). Cosa totalmente distinta sería si dichas distancias fuesen continuas.

Algorithm 2 Búsqueda de los Cercanos

```
1: db ← DB // base de datos de objetos
2: pivotes ← SSS(alfa, db) // calculamos los pivotes
3: distancias ← Distance(db, pivotes) // calculamos la distancia entre la db y los
   pivotes
4: Array Cercanos(target):
5:     qdp, ans ← [ ]
6:     minR ← ∞
7:     for i ← 0 to |pivotes|
8:         objeto ←db[i]
9:         val ←Levenshtein(target,objeto)
10:        if val < minR
11:            minR ← val
12:            and ← ans ⋃ objeto
13:        qdp← qdp ⋃ (pivotes[i], val)
14:    c ← [ ], ci ← [ |pivotes| ]
15:    for i ← 0 to |db|
16:        for j ← 0 to |pivotes|
17:            ci[j] ← abs(qdp[j][1]-distancias[i][j])
18:        val ← max(ci) // máximo valor en ci
19:        if (minR ≥ val) c ← c ⋃ (db[i], val)
20:    sortAscByVal(c) // ordenamos ascendentemente c por el valor
21:    for k ← 0 to |c|
22:        val = c[k][1]
23:        if(val > minR) break
24:        do ← Levenshtein(c[k][0],target)
25:        if (do = minR) ans ← ans ⋃ c[k][0]
26:        else if(do < minR) ans ← [ objeto ] minR, ← do
27:    return ans
```

2.3.4. Selección Espacial de Pivotes Dispersos para la Búsqueda por Similitud en Espacios Métricos

Sparse Spatial Selection [21] (SSS) es una heurística que pretende realizar una buena selección de los pivotes para el índice de búsqueda por similitud en espacios métricos, de forma tal que los pivotes seleccionados se encuentren lo suficientemente alejados entre sí con la finalidad de tener una cantidad de pivotes que logre ser representativa con respecto al conjunto de objetos en el espacio.

En simples palabras, SSS restringue la selección de pivotes, dando un rango de distancia mínima, permitiendo que los pivotes seleccionados abarquen una mayor superficie de muestreo. En el Algoritmo 3 se muestra el pseudocódigo de SSS.

Algorithm 3 Sparse Spatial Selection (SSS)

```
1:  Array SSS(alfa, DB):
2:      dMax ← CalculateDMax(DB) // valor del objeto mas largo
3:      pivotes ← [ ] , pivotes ← DB[0]
4:      for i ← 0 to |DB|
5:          isPivote = True
6:          for j ← 0 to |pivotes|
7:              if (levenshtein(db[i],pivotes[j]) < alfa*dMax)
8:                  isPivote ← False, break
9:          if (isPivote) pivotes ← pivotes ⋃ db[i]
10:     return pivotes
```

3. Análisis del Problema

3.1. Trabajos Existentes

Dentro de esta área es posible encontrar varios trabajos realizados, siendo los más comunes y conocidos los presentes en Sistemas Operativos Móviles o de Escritorio. No obstante hay más y de los cuales se nombran a continuación:

- **Reconocimiento de voz en Microsoft Windows:** Microsoft, en su sistema operativo Windows versión 7, incorporó un sistema de red neuronal capaz de entrenarse y reconocer al usuario para ejecutar distintos comandos. Posteriormente, dicho sistema fue mejorándose y para su versión 10, recibe el nombre de *Cortana* [17].

- **Cloud Speech API:** API de reconocimiento de voz creada por Google, que permite a los desarrolladores transformar audio a texto. Esta API permite el reconocimiento de voz en más de 80 lenguajes distintos. Este servicio es gratuito para audios de entre 0 a 60 minutos, y para audios más largos cobran $0.006 USD mensual con capacidad límite de 1 millón de minutos [7].

- **Siri Personal Assistent:** Siri [2] es un asistente personal creado por *Apple* para sus dispositivos móviles, tablets y, desde el año 2016, a sus computadores personales y su AppleTV. Este servicio presenta soporte en varios lenguajes.

- **VeriSpeak SDK:** Es un Kit de Desarrollo de Software (SDK) para identificar al hablante. Este sistema está diseñado para el desarrollo de sistemas biométricos. Tiene soporte para sistemas stand-alone y para sistemas web, sin dejar de lado a Móviles iOS y Android [18].

Cada sistema existente ha definido una estructura de trabajo basado en sus beneficios y/o utilidad, ya sea cobrando por el uso de estos servicios o limitándose a trabajar sólo con sus propios dispositivos.

3.2. Propuesta de Solución

Dado los sistemas que existen, ninguno cuenta con una solución que permita anexar este servicio a nuestros propios sistemas y que a su vez no requieran conexión a internet. Es por ello que se pretende crear un API de Reconocimiento de voz que trabaje a modo offline, que reciba archivos de audio y responda el texto correspondiente a dicho audio.

Las etapas a cumplir para obtener dicho resultados son:

- Wavelet: Etapa para convertir señales de audio análogas en digitales.
- Extracción de características: Etapa para extraer los datos relevantes del audio, y los prepara para ser posteriormente ingresados a la red neuronal.
- Clasificador: Modelo de red neuronal que convierte información de un audio a texto, el cual puede ser entrenado.

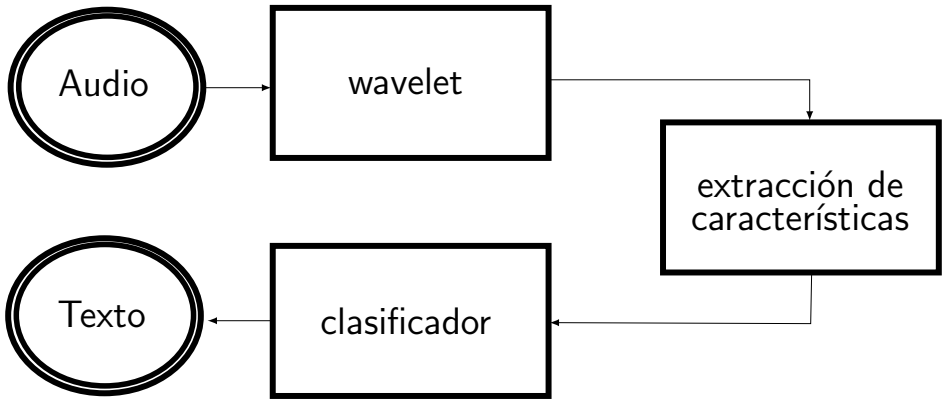

Figura 3.1: Etapas de Reconocimiento de Voz.

Existen otras etapas, como corrector de sintaxis, de semántica, eliminación de ruido, que buscan mejorar el porcentaje de exactitud en la respuesta. Sin embargo, para no extender demasiado el problema, no son contemplados.

4. Desarrollo

4.1. Arquitectura de Software

La Arquitectura de Software es una de las etapas más importantes del Desarrollo de Software, ya que de ella se obtiene un modelo base para el sistema. Este modelo enmarca y restringe lo que debe implementar el desarrollador. Esto puede ser un beneficio, si el sistema está bien definido usando un diseño adecuado. Muy por el contrario, si la arquitectura del sistema no logra abarcar la completitud del software (incluyendo además escalabilidad, aumento de la cohesión, reducción del acoplamiento, reusabilidad, robustez, entre otros) el resultado obtenido al final del periodo de desarrollo es un programa inutilizable.

Por esta razón y muchas otras es que Mark Richards en su libro Software Architecture Patterns [22], define qué arquitectura de software uno debería utilizar. Para este caso, Richards estipula el uso de una arquitectura basada en Servicios. La Arquitectura de Servicios, es un patrón de diseño que está destinado a dichas aplicaciones que pretenden entregar una nueva funcionalidad o alternativa a otras aplicaciones. Normalmente, este diseño es desarrollado en unidades separadas, permitiendo un fácil desarrollo a través de una línea de trabajo efectiva, aumentando la escalabilidad y separando la aplicación de los componentes externos. Dado lo anterior, es que se ha definido el diseño arquitectónico de nuestra API tal como se presenta en la Figura 4.1.

La Capa de Servicios es la encargada de recibir las peticiones de los clientes, la cual reconoce qué es lo que se está pidiendo y designa a qué método debe hacer la llamada para posteriormente entregar una respuesta. Por otra parte los módulos *Reconocedor de Voz* y *Corrector de Palabra* son los contenedores destinados a con almacenar las funcionalidades detalladas en la Figura 4.1.

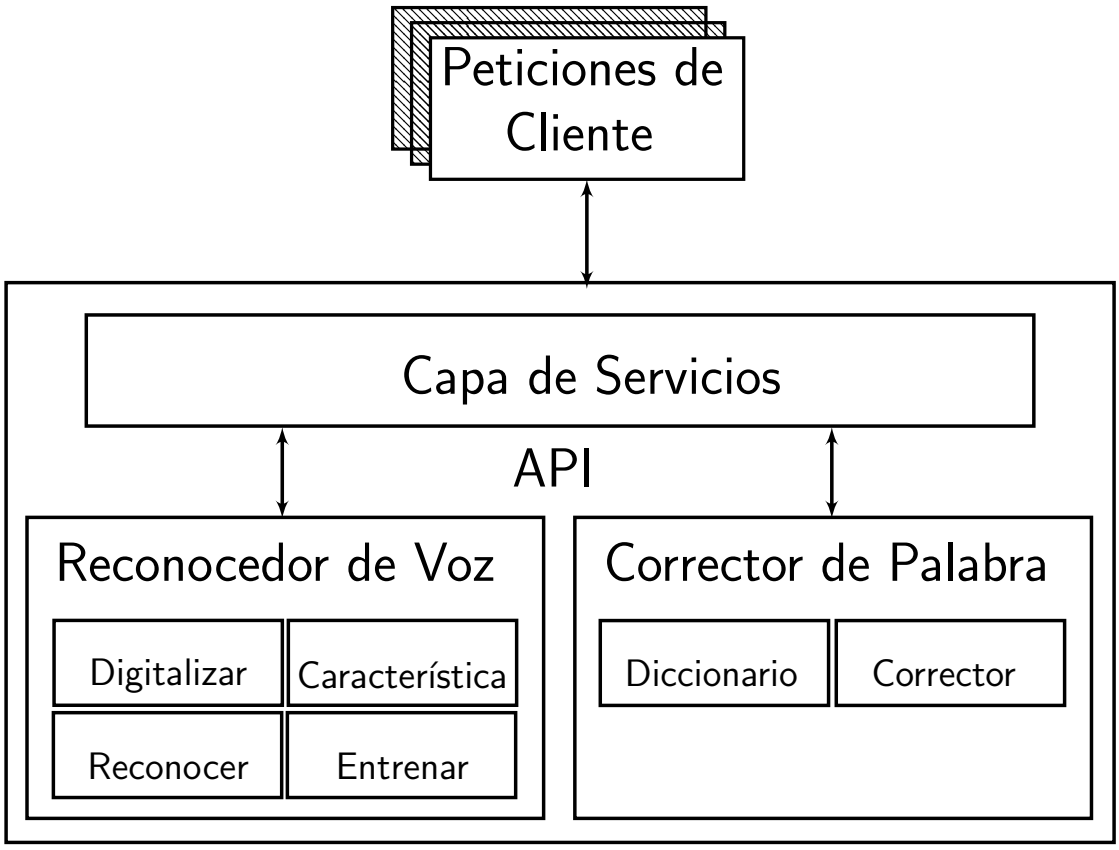

Figura 4.1: Arquitectura de la API.

4.2. Etapas del Proyecto

Dada la Arquitectura de Software seleccionada, es que el proyecto se divide en dos grandes etapas que permiten implementar los módulos presentes en la Figura 4.1. Dichas etapas corresponden a las siguientes:

1. Reconocimiento de Voz.
2. Corrector de palabras.

4.3. Metodología de Desarrollo

En cuanto a la metodología de desarrollo a utilizar, Mark Richards [22] propone que para arquitecturas basadas en servicios es mucho mejor trabajar con un proceso ágil; es por ello que esta memoria se desarrolla bajo *SCRUM*.

Todo el proceso de desarrollo es dividido en Sprints basados en las etapas nombradas anteriormente en la Sección 4.2, agregando una etapa extra al final para testing e integración. Hay que considerar que existen procesos como la creación de Bases de datos de entrenamiento, como un proceso paralelo al de ejecución de la solución.

Núm. Sprint	**Objetivo**	**Historias de Usuario**
Sprint 1	Transformación de audio análogo a digital y obtención de características de uno o más audios utilizando MFCC.	H1
Sprint 2	Creación de un modelo de red neuronal capaz de aprender a transcribir audios.	H1
Sprint 3	Creación de una base de datos de audio y sus respectivas transcripciones para el lenguaje Castellano.	H2
Sprint 4	Utilización del modelo previamente entrenado, para el reconocimiento de audios distintos al de entrenamiento.	H3
Sprint 5	En base al conjunto de palabras, responder cuál es la palabra que se ha intentado utilizar.	H4
Sprint 6	Creación de un diccionario de palabras en Castellano basado en verbos, pronombres, conectores, entre otros.	H5
Sprint 7	Revisión de las etapas e integración de ambos módulos (Reconocimiento de Voz y Corrector de palabras) en un servicio de respuesta (API).	H6, H7, H8

Cuadro 4.1: Etapas del Proyecto.

En el Cuadro 4.1 se detallan a grandes rasgos los Sprint utilizados para el desarrollo de la solución. Las historias de usuario, por su parte, pueden ser encontradas de forma detallada en el Anexo B.

4.3.1. Entorno de Trabajo

En cuanto al entorno de trabajo que se utiliza para el desarrollo de nuestra propuesta de solución se logran destacar ciertos aspectos importantes:

Lenguaje de Programación: La solución desarrollada se lleva a cabo bajo un lenguaje de programación orientado a objetos llamado *Python* en su versión 3.6.2.

Framework de Programación: Para la creación del modelo de red neuronal se utiliza *Tensorflow*, software creado por Google para implementaciones con Deep Learning, en su versión 1.2.0. Para más detalles ver su documentación [1].

IDE: Para escribir nuestra solución se hace uso de *Sublime Text*, un editor de texto avanzado, en su versión 3.0.

Equipo de Desarrollo: Todo el desarrollo se lleva a cabo en un equipo Macbook Pro mid 2012, con las siguientes características: procesador i5 de 2.5 GHz, 16Gb Ram 1600 MHz DDR3, macOS Sierra version 10.12.6.

Equipo de Pruebas: Las pruebas de entrenamiento se llevan a cabo en un servidor virtualizado con las siguientes características: 7 cores Intel Xeon E5-2620 de 2.4 GHz, 16Gb Ram 1600 MHz DDR3, Fedora 25.

Las pruebas de corrección de palabras se usaron en el mismo Equipo de desarrollo descrito anteriormente.

Controlador de versiones: Las distintas versiones del sistema están controladas y respaldas por *Git*. Dicho repositorio se encuentra online, pero de forma privada para mantener la propiedad intelectual de la empresa.

4.4. Modelo de la Red Neuronal

Al crear un modelo de red neuronal de cualquier tipo y propósito, es necesario determinar varios aspectos importantes:

- ¿Qué se recibe por entrada?
- ¿Cuántas neuronas se usa en la capa de entrada?
- ¿Cuántas capas ocultas se considera?
- ¿Cuántas neuronas tendremos en nuestra capa oculta?
- ¿Qué se produce por salida?
- ¿Cuántas neuronas se utiliza en la capa de salida?

Estas preguntas pueden ser determinantes al momento de crear dicho modelo, pero para responder a esto es necesario ir paso a paso.

4.4.1. Entradas

Para determinar qué se recibe por entrada a la red, simplemente debemos tener claro qué es lo que se desea hacer y qué valores de entradas se preservan entre uno y otro. Por ejemplo, no todos decimos "Hola"de la misma forma, normalmente encontraríamos diferencias inclusive en nuestras propias grabaciones.

Existen algoritmos que permiten tener una representación más cercana a la que realizamos los seres humanos y es el MFCC [11], el cual es un conjunto de filtros que permiten una obtención de características de mayor precisión ya que puede llegar a tomar la sensibilidad de la percepción humana en base a las frecuencias adquiridas.

Con esta nueva representación numérica ya podemos pasar dicha información a una red neuronal; pero antes es de suma importancia determinar cuántas entradas tendrá la red. Para ello simplemente utilizando la implementación de MFCC podemos obtener que los audios son divididos y redistribuidos en arreglos de un largo acotado, por lo que a simple vista podemos aplicar un cálculo para obtener largo máximo. En algunos frameworks de redes neuronales, esto no es necesario declararlo, ya que tienden a adecuarse a la entrada simplemente declarando que el largo de la entrada es desconocido.

4.4.2. Salidas

Inicialmente las salidas de la red neuronal corresponden a las letras del abecedario español, incluyendo el espacio y descontando los símbolos de puntuación. Esto suma un número de 34 neuronas de salida. Esta es una estimación general anticipada, ya que en base al entrenamiento dichas salidas pueden verse afectadas ya sea, por el lenguaje, la conjugaciones de letras, entre otros.

Aquí hay una desición crucial que se debe hacer, ya que existen dos posibilidades, reconocer las letras que conforman la palabra o simplemente reconocer las palabras completas. Si nos inclinamos por la segunda opción, nuestra red debería tener todas las palabras posibles, y si se quisiera reconocer algo que no se encuentre en el conjunto de entrenamiento, la red no sabría que responder. Por otra parte el entrenamiento debe contar con muchos datos de entrenamiento para cada una de las palabras, cosa tal es, en resumidas cuentas, imposible. Frente a esto, es que se escoge el entrenamiento basado en el reconocimiento de las letras que conforman las palabras.

4.4.3. Capas Ocultas

En base a la investigación realizada por Ms. P. Jennifer [13], se obtiene que para redes neuronales de reconocimiento del habla, no se presenta diferencias significativas usando más de una capa oculta. Gonzalo Acuña, en base a su basta trayectoria como profesor en la Universidad de Santiago de Chile (Usach) en el campo de la inteligencia artificial, sugiere que para calcular el número total de neuronas en la capa oculta N_w dado el largo de la Base de Datos de entrenamiento $|DB|$, es necesario utilizar las Ecuaciones 4.1 y 4.2 como una aproximación. Cabe destacar que ambas Ecuaciones deben cumplirse de forma simultanea.

$$N_w < \frac{|DB|}{10} \tag{4.1}$$

En base al número de neuronas de entradas N_e, el total de capas ocultas N_c y el número de neuronas de salida N_s, es posible estimar el número de neuronas de la capa oculta, tal como se muestra a continuación:

$$N_w = (N_e + 1) \cdot N_c + (N_c + 1) \cdot N_s \tag{4.2}$$

Conforme a lo descrito anteriormente, es que se decide la opción de utilizar sólo una capa oculta basada en un modelo de Red Neuronal LSTM.

Para dar solución al problema actual, se han considerado los siguientes valores:

- El número de entradas corresponde a la cantidad de características que entrega el algorítmo MFCC que por defecto es 13.
- El número de salidas, se ha definido como 34, para incluir el abecedario completo del castellano, incluyendo el uso de diéresis, tildes y simbología nueva para letras que usen combinaciones como *ch*, *pl*, *bl*, *pr*, entre otras.
- El número de capas ocultas es de 1.

Utilizando la Fórmula 4.2 descrita anteriormente, se obtiene que para el número de pesos en la capa oculta es de 82.

4.4.4. Capa de Clasificación Temporal

Debido a que nuestro modelo está basado en una red con retroalimentación, es necesario determinar que la salida sea la correcta. Cuando alguien dice *Hola*, nuestros oídos perciben la palabra como *hooola*, *holaaaa*, *hooolaaa*, entre otras. De la misma forma pasa con la salida de la red, por lo que es necesario utilizar algún mecanismo que permita dar solución a dicho problema. Frente a esto, nace la clasificación temporal de conexiones (CTC), algorítmo creado en 1990 [8], el cual consiste en una capa que en base a probabilidades calcula las etiquetas que deben activarse en momentos concretos.

Para utilizar esta función, es necesario introducir el símbolo vacío como un espacio en blanco en las posibles etiquetas que pueden emitir las redes neuronales recurrentes. De esta forma, la capa de salida de la red neuronal corresponde a probabilidades sobre todas las etiquetas posibles. Lo que se pretende lograr es que la red aprenda a encontrar por sí sola, cuál es la secuencia de etiquetas posibles. Usando la programación dinámica para sumar todas las secuencias posibles, CTC proporciona gradientes para la fase de retropropagación para entrenar a la RNN a aprender una buena selección de secuencias.

La fórmula de CTC, requiere la siguiente notación:

- y_k^t : la salida en el tiempo t para el símbolo k.

- L : la secuencia de etiquetas para la que queremos calcular un coste.
- L' : la misma secuencia de etiquetas pero con espacios en blanco entre cada letra.

Con lo anterior, es posible calcular la probabilidad de encontrar la secuencia x entre todas las secuencias de etiquetas posibles usando la Fórmula 4.3.

$$\mathcal{P}(\pi \mid x) = \prod_{t=1}^{T} y_k^t, \forall \pi \in L' \tag{4.3}$$

Ahora, la idea es poder obtener la suma, sobre todas las posibles trayectorias entre el paso de tiempo 1 y t que da como resultado las primeras etiquetas correctas s después de eliminar todas las etiquetas blancas y duplicadas. Esto se lleva a cabo usando la Fórmula 4.4.

$$\alpha_t(s) = \sum_{\pi \in N^T} \prod_{t'=1}^{t} y_{\pi_{t'}}^{t'} \tag{4.4}$$

Donde:

- $\pi \in N^T$: corresponde a todos los caminos posibles entre todas las etiquetas posibles desde el paso de tiempo 1 a t que dan las primeras etiquetas correctas s después de aplicar la transformación que elimina todos los espacios en blanco y duplicados.
- $y^{t'}$: corresponde a la salida de la red en el tiempo t'.

Dada la Fórmula 4.4, la respuesta del clasificador finalmente debe ser el etiquetado más probable para la secuencia de entrada:

$$\mathcal{H}(x) = \arg\max \alpha_t(s) \tag{4.5}$$

Esta etapa se vé simplificada con uso del Framework Tensorflow, ya que incluye esta funcionalidad en su implementación, pero es relevante tener presente que es importante contar con una etapa que rectifique la selección correcta de la palabra.

4.5. Conjunto de Datos de Entrenamiento

Para entrenar una red es necesario contar con un grupo de datos que reúna audios de distintas frases, junto con el archivo de transcripción. En base una etapa de búsqueda de un conjunto de datos de entrenamiento, se determinó que no es posible encontrar dicho conjunto de datos para el idioma castellano; esto pues, las bases de datos actuales para uso masivo están en idioma inglés. Por lo tanto, se decide fabricar un conjunto de entrenamiento en castellano, siguiendo el mismo formato de las bases en inglés, formato que se muestra en el Cuadro 4.2 y la Figura 4.2.

Para simplificar tanto la selección de frases y disminuir la complejidad de lectura de la base de datos de entrenamiento en castellano que se crea para este trabajo, es que se seleccionaron frases de libros para niños, generando un conjunto de 108 frases u oraciones de entrenamiento por cada persona. Los extractos son tomados de libros como:

1. Pinocho,
2. La gallina de los huevos de oro,
3. Los tres Chanchitos y
4. Blanca Nieves.

Cabe señalar que esta base de datos es sólo para pruebas de laboratorio, es decir, para corroborar que el modelo es correcto y es escalable con el solo hecho de ampliar dicho conjunto de entrenamiento.

La base de datos se compone de pares de audio-transcripción, junto con un documento de texto que detalla datos importantes del hablante como lo es su Nacionalidad, Ciudad, Género, Edad y un Identificador, separados por un espacio en blanco, tal como se muestra en el Cuadro 4.2.

La estructura de la base de datos consiste de carpetas, separando los audios de sus transcripciones tal como se muestra en la Figura 4.2. Para agregar nuevos audios de entrenamiento, simplemente debemos agregar los audios y transcripciones donde corresponde, como también no olvidar anexar al documento los datos de la persona. Cabe señalar que el formato de los audios debe ser *wav* y su transcripción en formato *txt* eliminando los saltos de línea.

ID	AGE	GENDER	COUNTRY	CITY
1	25	M	CHILE	ROMERAL
2	27	M	CHILE	CURICO
3	19	M	CHILE	LONTUE
4	23	F	CHILE	CURICO
5	21	M	CHILE	CURICO

Cuadro 4.2: Estructura Archivo Hablantes.

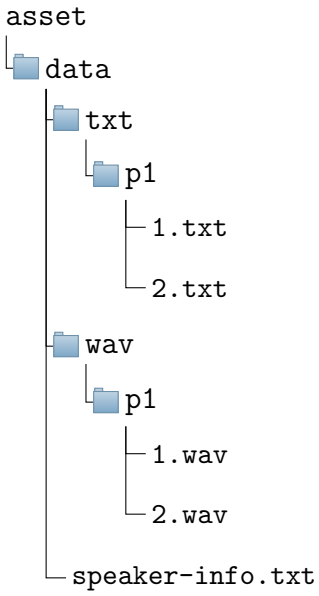

Figura 4.2: Estructura Base de Datos.

4.6. Entrenamiento

Por medio del entrenamiento del modelo aparecen detalles extras que a simple vista no son considerados. A juicio de este autor, los más grandes problemas en el área del reconocimiento de voz son contar con un conjunto de datos de entrenamiento, asegurar que dichos datos se encuentren correctos (validación de entrada) y los problemas fonéticos. Esto último es un problema grave para la forma de hablar de los Chilenos. Chile se caracteríza por tener un mal lenguaje y mayormente cuando se trata del habla. Por ejemplo, las letras *c*, *s*, *z*, *x* en este país no presentan ninguna diferencia fonética audible, lo que hace más díficil el entrenamiento de la red. Esto se ve reflejado en el Cuadro 4.3 donde muestra el sonido de las letras que presentan conflictos en Chile.

Letra	Caso	Sonido	Ejemplo
C	antes de la h	ch	Choque
	antes de e y de i	s	Cecilia
	después de la x	-	Excepción
	cualquier otro caso	k	Crocante
G	antes de e o de i	j	Género
	cualquier otro caso	g	Gato
H	cualquier caso	-	Ahora
L	antes de la l	y	Calle
	cualquier otro caso	l	Cola
Q	cualquier caso	k	Dique
R	cuando es la 1ra letra, cuando va después de una l, n, r, s, o cuando va antes de una consonante	rr	Carro
	cualquier otro caso	r	Pero
U	después de g o q, y antes de e o una i	-	Queso
	cualquier otro caso	u	Cuento
Ü	cualquier caso	u	Pingüino
W	cualquier caso	u	Walter
X	cualquier caso	ks	Exceso
Y	la palabra termina en una y	i	Buey
	cualquier otro caso	y	Rayo
Z	cualquier caso	s	Cazar

Cuadro 4.3: Sonido de letras críticas en Chile.

Para solventar estos problemas, se utilizan transformaciones de las palabras, de forma tal que reduzcan los errores fonéticos a costa de mayores errores léxicos. Ejemplo de ello, es eliminar las *h* de las palabras, considerar las *ch* como un caracter nuevo, etc.

4.7. Corrector de Palabras

Como es bien sabido, al hablar con alguien, nuestro cerebro interpreta qué palabra está intentando decir la otra persona basado en el contexto en que se esté hablando. Para dar solución a esto, estaríamos hablando de un proyecto de memoria por la magnitud del problema, pero existen alternativas que pueden ser muy útiles y que se conocen como corrector de palabras basado en diccionario.

Lo que hace este corrector es entregar la palabra más cercana a la deseada en base a un diccionario de palabras. Este problema se modela como un Espacio Métrico y se hace uso del Algoritmo de Levenstein (ver Algoritmo 1) como función para determinar cuál es la palabra dicha, usando la búsqueda de los vecinos cercanos (ver Algoritmo 2) para reducir el tiempo de consulta.

El problema de este tipo de corrector, es que la función de predicción de la palabra se hace respecto al cálculo de cuántas letras son las mínimas necesarias de cambiar, remover o intercambiar para llegar de la palabra actual a la palabra que marcamos como esperada. Esto suele tener conflictos en varios casos, ya que si por ejemplo alguien dice *varco*, el diccionario nos entrega como posibles palabras: *barco, marco, narco, parco, tarco, vareo, vario, vasco, zarco, arco, vaco* ya que todas estas palabras se encuentran a distancia 1.

Con el propósito de hacer frente a este problema, se utiliza una matriz de penalización que almacene el coste de inserción, eliminación o sustitución de cada letra, con la finalidad de priorizar algunos reemplazos en vez de otros. Para ello, se modifica el Algoritmo 1 que se encuentra en la Sección 2.3.1, agregando los costes de penalización como se muestra en el Algoritmo 4.

Algorithm 4 Levenshtein con matriz de penalización.

```
1: int Levenshtein(p1, p2):
2:     if (source == target) return 0
3:     (sp1, sp2) = (|p1|, |p2|)
4:     matrix←[sp1+1][sp2+1]
5:     for i ← 0 to |sp1+1|
6:         matrix[i][0]←i
7:     for j ← 0 to |sp2+1|
8:         matrix[0][j]←j
9:     for i ← 0 to |sp1|
10:        for j ← 0 to |sp2+1|
11:            cost ← p1[i] == p2[j] ? 0: penalization[p1[i]][p2[j]]
12:            matrix[i + 1][j + 1] ← min(matrix[i][j + 1] + penalization[ε][p2[j]],
    matrix[i + 1][j]+ penalization[p1[i]][ε], matrix[i][j] + cost)
13:    return matrix[sp1][sp2]
```

La matriz de penalización es una matriz de $N \times N$ para cada una de las letras del abecedario, incluyendo el ε que representa al carácter vaciío. Dicha matriz posee valores numéricos de penalizaciones para los rangos nulo (VL), bajo (L), medio (M) y alto (H) como se muestra en el Cuadro 4.4.

VL	ε	a	b	c	d	e	f	g	h	i
ε	VL	M	M	M	M	M	M	M	L	M
a	M	VL	H	H	H	M	H	H	H	H
b	M	H	VL	H	H	H	H	H	H	H
c	M	H	H	VL	H	H	H	H	H	H
d	M	H	H	H	VL	H	H	H	H	H
e	M	M	H	H	H	VL	H	H	H	M
f	M	H	H	H	H	H	VL	H	H	H
g	M	H	L	H	H	H	H	VL	H	H
h	M	H	H	H	H	H	H	H	VL	H
i	M	M	H	H	H	M	H	H	H	VL
j	M	H	H	H	H	H	H	L	H	H

Cuadro 4.4: Extracto Matriz de penalización.

4.8. Implementación de la API

La API es el servicio que encapsula tanto al módulo reconocedor de voz como al módulo de corrección de palabras. Esto permite a los usuarios que utilicen el servicio desvincularse de la implementación, simplemente restringiéndose a utilizar las operaciones que se entrega, de entre las cuales se destacan:

- reconocer(): función que permite al usuario reconocer el contenido de un audio, entregando la respuesta en formato de texto. Este método funciona bajo peticiones *POST*, entregando en el formulario de consulta el audio a reconocer. La respuesta a dicha consulta, se entrega en formato estándar *JSON*.

- corregir(): función que permite al usuario corregir una palabra basado en su diccionario interno, entregando la respuesta en formato *JSON*. Este método funciona bajo peticiones *GET*, entregando en la consulta la palabra a comprobar.

- agregar(): función que permite al usuario agregar una palabra en el diccionario interno, entregando una confirmación en formato *JSON*. Este método funciona bajo peticiones *POST*, entregando en la consulta la palabra a agregar.

Un ejemplo de consulta a la API es el siguiente `http://IP:5000/api/palabra/corregir/ggato`. En donde se declara utilizar el módulo de Corrector de Palabra de la API, y en particular el método corregir, entregando como parámetro a corregir la palabra *ggato*. Para este ejemplo la respuesta de la API es como se muestra a continuación:


```
[
  {
    "target": "ggato"
  },
  {
    "words": [
      "gato"
    ]
  }
]
```

En donde *target* corresponde al dato de consulta, y *words* las posibles palabras que se quisieron decir. Cabe destacar, que este es un ejemplo de consulta para formatos GET, en caso de ser POST, se debe realizar el formulario de consulta para mandar los datos ya sea por *Ajax*, *PHP*, *XMLHttpRequest*, entre otros.

Dentro de todo desarrollo existen prerrequisitos que deben ser cumplidos, ya sea de librerías instaladas como también en detalles de hardware. Es por ello que se recomienda utilizar equipos que cuenten con al menos las siguientes características:

- Procesador Pentium Dual Core.
- 4Gb Ram.
- 50Mb libre de disco.
- Python 3.
- Tensorflow 1.2.
- librerias de python: numpy, panda y python_speech_features

Para más detalles de la documentación de la API, modo de consulta, descripción de las funcionalidades, entre otros, ir al Anexo A.

5. Pruebas

5.1. Pruebas de Entrenamiento

En cuanto al entrenamiento se puede decir que es una de las etapas más complejas, que requiere un gasto enorme de tiempo; ya sea en recopilar datos para entrenar, como en validar dichos datos, entre otros factores.

Uno de los primeros cambios aspectos de interés que se puede obtener en base al entrenamiento es la importancia que tienen los datos de entrenamiento. Durante este proceso se obtiene que con simplemente refinar los audios de entrenamiento, lo que en este caso significa eliminar el ruido y las malas pronunciaciones, se puede obtener una mejora destacable.

En primera instancia, se hacen pruebas con los datos tal como fueron tomados, después refinando o readquiriendo los datos con problemas que suman un 66.6 % del contenido, y por último, el 33.4 % restante, obteniendo mejoras importantes en los resultados, los cuales se muestran en la Figura 5.1.

Es importante destacar, que al modificar los datos de entrenamiento, se requiere mucho menos iteraciones para alcanzar los resultados presentes en la Figura 5.1. En el Cuadro 5.1 se enlistan las épocas utilizadas para cada caso, usando los audios de 3 personas, correspondientes a 108 por cada uno, sumando un total de 324 audios de prueba.

Iteraciones	# Audios Corregidos	Porcentaje Error
750	0/324	80 %
300	216/324	48 %
120	108/324	42 %

Cuadro 5.1: Iteraciones vs correcciones.

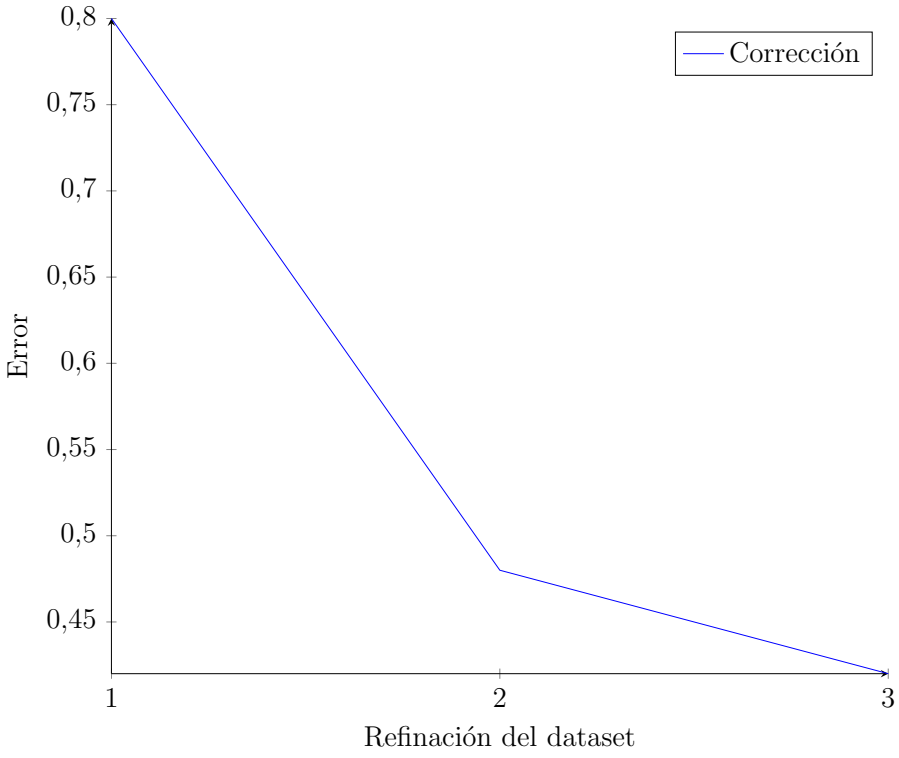

Figura 5.1: Porcentaje de error vs correciones en el dataset de prueba.

El problema de trabajar con Deep Learning es la captura de datos para entrenamiento. Contar con una gran cantidad de audios y su transcripción no es algo simple de realizar. En el intento de crear una base de datos propia, nos encontramos con el conflicto de seleccionar tanto un espacio donde grabar, gente que desee participar grabando audios de prueba y revisar los audios para regrabar los que se encuentren mal pronunciados.

Otro problema que es de una escala similar al anterior, es el tiempo que demora el entrenamiento. Por dar un ejemplo, una iteración para 324 archivos de prueba puede tomar un tiempo de 1 minuto, al aumentar dichos archivos a 1296, el tiempo

de prueba crece a 5 minutos y por último al aumentarlos a 43675 ya requiere una hora aproximado por iteración. Esta variación en los tiempos de prueba está dada por el largo de los audios de entrenamiento utilizados. Cabe destacar que todas las pruebas han sido hechas bajo un computador virtualizado que posee 7 núcleos y 16Gb de RAM.

5.2. Pruebas de Reconocimiento

En cuanto a las pruebas de reconocimiento, sinceramente hablando, es una de las etapas más complejas de llevar a cabo. En efecto, para obtener resultados representativos de un ambiente de producción, es necesario tener muchos datos de entrenamiento. Cosa tal es difícil de concretar al menos por una sola persona, ya que estamos hablando del orden de los cientos de miles de pares audio-transcripción. Las pruebas de reconocimientos hechas con la base de datos propia, entrega buenos resultados pero son resultados poco representativos, ya que al ser pocos datos de entrenamiento, el modelo se sobre entrena. La Figura 5.3 muestra la variación de error en el entrenamiento respecto a las iteraciones, en donde al avanzar en las iteraciones el porcentaje de equivocación es menor.

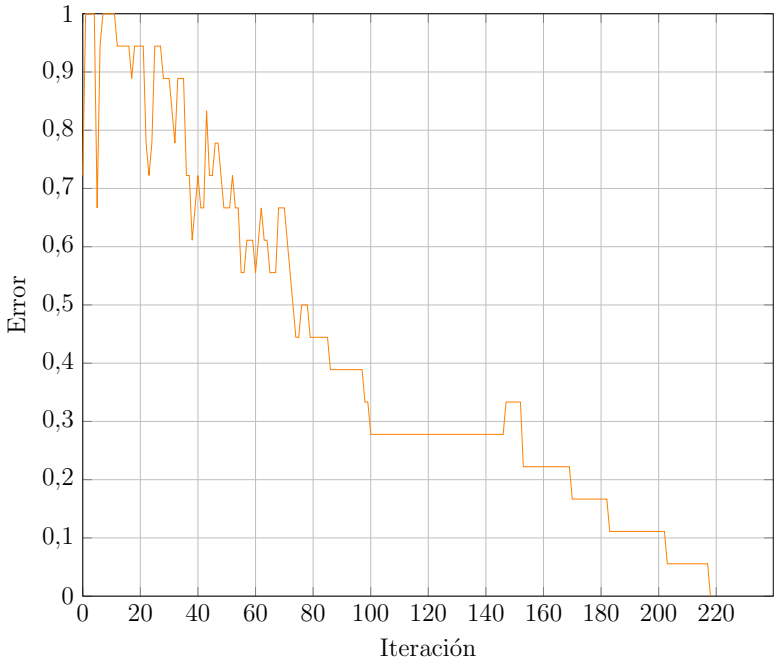

Figura 5.2: Error en Entrenamiento.

El problema de contar con un modelo que se encuentre sobre-entrenado es que si se hacen pruebas con otro archivo de audio que no se encuentre presente entre los de entrenamiento, el modelo responderá cualquier cosa. Esto se logra apreciar en la Figura 5.3 en donde el modelo calcula una secuencia de texto que no tiene sentido en español (Ver línea Resultado: et taes tba...). Este error es común y se puede producir por tener pocos datos de prueba, sólo contar con datos de pruebas ideales y no reales, o simplemente tener una base de datos desequilibrada.

```
Sergios-MacBook-Pro:v1 sflores$ python3 Main.py -f audio/3.wav
Restored Model
Resultado: esta es una prueba
Sergios-MacBook-Pro:v1 sflores$ python3 Main.py -f audio/4.wav
Restored Model
Resultado: et taes tbabpnestesesaetesebstbaueaestabab
```

Figura 5.3: Prueba de Reconocimiento.

En cuanto a pruebas de tiempo de respuesta, el modelo neural presenta los siguientes tiempos de respuesta, para audios de distintos largos:

Item	Largo audio	Tiempo extracción características	Tiempo reconocimiento
2	1s	1.697ms	0.135s
3	1s	5.001ms	0.110s
1	2s	1.884ms	0.157s
5	2s	1.718ms	0.153s
4	4s	2.548ms	0.300s
7	4s	2.506ms	0.282s
6	86s	29.300ms	5.535s

Cuadro 5.2: Tiempo de reconocimiento.

Dado los resultados del Cuadro 5.2, se logra apreciar que el tiempo de reconocimiento es bajo para audios de tamaños pequeños. Inicialmente el software está restringido a audios con un máximo de 10 segundos de duración, pero de igual forma se hacen pruebas con audios más prolongados, para corroborar el tiempo de respuesta. Este tiempo se puede estimar como $y = 0{,}0644 \cdot x$ según los resultados obtenidos, donde y es el tiempo de respuesta y x el largo del audio medido en segundos.

5.3. Pruebas de Correción de Palabras

La corrección de palabras es una etapa opcional que se concreta con la finalidad de mejorar la exactitud de respuesta del modelo de reconocimiento de voz. Considerando que los chilenos pronunciamos mal, eliminamos letras de las palabras o usamos palabras que no existen; este corrector pretende mitigar este tipo de problemas. Para efectos prácticos, se escogen palabras aleatoriamente entre mal escritas y escritas de buena manera, para comprobar como responde esta etapa tanto en exactitud como también en tiempo de respuesta. Estas palabras se muestran en el Cuadro 5.3.

Número	Palabra a evaluar	Respuesta
1	Buei	Buey
2	Mana	Maná
3	Ggato	Gato
4	Perroo	Perro
5	Canoa	Canoa
6	Gatito	Gatito
7	Escova	Escoba, Escoa
8	Simio	Simio
9	Estomago	Estómago
10	Carton	Cartón
11	Paralelepipedo	Paralelepípedo
12	Esternocleidomastodi	Esternocleidomastoideo
13	tualla	Talla

Cuadro 5.3: Extracto del Conjunto de Palabras de Prueba.

El porcentaje de exactitud para este corrector bordea el 93.8 %, encontrando complicaciones al utilizar palabras tildadas como parámetro de entrada. Esto no es un problema para efectos prácticos, ya que el modelo de reconocimiento de voz no responde con carácteres con tildes o con diéresis, entre otros.

Si evaluamos el corrector basados en el tiempo de respuesta, obtenemos un tiempo promedio de 0.4993 segundos con un tiempo agregado de 0.243 segundos para consultas por red, tal como se muestra en la Figura 5.4. Se debe considerar que,

además del tiempo real de cálculo (que en este caso, apenas es el 1 % del tiempo total), este tiempo incluye el empleado por el sistema operativo, y por los mensajes a través de la red. Cabe señalar que si la palabra no existe en el diccionario, el costo máximo de espera por el usuario es de 1s aproximadamente.

En efecto, cuando se ejecuta de forma local el conjunto de pruebas, vale decir, sin tomar en cuenta el sobrecosto de tiempo que introduce la utilización de una API Web, se obtiene que en promedio, la respuesta requiere 0.4967 segundos, lo cual significa que el hecho de usar una API casi no enlentece los tiempos de respuesta.

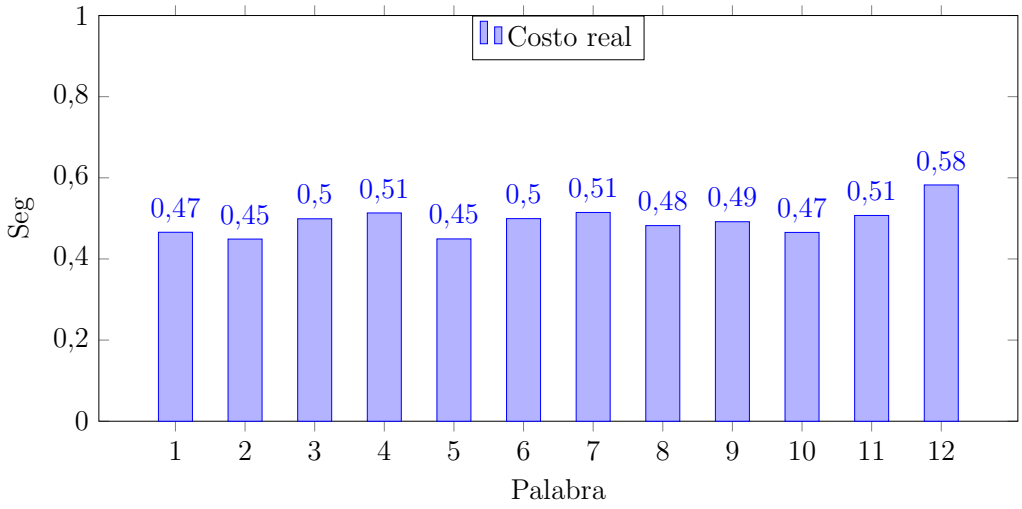

Figura 5.4: Costo de Corrección.

Para conseguir los resultados del Cuadro 5.3, es necesario evaluar el sistema variando la cantidad de pivotes para encontrar el valor que entregue el mejor resultado. Para uso de strings, con 7 pivotes (un $\alpha = 0.83$-0.86) basta para generar una respuesta óptima en términos de tiempo de procesamiento tal como se muestra en el Cuadro 5.4.

Como se logra apreciar en el Cuadro 5.4, al cambiar a un $\alpha = 0.87$, los pivotes resultantes son 4 y con un $\alpha = 0.82$ son 17 pivotes. Esto ocurre, ya que los pivotes se seleccionan en base al largo de la palabra que en promedio es de 8.67 caracteres (ver Figura 5.5). Estos largos son valores enteros entre 1 y 23, siendo este último el largo máximo o $dMax$. Para el cálculo de pivotes se utiliza el Algorítmo 3, el cuál contempla la distancia máxima, siendo esta un dato atípico según la Figura 5.5, por ello es que entrega esta variación tan amplia de selección de pivotes.

Alpha	Pivotes	Tiempo
1	1	9s
0.9	1	9s
0.87	4	2s
0.86	7	0.5s
0.85	7	0.5s
0.84	7	0.5s
0.83	7	0.5s
0.82	17	0.8s
0.8	17	0.8s
0.7	39	1.7s

Cuadro 5.4: Conjunto de Palabras de Prueba.

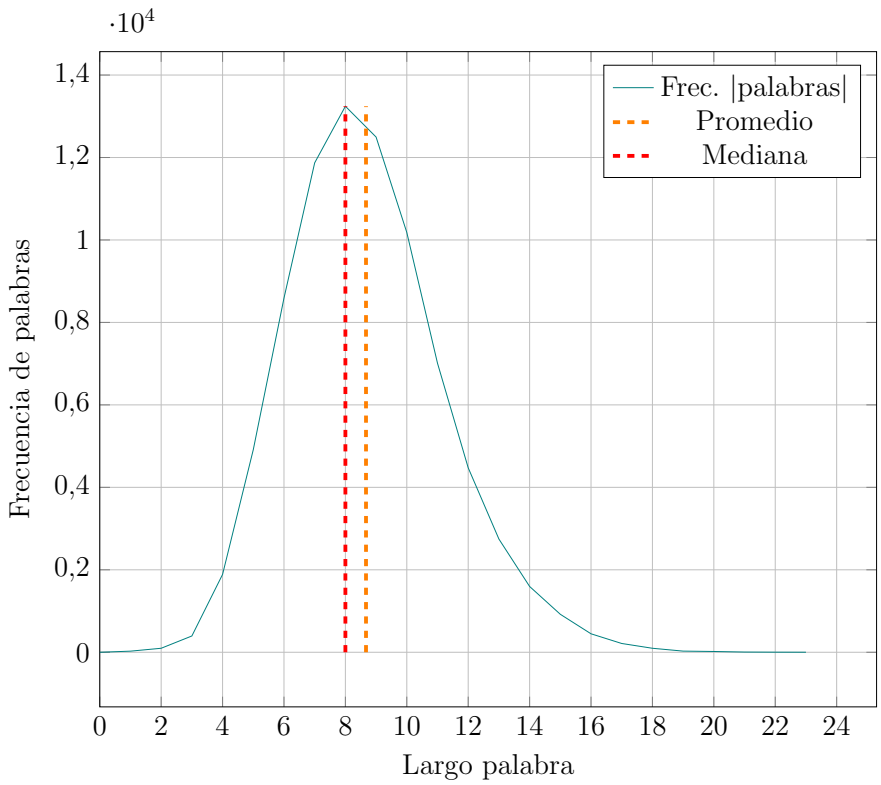

Figura 5.5: Histograma del largo de palabras del diccionario utilizado.

5.4. Resultados

Dado el trabajo realizado, y los experimentos posteriores, es que se obtienen varios resultados los cuales se detallan a continuación:

Lamentablemente, y se debe decir, español es un problema, partiendo por cómo hablamos, luego cómo escribimos y terminando en la estructura del lenguaje. Este idioma posee caracteres que no se encuentran en el Inglés, idioma mundial como referencia, por lo que hasta los lenguajes de programación requieren de una representación distinta para estos. Por dar un ejemplo, la letra á en *Python* se representa como un $\tilde{A}¡$ o en otras ocaciones como $\backslash xc3\backslash xa1$ dependiendo de la versión que se esté usando. Este es un problema grave al momento de generar comparaciones entre strings, ya que se insertan dos caracteres extras, por lo que la comparación basado en distancia presenta problemas para dichas palabras.

Por otra parte, el español posee uniones de caracteres que presentan un sonido muy similar fonéticamente hablando, y que en el caso de Chile, este sonido es el mismo, e.g. Casa - caza, hay - ay, valla - baya - vaya, ves - vez, ola - hola, entre otros. Si meditamos en esto, claramente si para los humanos muchas veces puede causar conflictos de aprendizaje o discriminación, cuanto más a una máquina que necesita reconocer de buena forma qué es lo que se está tratando de decir.

El tiempo de entrenamiento de un modelo neural con Deep Learning para reconocimiento de voz es muy extenso. Estamos hablando que para bases de datos de producción puede demorar meses. Google declara haber invertido unos 8 meses de sólo entrenamiento [10]. Considerando los grandes servidores que estos poseen, ese resultado sería similar al que uno podría alcanzar con una base de datos de aproximadamente unos 100mil archivos de prueba en un computador personal o servidor privado de similares características a los detallados en la Sección 4.3.1. Este valor se estima en base al tiempo que demora el entrenamiento con 324, 1296 y 43675 pares de datos de prueba. Hay que tener presente que la base de datos a crear debe estar compuesta por diferentes interlocutores, ya que en caso contrario, el modelo al ser entrenado intenta reconocer a la persona más que al lenguaje. Esto se vió reflejado durante el proceso de entrenamiento del modelo neural.

Al evaluar el modelo neural debemos comprobar que mecanismo de evaluación nos entrega resultados más cercanos a la respuesta obtenida. Inicialmente se usaron comparaciones de oraciones, lo cual fue un grave error. Por el simple hecho de contar

con una palabra mal transcrita marcaba la oración completa como mala. La primera opción que uno considera es usar la comparación por letras, pero en realidad uno no habla letras sino que palabras. De igual forma pasaría algo similar a la versión de comparación por oración ya que si la primera letra es errónea las demás también lo serán. Ej. "ola cómo estás" en vez de "hola cómo estás".

Dado lo anterior es que probamos con una comparación de palabras, lo cual entrego mejores resultados de los esperados. El modelo actualmente responde a un 83.3 % según la evaluación, entregando información relevante que no fue planificada. Como conclusiones finales se tiene que:

- El modelo reconoce de mejor forma oraciones que presenten palabras de largos pequeños.
- Las palabras que normalmente nunca fallan son artículos como el, la, los, las, una, un, etc.
- Las palabras más complejas de reconocer son aquellas que se desconoce su pronunciación. Ex. Gepeto, la multitud desconoce si se pronuncia "jepeto" o "yepeto".

6. Conclusiones y Trabajo futuro

6.1. Conclusión

Dada la implementación de un reconocedor de voz basado en redes neuronales, es que se logra comprender la razón por la cuál los usuarios que necesitan usar esta tecnología terminan pagando por ella, ya que es una tarea compleja de llevar a cabo por una sola persona. En primera instancia se necesitan cientos de miles de pares audios - transcripción para entrenamiento y/o pruebas, por otra parte se debe asegurar que este conjunto se encuentre validado y, finalmente, invertir en muchas horas de entrenamiento.

Los chilenos hablamos mal, eso es un hecho, sólo basta hablar con uno por un lapso tiempo corto para afirmarlo. Estos normalmente tienden a eliminar las "s" finales, "b" intervocálicas, "d" finales o intervocálicas, entre otras, causando mayores conflictos en el reconocimiento de voz que en países que practiquen el mismo lenguaje.

La selección de las personas a grabar como el lugar de grabación juegan un rol importante en el reconocimiento de voz, ya que de ello depende la precisión del modelo y cuánto trabajo extra se necesitará para asegurar que el audio concuerde con la transcripción.

Al crear una base de datos de entrenamiento, se debe considerar que es necesario contar con muchas personas ya que si son pocas el modelo se sobre entrena centrándose en características vocales de los locutores más que en el lenguaje en sí.

Si fuese necesario escoger una herramienta para el reconocimiento de voz luego de haber trabajado en la implementación propia, se recomienda el uso de un servicio externo como los que entregan las grandes empresas como Google, Microsoft o IBM. Uno puede pensar que la decisión de usar un servicio externo pasa por el consumo en disco o el tiempo de respuesta, pero no es así.

Cabe señalar que este tipo de sistema no requiere de mucho espacio en disco para funcionar, ya que el modelo neural entrenado no supera los 10Mb. Por lo que el problema no pasa por espacio en disco, si no que simplemente se focaliza en el porcentaje de error que uno pueda aceptar y qué tan largos son los audios que uno desee transcribir. Para el entrenamiento es donde se necesitan más recursos, ya que con una base de datos de 45mil pares de archivos se invierte unos 18Gb por parte baja para almacenarlos, esto puede fluctuar dependiendo de la calidad de los audios y de su largo.

6.2. Trabajo Futuro

El trabajo futuro involucra tanto mejoras en la exactitud de respuestas como alternativas extras que amplíen la capacidad de la API en general. Para llevar estas propuestas a la realidad, una alternativa es crear una base de datos de entrenamiento más grande, la cual bordee los 100 mil pares audio-transcripción, para verificar como mejora la capacidad del modelo de reconocimiento de voz para lenguaje Castellano. Se debe tener en consideración que al generar dicha base datos de entrenamiento, ésta debe contener audios grabados por distintos locutores, ya que en caso contrario el entrenamiento se centra en características vocales de ellos y no en el lenguaje en sí.

Otra opción es incluir una etapa de corrección de semántica y/o incluir una etapa de corrección gramatical. Con esto último, ya la API podría dar una respuesta más sólida y concisa, de igual forma como lo hace el buscador de Google presentando la frase típica conocida por todos, *quizás quisiste decir*.

Se debe considerar que las mejoras demandarán mucho tiempo de trabajo, ya que el simple hecho de crear una base de datos mayor requiere un coste de tiempo que puede ser fácilmente un año o una memoria de título.

Glosario

RNA: Término comúnmente utilizado para referirse a una Red Neuronal Artificial.

LSTM: Tipo de Red Neuronal, en donde sus siglas provienen del inglés *Long Short-Term Memory*, que tiene por significado Gran Memoria de corto plazo.

RNN: Término comúnmente utilizado para referirse a una Red Neuronal Recurrente, es decir, que presenta retroalimentación en sus entradas.

MFCC: Conjunto de filtros de frecuencias que permiten obtener la sensibilidad de la percepción de un audio de forma similar a la realizada por los humanos.

CTC: Se traduce como Clasificación Temporal de Conexiones, es un algoritmo en base a probabilidades calcula las etiquetas que deben activarse en momentos concretos.

API: Sus siglas se traducen del inglés a Interfaz de Programación de Aplicaciones, el cual hace referencia a los procesos, las funciones y los métodos que brinda una determinada biblioteca de programación.

SSS: Sparse Spatial Selection, es un algorítmo que garantiza una buena selección de los pivotes en una búsqueda por similitud en espacios métricos

SDK: Es un conjunto de herramientas de desarrollo de software que le permite al programador crear aplicaciones informáticas.

JSON: (JavaScript Object Notation) es un formato para el intercambios de datos, empleado para identificar y gestionar información.

Bibliografía

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Apple. Siri personal asistent. `http://www.apple.com/ios/siri/`, Octubre 2011. Acceded: 28 de Marzo del 2017.

[3] Rodero Antón Emma. El tono de la voz masculina y femenina en los informativos radiofónicos: un análisis comparativo. `https://goo.gl/TXgRyP`, Noviembre 2001.

[4] Florian Eyben, Felix Weninger, Florian Gross, and Björn Schuller. Recent developments in opensmile, the munich open-source multimedia feature extractor. In *Proceedings of the 21st ACM International Conference on Multimedia*, MM '13, pages 835–838, New York, NY, USA, 2013. ACM.

[5] K. Figueroa, G. Navarro, and E. Chávez. Metric spaces library, 2007. Available at `http://sisap.org/?f=library`.

[6] R García Martínez. *Sistemas Autónomos. Aprendizaje Automático.* ISBN 950-9088-84-6. Ed. Nueva Librería., 1997.

[7] Google. Speech api. `https://cloud.google.com/speech/`, Agosto 2016. Acceded: 28 de Marzo del 2017.

[8] Alex Graves, Santiago Fernández, and Faustino Gomez. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the International Conference on Machine Learning, ICML 2006*, pages 369–376, 2006.

[9] Donald O. Hebb. *The organization of behavior: A neuropsychological theory*. Wiley, New York, June 1949.

[10] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, 29:82–97, 2012.

[11] Abdelmajid H.mansour, Gafar Zen Alabdeen Salh, and Khalid A. Mohammed. Voice recognition using dynamic time warping and mel-frequency cepstral coefficients algorithms. *International Journal of Computer Applications*, 116(2):34–41, 2015.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[13] MSarad P Jennifer. Speech recognition finding number of hidden layers – neural networks. *International Journal of Computer Trends and Technology (IJCTT)*, 2(1):26–36, Sep 2011.

[14] Eric Jones, Travis Oliphant, and Pearu Peterson. SciPy: Open source scientific tools for Python, 2001–. [Online; accedido en Junio].

[15] Laveen N. Kanal. Perceptron. In *Encyclopedia of Computer Science*, pages 1383–1385. John Wiley and Sons Ltd., Chichester, UK, 2003.

[16] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[17] Microsoft. What is cortana? `https://goo.gl/Kr70rc`, Marzo 2017. Acceded: 27 de Marzo del 2017.

[18] Neurotechnology. Verispeak sdk. `https://goo.gl/DoyrZF`, Junio 2016. Acceded: 30 de Marzo del 2017.

[19] H. Nyquist. Certain factors affecting telegraph speed. *Bell System Technical Journal*, 3(2):324–346, 1924.

[20] A. Papadopoulos. *Metric Spaces, Convexity and Nonpositive Curvature*. IRMA lectures in mathematics and theoretical physics. European Mathematical Society, 2005.

[21] Oscar Pedreira and Nieves R. Brisaboa. Spatial selection of sparse pivots for similarity search in metric spaces. In *SOFSEM 2007: Theory and Practice of Computer Science, 33rd Conference on Current Trends in Theory and Practice of Computer Science*, volume LNCS 4362, pages 434–445, 2007.

[22] Mark Richards. *Software Architecture Patterns*. O'Reilly Media, Inc., 2015.

[23] David A. Sousa. *How the Brain Learns / Cómo Aprende el Cerebro.* ISBN 9780761946663. International Educational Consultant, 2002.

[24] Akanksha Singh Thakur and Namrata Sahayam. Speech recognition using euclidean distance. *International Journal of Emerging Technology and Advanced Engineering*, 3(3):34–41, 2013.

[25] DeLiang Wang. Deep learning reinvents the hearing aid. *IEEE Spectrum Magazine*, 54:32 – 37, 2017.

[26] R. K. Rao Yarlagadda and John E. Hershey. *Hadamard Matrix Analysis and Synthesis: With Applications to Communications and Signal/Image Processing*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

ANEXOS

A. Formato de Documentos Oficiales Utilizados

Durante el proceso de memoria, se utilizaron distintos documentos para lograr un vínculo entre la empresa *Exe Ingeniería*, quienes fueron los procursores de esta iniciativa, y la Universidad, por lo que cada proceso requiere contar con documentos que comprometan a ambas partes, de los cuales se entregan a continuación.

A.1. Formato de Listado de Requisitos

Validación de Requisitos

Título del proyecto: "API de Reconocimiento de Voz para lenguaje Castellano usando Redes Neuronales"

El propósito de este documento es dejar constancia del listado de requisitos del sistema a implementar, impidiendo que existan déficit en el software a desarrollar, o se incorporen trabajos extras no especificados en primera instancia:

Al respecto se exponen los siguientes requisitos:

Requisitos de usuario

RU001	
Descripción :	El sistema debe estar perfectamente comentado de forma tal que cualquier desarrollador pueda comprender y realizar cambios futuros.
Fuente :	Empresa
Prioridad :	4
Estabilidad :	Necesario
Fecha Actualización :	12 de Mayo del 2017
Estado :	Cumple
Incremento :	1
Tipo :	Usuario

RU002	
Descripción :	El sistema debe responder a lo que se ha dicho, teniendo presente que se pueden usar palabras mal dichas.
Fuente :	Empresa
Prioridad :	4
Estabilidad :	Necesario
Fecha Actualización :	12 de Mayo del 2017
Estado :	Cumple
Incremento :	3
Tipo :	Usuario

Requisitos Funcionales

RF001	
Descripción :	Implementar un algoritmo adaptable que pueda aprender del proceso dinámico de autogeneración del lenguaje dado un audio.
Fuente :	Empresa
Prioridad :	1
Estabilidad :	Intransable
Fecha Actualización :	12 de Mayo del 2017
Estado :	Cumple
Incremento :	3
Tipo :	Funcional

RF002	
Descripción :	Crear una red neuronal que pueda ser entrenada con audio y texto.
Fuente :	Empresa
Prioridad :	1
Estabilidad :	Intransable
Fecha Actualización :	12 de Mayo del 2017
Estado :	Cumple
Incremento :	3
Tipo :	Funcional

RF003	
Descripción :	Generar un proceso masivo de entrenamiento que permita ampliar el rango de experiencia inicial.
Fuente :	Empresa
Prioridad :	2
Estabilidad :	Intransable
Fecha Actualización :	12 de Mayo del 2017
Estado :	Cumple
Incremento :	4
Tipo :	Funcional

RF004	
Descripción :	La respuesta del sistema debe ser para el lenguaje *Castellano*.
Fuente :	Empresa
Prioridad :	4
Estabilidad :	Intransable
Fecha Actualización :	12 de Mayo del 2017
Estado :	Cumple
Incremento :	4
Tipo :	Funcional

RF005	
Descripción :	Generar una base de datos inicial para el entrenamiento, que pueda ser escalable en un futuro.
Fuente :	Empresa
Prioridad :	1
Estabilidad :	Intransable
Fecha Actualización :	12 de Mayo del 2017
Estado :	Cumple
Incremento :	2
Tipo :	Funcional

RF006	
Descripción :	Anexar una etapa correctora de léxico.
Fuente :	Empresa
Prioridad :	1
Estabilidad :	Opcional
Fecha Actualización :	12 de Mayo del 2017
Estado :	Cumple
Incremento :	5
Tipo :	Funcional

RF007	
Descripción :	Incluir un módulo corrector de sintáctico.
Fuente :	Empresa
Prioridad :	1
Estabilidad :	Opcional
Fecha Actualización :	12 de Mayo del 2017
Estado :	Cumple
Incremento :	5
Tipo :	Funcional

Si considera que estos requisitos cumplen con las necesidades de la empresa, entonces por medio de este documento doy por firmado que lo declarado anteriormente es correcto.

Yo .....

Cédula de identidad N^{o}: .....

Encargado de: .....

Empresa: .....

Doy por aprobado dicho listado de requisitos con fecha: ..... / ..... / .....

.....

Firma

A.2. Formato Consentimiento Informado

Consentimiento informado

Título del proyecto: "API de Reconocimiento de Voz para lenguaje Castellano usando Redes Neuronales"

Sr: .....

El propósito de este documento es entregarle toda la información necesaria para que usted pueda decidir libremente si desea participar de la investigación que se le ha explicado verbalmente y que a continuación se describe de forma resumida:

La investigación busca generar una base de datos para entrenar un software que permita en un futuro completar formularios de manera automática. La información a recopilar será en formato de audio, utilizando frases comunes utilizadas en castellano.

Al respecto expongo que:

- He sido informado sobre el estudio a desarrollar y que consecuencias podría traerme en caso de que mi información caiga en manos equivocadas.
- Estoy en pleno conocimiento que la información obtenida en la actividad en la que participaré, será absolutamente confidencial, y que no aparecerán mis datos personales, ni mi nombre en libros, revistas y ni otros medios de publicación derivadas de la investigación.
- Sé que la decisión de participar en esta investigación, es totalmente voluntaria. Si no deseo seguir colaborando o participando una vez iniciada la investigación, puedo hacerlo sin problemas. En ambos casos se me asegura que mi negativa no implica ninguna consecuencia negativa para mí.
- He leído el documento, entiendo las declaraciones contenidas en él y la necesidad de hacer constar mi consentimiento, para lo cual firmé libre y voluntariamente.

Yo .....
Cédula de identidad Nº: ....., con domicilio en:
.....
Consiento en participar en esta investigación.
Fecha: ....., Firma de la persona que consiente: .....

A.3. Firma Aprobación de Requisitos

Validación de Requisitos

Título del proyecto: "API de Reconocimiento de Voz para lenguaje Castellano usando Redes Neuronales"

El propósito de este documento es dejar constancia del listado de requisitos del sistema a implementar, impidiendo que existan déficit en el software a desarrollar, o se incorporen trabajos extras no especificados en primera instancia:

Al respecto se exponen los siguientes requisitos:

Requisitos de usuario

RU001	
Descripción :	El sistema debe estar perfectamente comentado de forma tal que cualquier desarrollador pueda comprender y realizar cambios futuros.
Fuente :	Empresa
Prioridad :	4
Estabilidad :	Necesario
Fecha Actualización :	12 de Mayo del 2017
Estado :	Cumple
Incremento :	1
Tipo :	Usuario

RU002	
Descripción :	El sistema debe responder a lo que se ha dicho, teniendo presente que se pueden usar palabras mal dichas.
Fuente :	Empresa
Prioridad :	4
Estabilidad :	Necesario
Fecha Actualización :	12 de Mayo del 2017
Estado :	Cumple
Incremento :	3
Tipo :	Usuario

Requisitos Funcionales

RF001	
Descripción :	Implementar un algoritmo adaptable que pueda aprender del proceso dinámico de autogeneración del lenguaje dado un audio.
Fuente :	Empresa
Prioridad :	1
Estabilidad :	Intransable
Fecha Actualización :	12 de Mayo del 2017
Estado :	Cumple
Incremento :	3
Tipo :	Funcional

RF002	
Descripción :	Crear una red neuronal que pueda ser entrenada con audio y texto.
Fuente :	Empresa
Prioridad :	1
Estabilidad :	Intransable
Fecha Actualización :	12 de Mayo del 2017
Estado :	Cumple
Incremento :	3
Tipo :	Funcional

RF003	
Descripción :	Generar un proceso masivo de entrenamiento que permita ampliar el rango de experiencia inicial.
Fuente :	Empresa
Prioridad :	2
Estabilidad :	Intransable
Fecha Actualización :	12 de Mayo del 2017
Estado :	Cumple
Incremento :	4
Tipo :	Funcional

RF004	
Descripción :	La respuesta del sistema debe ser para el lenguaje *Castellano*.
Fuente :	Empresa
Prioridad :	4
Estabilidad :	Intransable
Fecha Actualización :	12 de Mayo del 2017
Estado :	Cumple
Incremento :	4
Tipo :	Funcional

RF005	
Descripción :	Generar una base de datos inicial para el entrenamiento, que pueda ser escalable en un futuro.
Fuente :	Empresa
Prioridad :	1
Estabilidad :	Intransable
Fecha Actualización :	12 de Mayo del 2017
Estado :	Cumple
Incremento :	2
Tipo :	Funcional

RF006	
Descripción :	Anexar una etapa correctora de léxico.
Fuente :	Empresa
Prioridad :	1
Estabilidad :	Opcional
Fecha Actualización :	12 de Mayo del 2017
Estado :	Cumple
Incremento :	5
Tipo :	Funcional

RF007	
Descripción :	Incluir un módulo corrector de sintáctico.
Fuente :	Empresa
Prioridad :	1
Estabilidad :	Opcional
Fecha Actualización :	12 de Mayo del 2017
Estado :	Cumple
Incremento :	5
Tipo :	Funcional

Si considera que estos requisitos cumplen con las necesidades de la empresa, entonces por medio de este documento doy por firmado que lo declarado anteriormente es correcto.

Yo Luis González Martínez

Cédula de identidad Nº: 16004129-3

Encargado de: Jefe Centro de I+D+i

Empresa: EXE Ingeniería y Software

Doy por aprobado dicho listado de requisitos con fecha: 19/05/2017

Firma

A.4. Documentación API

Documentación API

API de Reconocimiento de Voz para lenguaje Castellano usando Redes Neuronales

Este documento ofrece una descripción detallada de los métodos disponibles, con ejemplos de las respuestas obtenidas, así como información acerca del uso de nuestra API.

1. Consideraciones

Algunas consideraciones para tener en cuenta:

- Las peticiones de la API deben ser realizadas con peticiones GET o POST según el caso.
- Puede considerar todas las respuestas que no sean devueltas con un código HTTP 200 como un error.
- Todas las respuestas son devueltas en formato JSON.

2. API Version

1.1.0

3. Interactuar con la API

Si quiere probar nuestra API de una manera rápida le recomendamos el uso de *PostMan*. Postman es una extensión de *Google Chrome* que ahora posee su propio software de escriorio, el cuál permite interactuar con HTTP API's de forma sencilla a través de una interfaz amigable para construir peticiones y obtener respuestas.

Lenguaje	HTTP client libraries
PHP	cURL, Guzzle
Python	urlLib2 + MultipartPostHandler

4. Formato de respuestas de la API

Siempre que realice una petición correcta a la API obtendrá un JSON y un código de respuesta HTTP 200.

```
{
  "respuesta": {
    ...
  }
}
```

Por el contrario, las peticiones incorrectas devolverán un JSON con el atributo error y un código de respuesta diferente a HTTP 200.

```
{
  "error": {
     "error information"
  }
}
```

5. Métodos API

5.1. Reconocimiento de Voz

En este punto hay que tener presente que para reconocer la voz, debemos contar con un modelo previamente entrenado. Esto se debe realizar de forma manual y previo a la etapa de reconocimiento.

5.1.1. Reconocer

Se utiliza para obtener la transcripción de un archivo de audio en formato de JSON.
Formato: POST.
Ejemplo de URI: https://IP:PORT/api/voz/reconocer
Parámatros URI: *file* (requirido), archivo de audio a transcribir en formato wav.
Respuesta: *result*, transcripción del audio en formato String.
error(optional), información de error ocurrido.

5.2. Corrector de Palabra

5.2.1. Corregir

Se utiliza para obtener la corrección basado en diccionario de una palabra en formato de JSON.
Formato: GET.
Ejemplo de URI: https://IP:PORT/api/palabra/corregir/target
Parámatros URI: *target* (requirido), palabra que se desea corregir.

Respuesta: *target*, palabra consultada.
words, arreglo de posibles palabras.
error(opcional), información de error ocurrido.

5.2.2. Agregar palabra

Se utiliza para agregar una palabra al diccionario de palabras. Se debe tener presente que la palabra no estará disponible hasta que se reinicie el servicio.

Formato: POST.
Ejemplo de URI: https://IP:PORT/api/palabra/agregar/
Parámatros URI: *word* (requirido), palabra que se desea agregar al diccionario.
Respuesta: *result*, respuesta booleana del logro de inserción de la palabra.
error(opcional), información de error ocurrido.

6. Entrenamiento Modelo Reconocimiento

En caso de no contar con el modelo de reconocimiento o querer ampliar la base de reconocimiento es necesario entrenar el modelo. Esto se debe hacer de forma manual, ya que demora mucho tiempo el entrenamiento (meses).

Para iniciar el entrenamiento, diríjase a la carpeta de reconocimiento de voz y ejecute el programa *training.py* de la siguiente forma:

```
python3 training.py
```

Debe tener en consideración que el equipo en donde se esté ejecutando el entrenamiento estará innutilizable hasta que termine el entrenamiento.

7. Requisitos de Hardware

Para utilizar la API necesita contar con un dispositivo que trabaje como servidor, el cuál puede ser el mismo que se utilice para mantener la aplicación que consuma de los servicios de la API. Dicho computador debe tener las siguientes prestaciones como mínimo:

- Procesador pentium Dual Core.
- 4Gb re RAM.
- 50Mb libres de disco.

Para entrenar el modelo, debe tener presente que deberá tener espacio de disco extra para almacenar los datos de entrenamiento.

8. Requisitos de Software

Como requisitos de software se recomienda tener instalado la version 3.6 de Python.

Además, la API requiere de instalar un par de librerías de Python las cuales se enlistan a continuación:

- python_speech_features
- tensorflow
- numpy
- panda

Por otro lado, el sistema operativo debe contar con las librerias *six*, *wheel*, *bazel*. En caso de ser un Mac, debe tener instalado *brew*.

Para instalar estas aplicaciones se enlistan los comandos para MacOSX:

```
brew install pip3
pip3 install python_speech_features
pip3 install tensorflow
brew install bazel
sudo easy_install -U six
sudo easy_install wheel
```

9. Contacto

Frente a cualquier duda o consulta, contactar vía correo electrónico a serflo.elec@gmail.com

B. Historias de Usuario

A continuación se detallan las Historias de Usuario rescatadas desde los requisitos planteados por el Usuario, los cuales pueden ser encontrados en Anexo A.

Historias de Usuario	Descripción
H1	Como usuario deseo convertir audios a texto.
H2	Como usuario deseo ampliar la capacidad de reconocimiento en un futuro.
H3	Como usuario deseo entrenar el sistema de forma masiva.
H4	Como usuario deseo recibir respuesta en lenguaje castellano.
H5	Como usuario deseo corregir palabras mal escritas.
H6	Como usuario deseo recibir siempre una respuesta, aunque esta no sea la mejor.
H7	Como usuario debo poder realizar cambios o mejoras en la implementación, por lo que es necesario una buena documentación.
H8	Como usuario debo poder incluir estas funcionalidades a software desarrollados por la empresa.

C. Pruebas de Corrección

A continuación se muestra un extracto de las Pruebas de Corrección de palabras basadas en diccionario. Cabe destacar que sólo se incluyen los casos que son relevantes.

Consulta: buei.


```
[
  {
    "target": "buei"
  },
  {
    "words": [
      "buey"
    ]
  }
]
real 0m0.494s
user 0m0.005s
sys 0m0.003s
```


Consulta: mani.


```
[
  {
    "target": "mani"
  },
  {
```

```
    "words": [
      "mana",
      "man\u00e1"
    ]
  }
]

real 0m0.488s
user 0m0.004s
sys 0m0.003s
```


Consulta: perroo.


```
[
  {
    "target": "perroo"
  },
  {
    "words": [
      "perro"
    ]
  }
]

real 0m0.558s
user 0m0.004s
sys 0m0.003s
```


Consulta: negocea.


```
[
  {
    "target": "negocea"
  },
  {
    "words": [
```

```
      "noceda",
      "necear",
      "goce",
      "negociar",
      "noca"
    ]
  }
]

real 0m3.254s
user 0m0.004s
sys 0m0.003s
```

Consulta: prespectiva.

```
[
  {
    "target": "prespectiva"
  },
  {
    "words": [
      "perspectiva"
    ]
  }
]

real 0m0.623s
user 0m0.004s
sys 0m0.003s
```

Consulta: gomito.

```
[
  {
    "target": "gomito"
  },
```

```
  {
    "words": [
      "mito",
      "v\u00f3mito"
    ]
  }
]

real 0m0.605s
user 0m0.004s
sys 0m0.003s
```


Consulta: tualla.


```
[
  {
    "target": "tualla"
  },
  {
    "words": [
      "talla"
    ]
  }
]

real 0m0.592s
user 0m0.004s
sys 0m0.003s
```