



UNIVERSIDAD
DE MÁLAGA



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
INFORMÁTICA
INGENIERÍA DE LA SALUD

**Desarrollo de una librería Python para la evaluación
de los resultados obtenidos con Kaldi en problemas de
Reconocimiento Automático del Habla (RAH)**

**Development of a Python library for the evaluation of
the results obtained with Kaldi in Automatic
Speech Recognition (ASR) problems**

Realizado por
David Gómez Gómez

Tutorizado por
Enrique Nava Baro
Ignacio Moreno-Torres Sánchez

Departamento
Departamento de Ingeniería de Comunicaciones

UNIVERSIDAD DE MÁLAGA
MÁLAGA, JUNIO DE 2020

Fecha defensa: julio de 2020

Resumen

En este Trabajo de Fin de Grado (TFG) se presenta el diseño de una librería Python con la que se podrán evaluar los resultados obtenidos con el software Kaldi en el Reconocimiento Automático del Habla (RAH). Para realizar este diseño se utilizarán los resultados obtenidos al ejecutar la receta ASCIA en el software Kaldi, la cual está pensada para detectar problemas de afasia en diversos sujetos mientras estos participan en una terapia del habla.

La librería permite identificar claramente los aciertos y errores que el software de RAH ha obtenido, así como evaluar el tipo de errores y el motivo de que éstos se hayan producido.

Palabras clave: RAH, Librería, Python, Estadística, Kaldi

Abstract

In this Final Degree Project, it is presented the design of a Python library with which the results obtained with the Kaldi software in Automatic Speech Recognition (ASR) can be evaluated. In order to carry out this design, the results obtained when executing the ASCIA recipe in the Kaldi software will be used. This recipe is designed to detect aphasia problems in different subjects while they participate in a speech therapy.

The library allows clearly identify the hits and errors that the ASR software has obtained, as well as evaluation of the type of errors and the reason for them.

Keywords: ASR, Library, Python, Statistics, Kaldi

Índice General

1. INTRODUCCIÓN	1
1.1 MOTIVACIÓN	1
1.2 DESCRIPCIÓN DEL PROBLEMA	2
1.3 ESTADO DEL ARTE	3
1.4 OBJETIVOS.....	3
1.5 TECNOLOGÍAS Y RECURSOS EMPLEADOS	4
1.5.1 <i>Python 3 y Anaconda Navigator</i>	4
1.5.2 <i>Script de Shell - Unix</i>	7
1.5.3 <i>Git: Sistema de control de versiones</i>	7
1.5.4 <i>Kaldi</i>	8
1.5.5 <i>Receta ASICA para Kaldi</i>	9
1.6 ESTRUCTURA DE LA MEMORIA	9
2. FUNDAMENTOS.....	11
2.1 AFASIA	11
2.2 TIPO DE CONSONANTES.....	12
2.2.1 <i>Sonoridad</i>	12
2.2.2 <i>Lugar de articulación</i>	12
2.2.3 <i>Modo de articulación</i>	13
2.3 MATRIZ DE CONFUSIÓN	13
2.4 SENSIBILIDAD	15
2.5 ESPECIFICIDAD	16
2.6 EXACTITUD.....	16
2.7 PRECISIÓN	16
2.8 ERROR MEDIO	18
2.9 VALOR-F	18
2.10 RATIO DE FALSOS POSITIVOS	19
2.11 COEFICIENTE DE CORRELACIÓN DE MATTHEWS	19
2.12 ÍNDICE DE JACCARD.....	20
2.13 PARÁMETRO D'	21
2.14 LIBRERÍA Y TIPOS	23
2.14.1 <i>Librería estática</i>	23

2.14.2 <i>Librería dinámica</i>	23
2.14.3 <i>Ventajas e inconveniente de cada tipo de librería</i>	24
3. DESARROLLO INICIAL.....	25
3.1 INSTALACIÓN DE KALDI	25
3.2 INSTALACIÓN RECETA ASICA.....	27
3.3 OBTENCIÓN DE DATOS PARA REALIZAR LA LIBRERÍA	30
4. DESARROLLO DE LA LIBRERÍA	33
4.1 CONVERSIÓN DE DATOS.....	33
4.2 FUNCIÓN DE CÁLCULO DEL PORCENTAJE DE SÍLABAS RECONOCIDAS CORRECTAMENTE	34
4.3 FUNCIÓN DE CÁLCULO DE MATRICES DE CONFUSIÓN	36
4.4 FUNCIÓN DE AGRUPACIÓN DE CONSONANTES	40
4.5 FUNCIÓN PARA GRAFICAR Y COLOREAR MATRICES NORMALIZADAS.....	46
4.6 FUNCIÓN PARA TRANSFORMAR LAS MATRICES DE CONFUSIÓN TOTALES EN SUBMATRICES DE TAMAÑO 2x2	52
4.7 FUNCIONES ESTADÍSTICAS	54
4.7.1 <i>Sensibilidad</i>	55
4.7.2 <i>Especificidad</i>	56
4.7.3 <i>Exactitud</i>	57
4.7.4 <i>Precisión</i>	57
4.7.5 <i>Error Medio</i>	58
4.7.6 <i>Valor-F</i>	58
4.7.7 <i>Ratio de Falsos Positivos</i>	59
4.7.8 <i>Coeficiente de correlación de Matthews</i>	59
4.7.9 <i>Índice de Jaccard</i>	60
4.7.10 <i>Parámetro D'</i>	60
4.8 CREACIÓN DE TABLA DE FUNCIONES ESTADÍSTICAS.....	61
4.9 CÁLCULO Y CREACIÓN DE UNA TABLA PARA LAS FUNCIONES ESTADÍSTICAS DE MANERA INDIVIDUAL EN UNA MATRIZ DE CONFUSIÓN DE LETRAS.....	67
4.10 FUNCIÓN PARA GUARDAR DATAFRAME EN FORMATO .XLSX	70
4.11 SUBIR LIBRERÍA A GITHUB	71
5. PRUEBA DE LAS FUNCIONES.....	73

5.1	PRUEBA DEL PORCENTAJE DE SÍLABAS RECONOCIDAS CORRECTAMENTE.....	73
5.2	PRUEBA DE MATRICES DE CONFUSIÓN	74
5.3	PRUEBA DE AGRUPACIÓN DE CONSONANTES.....	75
5.4	PRUEBA GRAFICAR Y COLOREAR MATRICES NORMALIZADAS.....	77
5.5	PRUEBA PARA DE LA FUNCIÓN DE TRANSFORMACIÓN DE MATRIZ DE CONFUSIÓN MxM EN 2x2.....	78
5.6	PRUEBA DEL RESULTADO DE LA TABLA DE FUNCIONES ESTADÍSTICAS.....	79
5.7	PRUEBA DEL CÁLCULO Y CREACIÓN DE UNA TABLA CON FUNCIONES ESTADÍSTICAS DE MANERA INDIVIDUAL DE UNA MATRIZ DE CONFUSIÓN DE LETRAS....	80
6.	CONCLUSIONES Y LÍNEAS FUTURAS	85
7.	BIBLIOGRAFÍA	87
I.	GLOSARIO.....	93
II.	ANEXOS	95
II.I	ANEXO I: INSTALACIÓN DE KALDI Y DE RECETA ASICA	95
II.II	ANEXO II: ELECCIÓN DE GRADACIÓN DE COLOR PARA GRAFICAR MATRICES	96
II.III	ANEXO III: COMPROBACIÓN DE RESULTADOS DEL PARÁMETRO D'	99

Índice de Figuras

FIGURA 1: ENTORNO DE ANACONDA NAVIGATOR	6
FIGURA 2: SPYDER - ENTORNO DE DESARROLLO PARA PYTHON	6
FIGURA 3: SCRIPT SHELL MACOS	7
FIGURA 4: MATRIZ DE CONFUSIÓN MxM (FILAS X COLUMNAS)	14
FIGURA 5: MATRIZ DE CONFUSIÓN 2x2 (FILAS X COLUMNAS)	15
FIGURA 6: REPRESENTACIÓN GRÁFICA DE LA EXACTITUD Y LA PRECISIÓN	17
FIGURA 7: REPRESENTACIÓN GRÁFICA DE LOS POSIBLES VALORES DE LA EXACTITUD Y LA PRECISIÓN	18
FIGURA 8: INTERSECCIÓN DE LOS CONJUNTOS A Y B	20
FIGURA 9: UNIÓN DE LOS CONJUNTOS A Y B	20
FIGURA 10: REPRESENTACIÓN PARÁMETRO D' EN DISTRIBUCIÓN NORMAL	21
FIGURA 11: INSTALACIÓN HOMEBREW EN MACOS	26
FIGURA 12: PRUEBA 1 - FUNCIONAMIENTO ASICA.PY	29
FIGURA 13: PRUEBA 2.1 ASICA - FUNCIONAMIENTO CROSSVAL_SPK.SH	29
FIGURA 14: PRUEBA 2.2 ASICA - FUNCIONAMIENTO CROSSVAL_SPK.SH	30
FIGURA 15: EJEMPLO DE ARCHIVO DE TEXTO "SALIDA.TXT"	31
FIGURA 16: DIAGRAMA DE FLUJO DE LA FUNCIÓN "PORCENTAJEACIERTOS (DATAFRAME)"	36
FIGURA 17: DIAGRAMA DE FLUJO DE LA FUNCIÓN "CONFUSIONMATRIXVOC (DATAFRAME)"	39
FIGURA 18: DIAGRAMA DE FLUJO DE LA FUNCIÓN "NORMALCONFUSIONMATRIXVOC (DATAFRAME)"	40
FIGURA 19: DIAGRAMA DE FLUJO DE LA FUNCIÓN "CSONORA(DF)"	46
FIGURA 20: DIAGRAMA DE FLUJO DE LA FUNCIÓN "PLOTMATORANGEBLUE(NORMALIZE_CONFUSION_MATRIX, NOMBRE)"	51
FIGURA 21: DIAGRAMA DE FLUJO DE LA FUNCIÓN "MATRIZ2X2(CONFUSION_MATRIX)"	54
FIGURA 22: DIAGRAMA DE FLUJO DE LA FUNCIÓN "SENSIBILIDAD(VP, FN, FP, VN)".....	56
FIGURA 23: DIAGRAMA DE FLUJOS DE LA FUNCIÓN "FUNCIONESTADISTICAS(CONFUSION_MATRIX)"	66
FIGURA 24: DIAGRAMA DE FLUJO DE LA FUNCIÓN "ESTATSENSIBILIDAD(MATRIZ)".....	69
FIGURA 25: DIAGRAMA DE FLUJO DE LA FUNCIÓN "DFTOEXCEL(DATAFRAME, NOMBRE)"	71

FIGURA 26: RESULTADO DEL PORCENTAJE DE SÍLABAS RECONOCIDAS CORRECTAMENTE PARA "SALIDA.TXT"	73
FIGURA 27: RESULTADO MATRIZ DE CONFUSIÓN DE CONSONANTES PARA "SALIDA.TXT"	74
FIGURA 28: RESULTADO MATRIZ DE CONFUSIÓN NORMALIZADA DE CONSONANTES PARA "SALIDA.TXT"	75
FIGURA 29: RESULTADO MATRIZ DE CONFUSIÓN DE CONSONANTES SONORAS PARA "SALIDA.TXT"	76
FIGURA 30: RESULTADO MATRIZ DE CONFUSIÓN NORMALIZADA DE CONSONANTES SONORAS PARA "SALIDA.TXT"	76
FIGURA 31: MATRIZ DE CONFUSIÓN COLOREADA EN NARANJA-AZUL PARA LAS CONSONANTES	77
FIGURA 32: MATRIZ DE CONFUSIÓN COLOREADA EN VIRIDIS PARA LAS CONSONANTES ..	78
FIGURA 33: MATRIZ DE CONFUSIÓN VOCALES	79
FIGURA 34: MATRICES DE CONFUSIÓN 2x2 PARA VOCALES	79
FIGURA 35: TABLA QUE RECOGE TODAS LAS FUNCIONES ESTADÍSTICAS PARA LA MATRIZ DE CONFUSIÓN DE LAS VOCALES DEL ARCHIVO "SALIDA.TXT"	80
FIGURA 36: SENSIBILIDAD DE LAS VOCALES PARA EL ARCHIVO "SALIDA.TXT"	81
FIGURA 37: ESPECIFICIDAD DE LAS VOCALES PARA EL ARCHIVO "SALIDA.TXT"	81
FIGURA 38: EXACTITUD DE LAS VOCALES PARA EL ARCHIVO "SALIDA.TXT"	81
FIGURA 39: PRECISIÓN DE LAS VOCALES PARA EL ARCHIVO "SALIDA.TXT"	82
FIGURA 40: ERROR MEDIO DE LAS VOCALES PARA EL ARCHIVO "SALIDA.TXT"	82
FIGURA 41: VALOR-F DE LAS VOCALES PARA EL ARCHIVO "SALIDA.TXT"	83
FIGURA 42: RATIO DE FP DE LAS VOCALES PARA EL ARCHIVO "SALIDA.TXT"	83
FIGURA 43: COEFICIENTE DE MATTHEWS DE LAS VOCALES PARA EL ARCHIVO "SALIDA.TXT"	83
FIGURA 44: ÍNDICE DE JACCARD DE LAS VOCALES PARA EL ARCHIVO "SALIDA.TXT"	84
FIGURA 45: PARÁMETRO D' DE LAS VOCALES PARA EL ARCHIVO "SALIDA.TXT"	84
FIGURA 46: MATRIZ DE CONFUSIÓN COLOREADA EN UNA GRADACIÓN AZUL DE MANERA LINEAL	96
FIGURA 47: MATRIZ DE CONFUSIÓN COLOREADA EN UNA GRADACIÓN AZUL EN ESCALA LOGARÍTMICA	97
FIGURA 48: MATRIZ DE CONFUSIÓN COLOREADA EN UNA GRADACIÓN NARANJA-AZUL-ROSA EN ESCALA LOGARÍTMICA	98

FIGURA 49: FUNCIÓN NORMAL DE DISTRIBUCIÓN INVERSA PARA UN MILLÓN DE VALORES ENTRE 0 Y 1	100
--	-----

X

Índice de tablas

TABLA 1: TRANSFORMACIÓN DE "SALIDA.TXT" A UN DATAFRAME EN PYTHON.....	33
TABLA 2: FUNCIÓN DE CÁLCULO DEL PORCENTAJE DE ACIERTO PYTHON	34
TABLA 3: FUNCIONES DE CREACIÓN DE MATRICES DE CONFUSIÓN PARA VOCALES Y CONSONANTES	37
TABLA 4: FUNCIONES DE CREACIÓN DE MATRICES DE CONFUSIÓN PARA LAS DISTINTAS AGRUPACIONES DE CONSONANTES.....	41
TABLA 5: FUNCIONES PARA GRAFICAR Y COLOREAR MATRICES DE CONFUSIÓN	48
TABLA 6: FUNCIÓN PARA TRANSFORMAR MATRICES DE CONFUSIÓN GRANDES EN SUBMATRICES 2x2	52
TABLA 7: FUNCIÓN PARA REALIZAR EL CÁLCULO DE LA SENSIBILIDAD.....	55
TABLA 8: FUNCIÓN PARA REALIZAR EL CÁLCULO DE LA ESPECIFICIDAD	56
TABLA 9: FUNCIÓN PARA REALIZAR EL CÁLCULO DE LA EXACTITUD	57
TABLA 10: FUNCIÓN PARA REALIZAR EL CÁLCULO DE LA PRECISIÓN	57
TABLA 11: FUNCIÓN PARA REALIZAR EL CÁLCULO DEL ERROR MEDIO	58
TABLA 12: FUNCIÓN PARA REALIZAR EL CÁLCULO DEL VALOR-F	58
TABLA 13: FUNCIÓN PARA REALIZAR EL CÁLCULO DEL RATIO DE FALSOS POSITIVOS	59
TABLA 14: FUNCIÓN PARA REALIZAR EL CÁLCULO DEL COEFICIENTE DE CORRELACIÓN DE MATTHEWS	59
TABLA 15: FUNCIÓN PARA REALIZAR EL CÁLCULO DEL ÍNDICE DE JACCARD	60
TABLA 16: FUNCIÓN PARA REALIZAR EL CÁLCULO DEL PARÁMETRO D'	61
TABLA 17: FUNCIÓN PARA IMPRIMIR UNA TABLA CON TODOS LOS PARÁMETROS ESTADÍSTICOS CALCULADOS PARA CADA LETRA DE UNA MATRIZ DE CONFUSIÓN DE ENTRADA	61
TABLA 18: FUNCIONES PARA EL CÁLCULO DE PARÁMETROS ESTADÍSTICOS CONCRETOS PARA UNA MATRIZ DE CONFUSIÓN DE LETRAS	67
TABLA 19: FUNCIÓN PARA GUARDAR LOS DATAFRAMES OBTENIDOS EN FORMATO .XLSX	70
TABLA 20: INSTALACIÓN KALDI EN MACOS	95
TABLA 21: INSTALACIÓN RECETA ASICA	95
TABLA 22: COMPROBACIÓN DE LA FUNCIÓN NORMAL DE DISTRIBUCIÓN INVERSA	99

Índice de ecuaciones

ECUACIÓN 1: CÁLCULO DE LA SENSIBILIDAD PARA MATRICES DE CONFUSIÓN	15
ECUACIÓN 2: CÁLCULO DE LA ESPECIFICIDAD PARA MATRICES DE CONFUSIÓN	16
ECUACIÓN 3: CÁLCULO DE LA EXACTITUD PARA MATRICES DE CONFUSIÓN.....	16
ECUACIÓN 4: CÁLCULO DE LA PRECISIÓN PARA MATRICES DE CONFUSIÓN	17
ECUACIÓN 5: CÁLCULO DEL ERROR MEDIO PARA MATRICES DE CONFUSIÓN	18
ECUACIÓN 6: CÁLCULO DEL VALOR-F PARA MATRICES DE CONFUSIÓN	19
ECUACIÓN 7: CÁLCULO DEL RATIO DE FALSOS POSITIVOS PARA MATRICES DE CONFUSIÓN	19
ECUACIÓN 8: CÁLCULO DEL COEFICIENTE DE CORRELACIÓN DE MATTHEWS PARA MATRICES DE CONFUSIÓN	19
ECUACIÓN 9: CÁLCULO GENERAL DEL ÍNDICE DE JACCARD	20
ECUACIÓN 10: CÁLCULO DEL ÍNDICE DE JACCARD PARA MATRICES DE CONFUSIÓN.....	21
ECUACIÓN 11: CÁLCULO DEL PARÁMETRO D' PARA MATRICES DE CONFUSIÓN	22
ECUACIÓN 12: FUNCIÓN NORMAL DE DISTRIBUCIÓN INVERSA	22
ECUACIÓN 13: FUNCIÓN DE ERROR COMPLEMENTARIA	22
ECUACIÓN 14: CÁLCULO DEL PARÁMETRO D' PARA MATRICES DE CONFUSIÓN	99

1. Introducción

1.1 Motivación

Actualmente, uno de los pilares en el desarrollo de nuevas tecnologías es conseguir una interacción lo más natural y eficiente posible entre éstas y sus usuarios. El desarrollo de interfaces hombre-máquina a través de la voz ha sido un campo de gran interés en las ciencias de la computación, puesto que el habla es una de las formas de comunicación más naturales y eficientes que utiliza el ser humano.

El Reconocimiento Automático del Habla (RAH) es un subcampo interdisciplinario que combina la informática y la lingüística computacional, tiene como objetivo permitir la comunicación hablada entre seres humanos y computadoras. Para ello, se desarrollan metodologías y tecnologías con las que se permite el reconocimiento del lenguaje verbal (una onda acústica) por parte de una computadora y su posterior transcripción textual.

El RAH tiene una gran complejidad, esto se puede atribuir principalmente a dos factores, en primer lugar, existe una gran variabilidad en la señal de habla emitida por el locutor, puesto que depende de diversas circunstancias particulares como son la velocidad, estilo o medio donde se realiza la locución; en segundo lugar, debido a la dificultad de encontrar palabras individuales en un entorno acústico donde haya diversos factores que puedan distorsionarlas y, realizar al mismo tiempo, la tarea de segmentación y clasificación de estas (Evin, 2011). La información acústica que proporciona la señal del habla no es utilizada por completo en los sistemas de RAH, lo que estos hacen es interpretar la señal como secuencias de unidades cuyas duraciones se encuentran a nivel de segmentos fonéticos. Para que se pueda realizar una correcta interpretación de estas locuciones, es necesario realizar un entrenamiento previo del sistema de RAH; para un funcionamiento adecuado, es de vital importancia que el entrenamiento sea correcto, consiguiendo así un alto ratio de palabras interpretadas de manera correcta.

En los últimos años, se han dado grandes pasos en los avances de estas tecnologías, sin embargo, aún se está lejos de lograr un sistema que permita comprender cualquier mensaje y realizar una contestación tal y como lo haría una persona. Aun así, la tecnología actual permite realizar sistemas de reconocimiento automático del habla que tengan un error aceptable para un entorno semántico determinado.

1.2 Descripción del problema

La unidad fundamental del sonido hablado es el fonema, conociéndose su realización física como fono. Las palabras están constituidas por uno o más fonemas en secuencia, los cuales, pueden ser grabados como una señal acústica continua al ser emitidos por una persona. El objetivo de los programas de RAH en general y de Kaldi en particular (el programa que se usará en este proyecto), es inferir las palabras originales pronunciadas por el emisor a partir de la señal acústica recibida; para lograr esto, se utiliza un enfoque probabilístico, en el que la señal hablada corresponde con una cierta probabilidad a una secuencia de palabras (Sobrino, 2018).

Kaldi es un kit de herramientas de reconocimiento de voz y procesamiento de señales de código abierto, escrita principalmente en C++ y centrada en la investigación del RAH. Está disponible de manera gratuita bajo la licencia Apache v2.0, siendo de esta manera muy poco restrictivo y adecuado para una amplia comunidad de usuarios (Povey, 2011).

Este software tiene una gran cantidad de utilidades, además del reconocimiento automático del habla en diferentes idiomas de personas sin ninguna anomalía en el habla, una de las líneas de investigación posible es el reconocimiento de diversas patologías a través del habla de esta persona.

Para que Kaldi realice el RAH de manera correcta, es necesario un entrenamiento previo mediante una base de datos con grabaciones de un locutor en donde cada fonema se ha identificado previamente mediante una anotación que refleja la traducción textual de su señal acústica correspondiente. Además, se deben implementar distintas recetas de código que cuentan con diversos parámetros con los que realizar un reconocimiento adecuado según el idioma y necesidades del locutor. Una vez realizados estos procesos,

en el programa de RAH se pueden producir diversos errores en función de lo adecuado que haya sido el entrenamiento y la receta utilizada. Para la detección y corrección de estos errores, una de las herramientas utilizadas es la obtención de diversos parámetros estadísticos con los que analizar y verificar la precisión de los resultados dados por del sistema, así como identificar dónde y por qué se producen los posibles errores.

En la actualidad, los programas de RAH también se están utilizando para ayudar a pacientes con trastornos en habla como la afasia (Le D. L., 2018).

1.3 Estado del arte

Desde la aparición del primer prototipo para el RAH a mediados del siglo XX, esta tecnología ha sufrido grandes avances hasta el día de hoy, existiendo un gran abanico de aplicaciones con diversas funcionalidades.

Ámbitos donde esta tecnología está ampliamente extendida son la atención telefónica (servicios de información, concertar citas o notificación de averías); aplicaciones de mando y control, como el control de dispositivos de manera remota para aumentar así su seguridad, por ejemplo, en el control de la radio, climatizador o GPS dentro de un vehículo o la comodidad, como por ejemplo la búsqueda de información en motores de búsqueda como Google, control de aplicaciones en domótica o sistemas de dictado que permiten redactar documentos.

Estos sistemas también se están empleando en el campo de la medicina con usos como la transcripción de notas o informes médicos; en los últimos años, también se están desarrollando sistemas de RAH para la detección de problemas en el habla como la parafasia (Le D. L., 2017), deterioros cognitivos (Tóth, 2015) o análisis cuantitativos de la afasia espontánea (Le D. L., 2018).

1.4 Objetivos

El propósito de este trabajo de fin de grado (TFG) es el desarrollo de una librería estadística en Python para la evaluación de resultados en el RAH.

Para ello, se implementarán una serie de pruebas estadísticas con las que el usuario que utilice un sistema de RAH con el software Kaldi, podrá identificar de manera clara la exactitud de los resultados obtenidos y los posibles errores. Se proporcionarán tanto resultados numéricos como gráficos para una mayor facilidad de identificación de los resultados del RAH por parte del usuario. Además, el usuario podrá implementar en su código la función que necesite para obtener los parámetros que considere oportunos.

Para poder llevar a cabo este trabajo, se utilizará la receta ASICA desarrollada previamente por el profesor D. Ignacio Moreno-Torres Sánchez, así como unos archivos de entrenamiento proporcionados en ella (Moreno-Torres Sánchez, 2019). ASICA es un proyecto de investigación para adaptar el RAH de Kaldi a sujetos con afasia, se espera que con este proyecto el sistema sea capaz de proporcionar retroalimentación a los sujetos con este problema.

Se ejecutará el software de RAH con la receta ASICA, proporcionándole una grabación controlada de un paciente, con lo que obtendremos unos resultados en los que se diferencian diversas variables a partir de las cuales se realizará la librería estadística para la evaluación y comprobación de los resultados obtenidos con la receta. De esta manera, se podrá valorar si el sistema está funcionando de manera correcta y posibles soluciones para mejorarlo. Esta librería de evaluación de resultados funcionará tanto para los usuarios que utilicen la receta ASICA como para usuarios que utilicen otras recetas que devuelvan los resultados similares.

1.5 Tecnologías y recursos empleados

Para el desarrollo de este TFG, se han utilizado las tecnologías y recursos descritos en los siguientes apartados.

1.5.1 Python 3 y Anaconda Navigator

El origen del lenguaje Python data en el año 1991. En este tiempo, Guido van Rossum que trabajaba en CWI (Centrum Wiskunde & Informatica) es asignado para realizar el desarrollo de un sistema operativo distribuido llamado Amoeba (Python,

2020). En CWI se utilizaba el lenguaje de programación ABC, ante las limitaciones y problemas de este lenguaje, Guido decide crear un nuevo lenguaje que mejore y corrija estos aspectos, de esta manera se da lugar al nacimiento de Python.

A lo largo de los años, el lenguaje Python ha ido actualizando sus versiones y, en estos momentos (junio de 2020), la última versión disponible (versión 3) cuenta con la actualización 3.8. Actualmente, se trata de un lenguaje multiparadigma potente, flexible y con una sintaxis clara y concisa; además, no requiere dedicar tiempo a su compilación puesto que es un lenguaje interpretado (Montoro, 2012). Posee un código de licencia abierto compatible con la Licencia pública general de GNU (Python D. , 2020).

Por defecto, en el sistema operativo utilizado (Mac OS X) viene instalado Python 2.7. Para instalar Python 3 es necesario en primer lugar instalar GCC (el compilador GNU de C), posteriormente y mediante el gestor de paquetes Homebrew, se instalará la versión actual deseada de Python (Reitz, 2016).

El entorno de desarrollo empleado ha sido Anaconda Navigator (*Figura 1*) distribuida por Anaconda. Es una interfaz gráfica de usuario (GUI) de escritorio que permite lanzar aplicaciones, administrar fácilmente paquetes de conda y entornos. Puede buscar paquetes tanto en Anaconda Cloud como en un repositorio local de Anaconda y está disponible para Windows, Linux y macOS (Home, 2020). Dentro de este entorno encontramos Spyder, un entorno científico escrito en Python y pensado para el desarrollo de este lenguaje en específico, pudiendo combinar funciones de edición, análisis, ejecución interactiva, inspección profunda y capacidad para la visualización de diversas figuras (*Figura 2*) (Contributors, 2018).

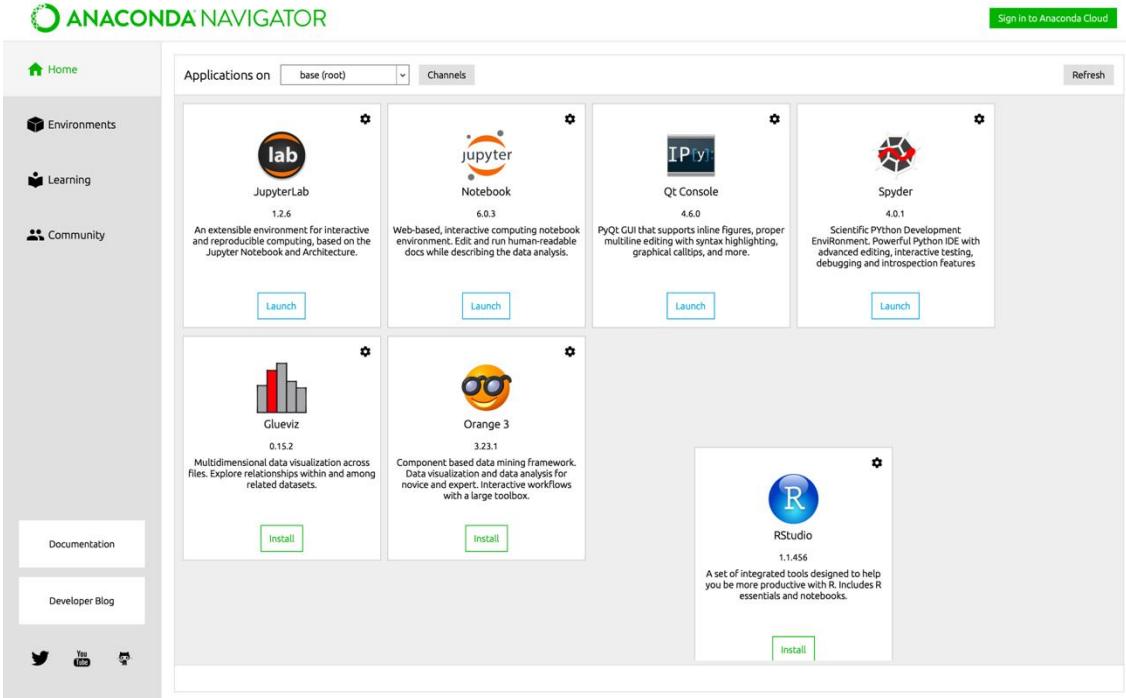


Figura 1: Entorno de Anaconda Navigator

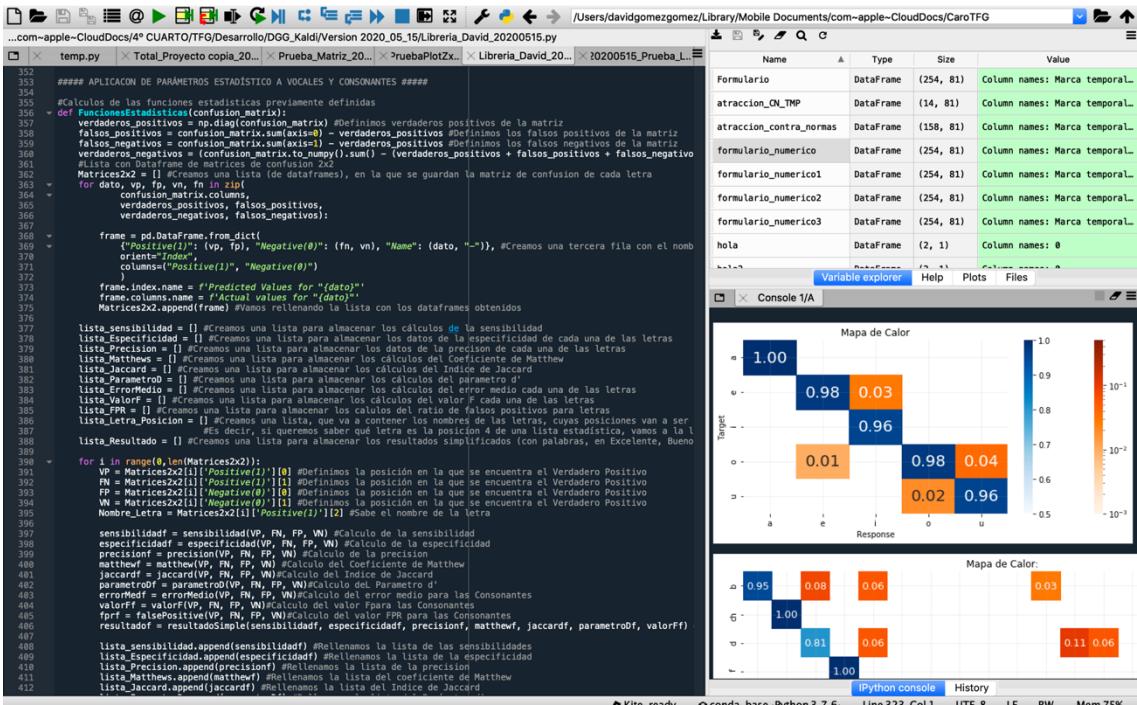


Figura 2: Spyder - Entorno de desarrollo para Python

1.5.2 Script de Shell - Unix

Un script de Shell es un archivo de texto que contiene uno o más comandos UNIX, estos sirven para introducir, ejecutar y esperar una respuesta de los comandos sin necesidad de una interacción directa que, de otro modo, se introducirían en la línea de comandos (*Figura 3*).

Son útiles porque se pueden combinar muchas tareas comunes en un único script, ahorrando tiempo y posibles errores de ejecución. Para indicar que el archivo de texto es ejecutable, se necesita la herramienta chmod (Inc., 2020).

A screenshot of a macOS terminal window. The title bar reads "davidgomezgomez — zsh — 71x20". The window contains the text: "Last login: Tue Jun 16 03:42:58 on ttys000" and "(base) davidgomezgomez@MacBook-Pro-de-David ~ %". The window has a standard OS X look with red, yellow, and green close buttons.

Figura 3: Script Shell macOS

1.5.3 Git: Sistema de control de versiones

Creado por Linus Torvalds en el año 2005, Git es un sistema de control de versiones pensado en la eficiencia y confiabilidad del mantenimiento de versiones de aplicaciones que pueden tener un gran número de archivos de código fuente. Está pensado para coordinar el trabajo entre programadores, aunque también se puede utilizar para rastrear los cambios en cualquier conjunto de archivos (Torvalds, 2005). Cada directorio de Git en cada ordenador es un repositorio completo con un historial completo y con capacidad

de seguimiento de versiones, independiente del acceso a la red o de un servidor central (Chacon, 2014). Git es un software libre distribuible bajo los términos de la versión 2 de la Licencia Pública General de GNU.

En este TFG se utilizará GitHub, una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git. Es una de las plataformas más utilizadas por esta razón, se ha decidido emplearla en este proyecto.

1.5.4 Kaldi

Kaldi es un kit de herramientas de código abierto para el reconocimiento de voz escrito en C++, disponible de manera gratuita bajo la licencia Apache v2.0. El objetivo de Kaldi es tener un código moderno y flexible que sea fácil de entender, modificar y ampliar. Las herramientas para su uso se compilan normalmente en sistemas Unix o en Microsoft Windows (Povey, 2011).

Pensado para investigadores del campo del RAH, cuenta con varios kits de herramientas libres con los que poder construir un sistema de reconocimiento. Se pueden destacar los kits HTK y Julius escritos en C, el kit Sphnx-4 escrito en Java y el kit RWTH ASR escrito en C++. Algunas de las características más importantes de Kaldi son:

- Integración con transductores de estado finito: Compilación mediante el kit de herramientas OpenFst, usándose como librería.
- Amplio soporte en álgebra lineal: Incluyendo librerías de matrices con los estándares de rutina BLAS y LAPACK.
- Diseño extensible: Los algoritmos se proporcionan de la forma más genérica posible.
- Licencia abierta: Trabaja bajo la licencia Apache v2.0, una de las licencias menos restrictivas que existen.
- Recetas completas: Cuenta con recetas completas para el RAH que funcionan con bases de datos como las que proporciona Linguistic Data Consortium (LDC).
- Pruebas exhaustivas: Tiene el objetivo de tener todas las rutinas de prueba correspondientes para los códigos utilizados.

El principal uso de Kaldi es la investigación en el modelado acústico, por lo que compite con kits de herramientas de reconocimiento de voz como HTK y RWTH ASR (RASR). La principal ventaja frente a estos otros kits es que Kaldi cuenta con un código más moderno, flexible, estructurado y mejor soporte, además de tener unos términos de licencia más abiertos (Povey, 2011).

1.5.5 Receta ASICA para Kaldi

ASICA es un proyecto de investigación que se encuentra actualmente en curso y, tiene como objetivo, adaptar el RAH de Kaldi para reconocer el habla producida en sujetos con afasia mientras participan en una tarea de terapia del habla como, por ejemplo, la repetición no verbal. A largo plazo, se espera que el sistema sea capaz de dar retroalimentación a los pacientes.

Para hacer frente a los errores fonéticos observados típicamente en sujetos con afasia, en ASICA se utiliza la siguiente estrategia: en lugar de entrenar al sistema para que aprenda palabras reales en español (enfoque estándar en el RAH), se entrena al sistema para que reconozca un número relativamente grande de sílabas (Moreno-Torres Sánchez, 2019).

1.6 Estructura de la memoria

En el primer capítulo, se realiza una introducción a este TFG, explicándose su motivación, descripción del problema, estado de arte, objetivos y las tecnologías y recursos empleados. En el segundo capítulo, se recogerán los fundamentos teóricos en los que está basado. En el tercer y cuarto capítulo, se describirán las fases del desarrollo del trabajo y las pruebas realizadas en ellas respectivamente. Por último, se describen las conclusiones y posibles líneas futuras con las que seguir desarrollando este trabajo.

2. Fundamentos

En este apartado, se encuentran los fundamentos teóricos necesarios para entender el desarrollo de este trabajo.

2.1 Afasia

La afasia es un trastorno del lenguaje adquirido a consecuencia de un daño cerebral. En general, este trastorno afecta a todas las modalidades del lenguaje: expresión y comprensión del lenguaje oral, escritura y comprensión lectora (Rafael González V., 2014). Cada una de estas variables puede afectar tanto cuantitativa como cualitativamente de distintas formas, por lo que grupos sindromáticos pueden coexistir con deficiencias en el procesamiento cognitivo. El síntoma predominante suele ser la anomia (dificultad para evocar palabras) (Nancy Helm-Estabrooks, 2005). También es habitual que las personas afásicas presenten dificultad en el lenguaje lecto-escrito, trastornos denominados alexia y agraphia respectivamente.

Estimaciones datan la incidencia de la afasia en 80.000 personas cada año, con prevalencia de un millón en Estados Unidos. Se puede producir por una de las siguientes causas: accidente cerebro vascular (ACV), traumatismo encefalocraneano (TEC), tumor (TU), infecciones o enfermedades neurodegenerativas (Rafael González V., 2014).

La evaluación tiene los propósitos de encontrar los procesos del lenguaje que están comprometidos, determinar la severidad del trastorno, precisar el tipo de afasia (fluente o no fluente) y planificar la rehabilitación del lenguaje. Esta evaluación puede ser formal, a través de la aplicación de pruebas estandarizadas, o informal, mediante la evaluación clínica (Rafael González V., 2014).

2.2 Tipo de consonantes

Las letras pueden clasificarse en diferentes subgrupos según características como el tipo de sonido o su manera de articulación. Se encuentran tres grandes grupos principales de clasificación para las consonantes: según la sonoridad, según el lugar de articulación y según el modo de articulación.

2.2.1 Sonoridad

Se pueden diferenciar dos subgrupos en función del uso de las cuerdas vocales para pronunciar las consonantes (Greelane, 2019):

- Sonoras: Requieren el uso de las cuerdas vocales para producir sonidos, las cuales modulan el flujo de aire expulsado de los pulmones. Se encuentran "b", "d", "g", "l", "m", "n", "ny", "r", "rr", "y".
- Sordas: No requiere el uso de las cuerdas vocales para producir sonidos, el aire fluye libremente de los pulmones a la boca, donde la lengua, dientes y labios se acoplan para modular el sonido. Se encuentran "ch", "f", "k", "p", "s", "t", "x", "z".

2.2.2 Lugar de articulación

Se pueden diferenciar tres subgrupos en función del lugar de articulación utilizado para la pronunciación de la consonante (DiCanio, 2013):

- Frontal: Estas consonantes reciben su nombre debido a que se articulan con la parte frontal de la boca. Se encuentran "b", "f", "m", "p".
- Coronal: Es aquella consonante que es articulada con la parte coronal de la lengua. Se encuentran "ch", "d", "l", "n", "ny", "r", "rr", "s", "t", "y", "z".
- Back: Reciben este nombre debido a que estas consonantes se articulan en la parte posterior de la boca. Se encuentran "g", "k", "x".

2.2.3 Modo de articulación

El modo de articulación describe la manera en la que sale el aire de la boca según las modificaciones que causan los articuladores (lengua, dientes, labios, etc.) para producir los diferentes fonemas. Se pueden diferenciar 5 subgrupos en función del modo de articulación de la consonante (Ganesan, 2019):

- **Oclusiva:** En un principio, se produce una oclusión total que bloquea la salida de aire posteriormente, el aire acumulado se escapa rápidamente. Se encuentran "b", "d", "g", "k", "p", "t".
- **Africada:** Aunque el aire está bloqueado, se producen varias veces salidas pequeñas mediante una pequeña apertura. Se encuentran "ch".
- **Fricativa:** Se caracteriza por el escape del aire mediante una pequeña apertura con la que se crea una fricción audible. Se encuentran "f", "s", "x", "y", "z".
- **Nasal:** Se forman por el escape de aire por la cavidad nasal, creando una resonancia o nasalidad. Se encuentran "m", "n", "ny".
- **Aproximante:** Se articulan mediante la aproximación de dos órganos de articulación, aunque sin que se interrumpa totalmente la corriente de aire ni se produzca una estrechamiento. Se encuentran: "l", "r", "rr".

2.3 Matriz de confusión

Una matriz de confusión (también llamada matriz de error), es una tabla de contingencia utilizada como herramienta estadística para el análisis de observaciones emparejadas. Actualmente, la matriz de confusión ha sido adoptada como estándar para informar sobre la exactitud técnica de cualquier producto de datos derivados de la teledetección (Comber, 2012). Esta misma herramienta puede ser también utilizada para evaluar la calidad de cualquier tipo de dato espacial; por ejemplo, la matriz de confusión aparece reconocida en la Norma Internacional ISO 19157, refiriéndose a la calidad de la información geográfica, como un mecanismo para ofrecer resultados de calidad de productos vectoriales o derivados de imágenes, como por ejemplo, la clasificación de una imagen (Ariza-López, 2018).

La matriz de confusión comprende un conjunto de valores que contabilizan el grado de semejanza entre observaciones emparejadas: un conjunto de datos bajo control (CDC) y un conjunto de datos de referencia (CDR) para los que se establece una clasificación. En cualquier tipología, en esta matriz los elementos CDC se comparan con sus homólogos CDR (Ariza-López, 2018).

Se visualiza como una matriz cuadrada de dimensión MxM (filas x columnas), donde M denota el número de clases que se consideran. Las clases de referencia se denominan Γ_R y las clases bajo control G_P (*Figura 4*). Las celdas de la diagonal de la matriz contienen las cantidades correspondientes a los datos bien clasificados, Γ_R y G_P coinciden, y son denominadas celdas de coincidencia (C_C); las celdas que se encuentran fuera de la diagonal principal son las cantidades correspondientes a las confusiones (errores de clasificación de datos), se denominan celdas de error (C_E). Una matriz de confusión proporciona una visión completa de los aciertos y confusiones en los datos analizados, pero es difícil de manejar de una manera sencilla (Ariza-López, 2018).

		Referencia			
		Γ_1	Γ_2	...	Γ_M
Conjunto de datos	G_1	n_{11}	n_{12}	...	n_{1M}
	G_2	n_{21}	n_{11}	...	n_{2M}
	:	:	:	:	:
	G_M	n_{M1}	n_{11}	...	n_{MM}
	Total	n_{+1}	n_{+2}		n_{+M}

Figura 4: Matriz de confusión MxM (filas x columnas)

Para analizar y visualizar los datos de la matriz de confusión de manera más sencilla y aplicar diferentes parámetros estadísticos, se puede transformar la matriz de confusión de dimensión MxM en una matriz de confusión 2x2 para cada uno de sus valores, o hacer

la suma total de todos ellos para de igual forma, convertir la matriz a una dimensión 2x2 (*Figura 5*).

		Valores observados	
		Positivos (1)	Negativos (0)
Valores Predichos	Positivos (1)	Verdaderos positivos (VP)	Falsos Positivos (FP)
	Negativos (0)	Falsos Negativos (FN)	Verdaderos Negativos (VN)

Figura 5: Matriz de confusión 2x2 (filas x columnas)

- Verdaderos Positivos (VP): Cantidad de positivos que fueron clasificados correctamente como positivos por el modelo.
- Verdaderos Negativos (VN): Cantidad de negativos que fueron clasificados correctamente como negativos por el modelo.
- Falsos Negativos (FN): Cantidad de positivos que fueron clasificados incorrectamente como negativos por el modelo.
- Falsos Positivos (FP): Cantidad de negativos que fueron clasificados incorrectamente como positivos por el modelo.

2.4 Sensibilidad

También conocida como Tasa de Verdaderos Positivos (“Recall” o “Sensitivity”), la sensibilidad indica la capacidad de nuestro sistema para dar como casos positivos los casos que realmente son positivos (capacidad de predecir verdaderos positivos) es decir, la proporción de casos positivos que son identificados de manera correcta por el algoritmo (Pita Fernández, 2003). Se calcula como (*Ecuación 1*):

$$\text{Sensibilidad} = \frac{VP}{VP + FN}$$

Ecuación 1: Cálculo de la Sensibilidad para matrices de confusión

2.5 Especificidad

También conocida como Tasa de Verdaderos Negativos (“Especificity”), la especificidad indica la capacidad de nuestro sistema para dar como casos negativos los casos que realmente son negativos (capacidad de predecir verdaderos negativos) es decir, la proporción de casos negativos identificados de manera correcta por el algoritmo (Pita Fernández, 2003). Se calcula como (*Ecuación 2*):

$$\text{Especificidad} = \frac{VN}{VN + FP}$$

Ecuación 2: Cálculo de la Especificidad para matrices de confusión

2.6 Exactitud

La exactitud (“Accuracy”) se refiere a cuán cerca del valor real se encuentra el valor medido. Se define como la proximidad existente entre un valor medido y un valor verdadero de un mesurando. Por lo tanto, una medición es más exacta cuanto más pequeño sea el error de medida, se suele decir que una medición que ofrece una menor incertidumbre es una medición más exacta (Armenteros, 2010). Se calcula como (*Ecuación 3*):

$$\text{Exactitud} = \frac{VP + VN}{VP + FP + FN + VN}$$

Ecuación 3: Cálculo de la Exactitud para matrices de confusión

2.7 Precisión

La precisión (“Precision”) en estadística se refiere a la dispersión del conjunto de valores obtenidos de mediciones repetidas de una magnitud. Se define como la proximidad existente entre los vales medios obtenidos en mediciones repetidas de un mismo objeto bajo unas condiciones específicas. Cuanto menor es la dispersión de una muestra, mayor será su precisión (Armenteros, 2010). Se calcula como (*Ecuación 4*):

$$Precisión = \frac{VP}{VP + FP}$$

Ecuación 4: Cálculo de la Precisión para matrices de confusión

En la figura que se muestra a continuación (*Figura 6*), se puede ver gráficamente cómo se representa la exactitud y la precisión en la medida de un valor. Donde la exactitud indica la proximidad de los resultados de la medición respecto al valor verdadero (valor de referencia) y la precisión indica la repetibilidad de la medida.

En la *Figura 7*, se observan las distintas posibilidades que puede darse en la representación gráfica de precisión y exactitud. La línea azul representa la gráfica resultante para valores precisos, pero poco exactos; la línea roja representa la gráfica resultante para valores imprecisos e inexactos; la línea negra representa la gráfica resultante para valores exactos pero imprecisos; por último, la línea verde representa la gráfica resultante para valores precisos y exactos.

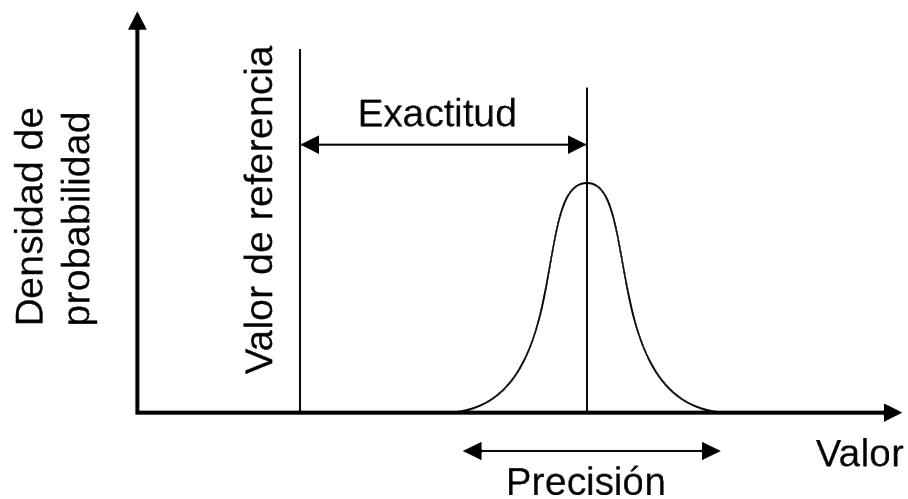


Figura 6: Representación gráfica de la Exactitud y la Precisión

Graph Accuracy Precision

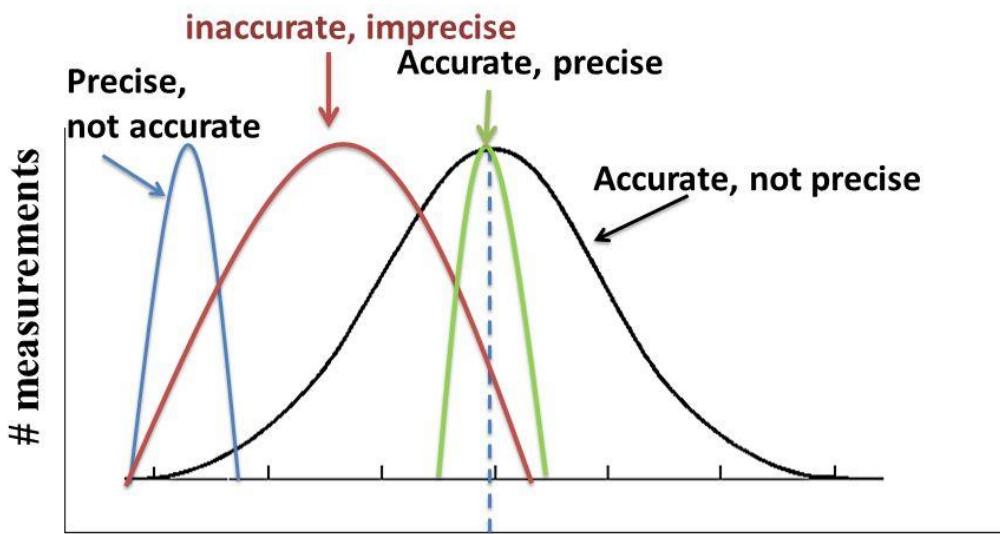


Figura 7: Representación gráfica de los posibles valores de la Exactitud y la Precisión

2.8 Error Medio

Se define el error medio o “Misclassification rate” (%), como la tasa de clasificación que se ha realizado erróneamente por el sistema (Pierre Rebours, 2006). Se calcula como (Ecuación 5):

$$\text{Error Medio} = \frac{FP + FN}{VP + FP + FN + VN}$$

Ecuación 5: Cálculo del Error Medio para matrices de confusión

2.9 Valor-F

También denominado “F-score”, es la medida de precisión que tiene una prueba. Para calcular este valor, se considera tanto la precisión como la sensibilidad, siendo su

valor final la media armónica de ambas y alcanzando su valor ideal en 1 (precisión y sensibilidad perfectas) (Reddy, 2014). Se calcula como (*Ecuación 6*):

$$Valor - F = 2 \times \frac{\text{Sensibilidad} * \text{Precisión}}{\text{Sensibilidad} + \text{Precisión}}$$

Ecuación 6: Cálculo del Valor-F para matrices de confusión

2.10 Ratio de Falsos Positivos

También denominado “Fall-out”, este ratio evalúa la probabilidad de que se rechace de manera errónea la hipótesis nula para una prueba en particular (Burke, 1988). Se calcula como (*Ecuación 7*):

$$\text{Ratio - FP} = \frac{FP}{FP + VN}$$

Ecuación 7: Cálculo del Ratio de Falsos Positivos para matrices de confusión

2.11 Coeficiente de correlación de Matthews

Denominado en inglés como “Matthews correlation coefficient (MCC)”, es utilizado para medir la calidad de una clasificación binaria (Matthews, 1975). Este coeficiente utiliza para sus medidas una matriz de confusión que cuenta con los valores de verdaderos y falsos negativos y positivos, es una medida bastante equilibrada y que puede ser utilizada en clases con tamaños bastante diferentes (Sabri Boughorbel, 2017). En esencia, es un coeficiente de correlación entre las clasificaciones binarias observadas y predichas, devolviendo un valor situado entre -1 y +1, donde +1 corresponde a una predicción perfecta, 0 a una predicción igual a otra aleatoria, y -1 a una predicción sin aciertos. En general, se considera una de las mejores medidas de este tipo en matrices de confusión.

Se calcula como (*Ecuación 8*):

$$\text{Coeficiente de Matthews} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)/TN + FN}}$$

Ecuación 8: Cálculo del Coeficiente de correlación de Matthews para matrices de confusión

2.12 Índice de Jaccard

Denominado en inglés “Jaccard index”, también se puede nombrar coeficiente de similitud de Jaccard. Mide el grado de similitud entre dos conjuntos de elementos, y se define como el tamaño de la intersección dividido por el tamaño de la unión de los conjuntos de muestras. Se calcula con la *Ecuación 9*, es decir, la cardinalidad de la intersección de ambos conjuntos (*Figura 8*) dividida entre la cardinalidad de su unión (*Figura 9*) (Real, 1999).

$$\text{Jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Ecuación 9: Cálculo general del Índice de Jaccard

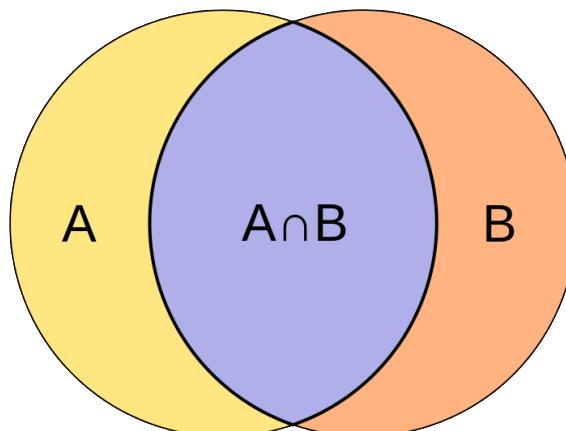


Figura 8: Intersección de los conjuntos A y B

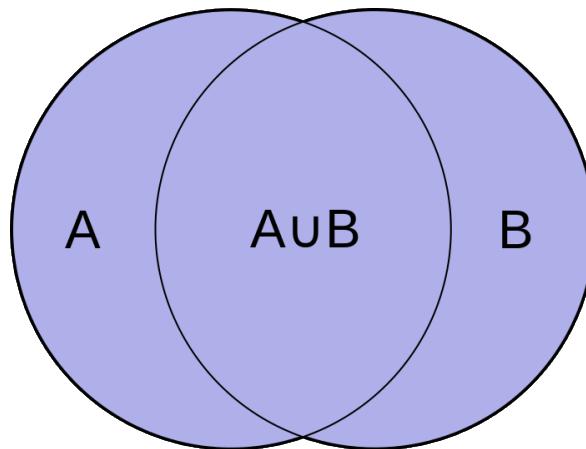


Figura 9: Unión de los conjuntos A y B

El intervalo de valores para este índice va desde 0 (no hay datos compartidos entre ambos conjuntos) hasta 1 (los dos conjuntos tienen los mismos datos) (Reyes, 2009). Para matrices de confusión, este índice se calcula como (*Ecuación 10*):

$$Jaccard = \frac{VP}{VP + FP + FN}$$

Ecuación 10: Cálculo del Índice de Jaccard para matrices de confusión

2.13 Parámetro D'

El parámetro d' (o índice de sensibilidad de la respuesta) es una medida estadística ampliamente usada en la teoría de detección de señales (TDS), mide la distancia entre la media de la señal y de las distribuciones de ruido comparadas con la desviación estándar de la distribución de la señal o del ruido (MacMillan & Creelman, 2005). Para la señal y el ruido de una distribución normal, el parámetro d' se representaría como (*Figura 10*):

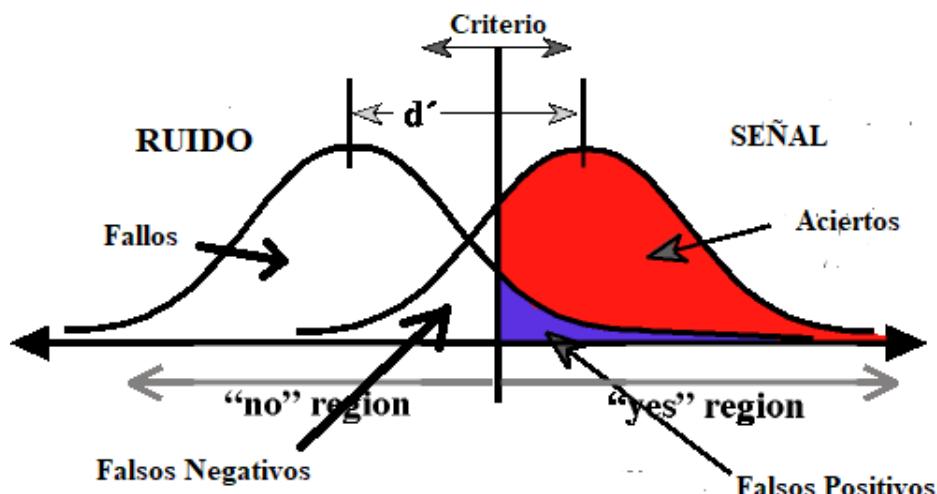


Figura 10: Representación parámetro d' en distribución normal

Cuanto más elevado sea el parámetro d' mejor será la realización del RAH (mayor tasa de acierto y menor tasa de fallos), siendo +infinito el resultado ideal.

Para una matriz de confusión 2x2, se puede calcular el Parámetro D' (Green, 1966) para cada elemento (letra) utilizando la fórmula (*Ecuación 11*):

$$d' = z(H) - z(FA)$$

Ecuación 11: Cálculo del Parámetro D' para matrices de confusión

Donde H es la probabilidad condicional de acierto, es decir, la probabilidad de que el programa reconozca una letra dada sabiéndose de antemano que era esa letra dada, se calcula como las veces que el programa ha acertado la letra (VP), dividido entre las veces que realmente era la letra (VP+FN). FA es la probabilidad condicional de fallo, es decir, la probabilidad de que el programa reconozca una letra dada sabiéndose de antemano que realmente no es esa letra, se calcula como las veces que el programa ha predicho la letra erróneamente (FP) y dividido entre el total de letras que no son esa letra dada (FP+VN). Una vez conocidas estas dos probabilidades condicionales, se debe aplicar la función normal de distribución inversa $z(q)$, también llamada función de distribución acumulada inversa (ICDF), para cada una de ellas y realizar la diferencia de ambos resultados. Pudiéndose expresar $z(q)$ en términos de la función error complementaria $\text{erfc}(x)$ como (*Ecuación 12*):

$$z(q) = -\sqrt{2}\text{erfc}^{-1}(2q)$$

Ecuación 12: Función Normal de Distribución Inversa

Definiéndose la función de error complementaria a partir de la función error $\text{erf}(x)$ como (*Ecuación 13*) (Andrews, 1998):

$$\text{erfc}(x) = 1 - \text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_x^{\infty} e^{-t^2} dt$$

Ecuación 13: Función de Error Complementaria

Esta función $z(q)$ es el valor de z para que $P(Z < z) = q$, asumiéndose que Z es una variable aleatoria normal. La ICDF calcula el inverso de la función de distribución acumulada (CDF) al determinar el valor asociado con una probabilidad acumulada específica (Glen, 2015).

2.14 Librería y tipos

Una librería o biblioteca es un conjunto de implementaciones funcionales, codificada en un lenguaje de programación y que ofrece una interfaz definida para la funcionalidad que invoca. Se diferencia de un programa ejecutable porque la librería no espera ser utilizada de manera autónoma, su finalidad es ser utilizada por otros programas, de manera independiente y de forma simultánea.

Las librerías pueden vincularse a un programa o a otras librerías en distintos puntos del desarrollo o de la ejecución, según el tipo de vínculo (o enlace), se pueden diferenciar dos tipos principales: librerías estáticas y librerías dinámicas (Wikipedia, Biblioteca (informática), 2020).

2.14.1Librería estática

Históricamente, las librerías solo podían ser estáticas; una librería estática, también conocida como archivo, es un fichero contenedor con varios archivos de código objetos empaquetados que posteriormente durante la compilación en el proceso de enlazado serán copiados y relocalizados en el fichero ejecutable final junto con el resto de los ficheros de código objeto (Wikipedia, Biblioteca (informática), 2020). Es decir, esta librería “se copia” en nuestro programa cuando se compila y, una vez que se tiene el ejecutable del programa, la librería no es utilizada más veces en este proyecto, por lo que incluso se podría borrar y el programa seguiría funcionando ya que este tiene copia de la parte de la librería que necesita para su ejecución (Commons, 2007).

En Windows tienen la extensión “.lib” y en Linux la extensión “.a”.

2.14.2Librería dinámica

Una librería dinámica es un fichero que contiene un código objeto construido de forma independiente a su ubicación de tal manera que están preparadas para ser utilizadas y cargadas en tiempo de ejecución por cualquier programa, en vez de tener que ser enlazadas previamente en el tiempo de compilación. Por lo tanto, han de estar siempre

disponibles como ficheros independientes al programa ejecutable para que este funcione, normalmente localizadas en directorios del sistema (Wikipedia, Biblioteca (informática), 2020). Durante el proceso de enlazado en la compilación, se genera un fichero ejecutable con anotaciones de las librerías que el programa requiere.

Es decir, el programa no copia la librería cuando este se compila. En vez de esto, cada vez que el código del ejecutable necesite alguna función de la librería, tendrá que buscarla, devolviendo un error en caso de que no la encuentre (Commons, 2007).

En Windows tienen la extensión “.dll”, en Linux la extensión “.so” y en Mac OS X la extensión “.dylib”.

2.14.3 Ventajas e inconveniente de cada tipo de librería

Una gran ventaja de las librerías dinámicas respecto a las estáticas es que se permite la reutilización del código y del espacio físico del sistema. Un mismo fichero de librería compartida puede ser utilizado por varios programas sin que estos copien en su interior su contenido; además, puede reutilizarse la memoria RAM para programas que utilicen la misma biblioteca.

El mayor inconveniente de las librerías dinámicas respecto a las estáticas es el tiempo de carga y los posibles errores de dirección a la hora de llamar a las distintas funciones en la librería. Esto se debe a que el programa debe buscar el fichero de la librería, cargarlo y relocalizar las llamadas que realiza este programa (Wikipedia, Biblioteca (informática), 2020).

Un programa compilado con librerías estáticas, al contrario de lo que ocurre en las dinámicas, se puede llevar a otro ordenador sin llevar también las librerías (Commons, 2007).

3. Desarrollo Inicial

En este punto, se describen las fases de desarrollo sucedidas en este trabajo de fin de grado. El trabajo se realiza en un MacBook Pro con sistema operativo macOS Catalina. Algunos procesos de instalación serían distintos en el caso de utilizar Windows o Linux.

3.1 Instalación de Kaldi

En macOS, la instalación del software Kaldi se realiza desde el terminal, utilizando un script de shell (archivo de texto que contiene uno o más comandos UNIX) (Apple, 2020). El software se localiza en la plataforma GitHub, y se debe acceder a un enlace en esta plataforma para descargarlo.

Previamente a la descarga, hay que asegurarse de tener descargados las siguientes herramientas:

- automake: Es una herramienta que genera automáticamente archivos Makefile.in (archivos de texto que definen un conjunto de tareas a ejecutar) portables para el uso de make, usado en la compilación de software, los cuales cumplen los estándares de codificación GNU (GNU, Automake, 2018). Para su uso, es necesario autoconf, un paquete que produce script de shell para configurar automáticamente paquetes de código fuente de software (GNU, Autoconf, 2016).
- wget: Es una herramienta libre utilizada para recuperar archivos usando diferentes protocolos de internet creada por el Proyecto GNU, es decir, permite de forma simple descargar contenidos desde servidores web. Es una herramienta de línea de comandos no interactiva, por lo que se puede llamar fácilmente desde scripts, trabajos cron, terminales sin soporte de X-Windows, etc. (GNU, GNU Wget, 2019).
- sox: Es un comando multiplataforma que permite trabajar con archivos de audio. Puede leer y escribir estos archivos de audio pudiendo combinar múltiples fuentes de entrada, sintetizar audio o actuar como un reproductor o grabadora. Se utiliza mediante el comando sox (Bagwell, 2013).

- gcc (GNU Compiler Collection): Es un conjunto de compiladores creados por el proyecto GNU, es un software libre distribuido por FSF bajo la licencia general pública GPL (GNU, GCC, the GNU Compiler Collection, 2020).
- libtool: Es un script genérico de soporte de bibliotecas creado por el Proyecto GNU, usada para crear bibliotecas de software portables. Tiene la complejidad de usar bibliotecas compartidas detrás de una interfaz consistente y portátil (Vaughan, 2016).

Para instalar estas herramientas en el sistema, se utilizará Homebrew, un sistema de gestión de paquetes que ayuda a la instalación y actualización de programas en los sistemas operativos macOS y GNU/Linux (Wikipedia, Homebrew (gestor de paquetes), 2020). Destaca por su facilidad de uso e integración con la línea de comandos; realiza un uso extensivo de GitHub para dar soporte a más paquetes (Terpstra, 2009). Se instalará brew con la orden (*Figura 11*):

```
(base) davidgomezgomez@MacBook-Pro-de-David ~ % /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"
```

Figura 11: Instalación Homebrew en macOS

Para realizar la instalación del software Kaldi, se han realizado las siguientes consideraciones:

- Se ha elegido utilizar la librería dinámica “.dylib” debido a que se considera que cuenta con mejores características para este caso concreto, pudiéndose ver las diferencias entre los tipos de librerías en la comparación realizada en el apartado “Ventajas e inconveniente de cada tipo de librería”. Al contar con un programa bastante pesado, una librería dinámica requiere un menor uso de memoria del sistema y memoria principal.
- En el make, se define como 8 el número de comandos que se pueden ejecutar simultáneamente. Se realiza mediante la marca “-j” (Jobs) (tutorialspoint, 2020).

Para facilitar la instalación de este software en otros terminales (con el sistema operativo macOS), se ha creado un script de instalación, el cual se encuentra en el *II.I*

ANEXO I: Instalación de Kaldi y Receta ASICA (Tabla 20: Instalación Kaldi en macOS).

Para que el software Kaldi funcione correctamente es necesario instalar “SRILM”, un kit de herramientas desarrollado para construir y aplicar modelos estadísticos de lenguaje, principalmente usado en el reconocimiento de voz, etiquetado y segmentación estadística y en la traducción automática (SRI, 2019). Una vez descargado el archivo de “SRILM” (SRILM, 2019), se debe trasladar a la carpeta “Kaldi/tools”, dentro de la carpeta donde se instaló Kaldi y ejecutarlo siguiendo los siguientes pasos:

1. Descomprimir el archivo.
2. En el script “Makefile”, eliminar el comentario de la parte donde se hace referencia a la localización de la carpeta srilm y sustituirlo por la ruta a esta carpeta dentro de nuestro ordenador.
3. Situarse en la carpeta srilm desde el script de shell del terminal y ejecutar: “cd tools”, “cd srilm”, “make”.

Una vez completados estos pasos, el software Kaldi estaría ya instalado. Para comprobar que funciona correctamente, es necesario instalar la receta que se quiera utilizar e implementarla con las pruebas que se deseen.

3.2 Instalación receta ASICA

En primer lugar, se debe instalar la receta ASICA (Receta ASICA para Kaldi), el proceso de instalación se describe en el repositorio de Git (Moreno-Torres Sánchez, 2019). Para poder realizar la instalación de manera más fácil, se ha realizado un script, disponible en el *II.I ANEXO I: Instalación de Kaldi y Receta ASICA (Tabla 21: Instalación Receta ASICA)*.

Para que la receta funcione de manera correcta, es necesario tener instalado Python 3 y las librerías:

- pandas: Librería de software escrita como una extensión de NumPy. Es utilizada para la manipulación y análisis de datos en el lenguaje de programación Python. Es de software libre y está distribuido bajo la licencia BSD (Abder-Rahman, 2016).

- `times`: Utilizada para manejar las tareas relacionadas con el tiempo como las conversiones horarias (Nvie, 2014).
- `subprocess.run`: Librería utilizada para ejecutar “processes” (Xando, 2013).
- `os`: Este módulo permite acceder a funcionalidades dependientes del Sistema Operativo, en especial aquellas que refieren información sobre el entorno del SO y permite manipular la estructura de directorios (uniwebsidad, 2020).
- `errno`: Este módulo permite los símbolos estándar del sistema errno. El valor de cada símbolo es su valor entero correspondiente (docs.python, 2020).
- `csv`: Este módulo permite importar y exportar hojas de cálculo y bases de datos (Rodríguez, 2018).
- `shutil`: Este módulo incluye operaciones de archivos de alto nivel como copiar y archivar (Rico Schmidt, 2018).

En caso de no tener alguna instalada, se puede instalar mediante el comando “\$ pip install nombre_*librería*”.

Una vez se tenga la receta ASICA instalada, se debe comprobar que funciona de manera correcta. Se debe tener en cuenta que la receta ASICA lleva consigo varias pruebas de voz, cuenta con los archivos “.kal” que son archivos con anotaciones asociadas a un archivo de audio “.wav”. Se realizan las siguientes comprobaciones:

1. Evaluación de la precisión del reconocimiento del habla (Accuracy) para el archivo de audio de un paciente (10150QL2). Se sabe que ASICA utiliza los audios de la carpeta “audio/train” para entrenar el sistema, y los de la carpeta “audio/test” para evaluarlo. Desde el script de shell, situarse en la carpeta “/ASICAKaldiRecipe” y ejecutar “ASICA.py” (script principal de la receta) mediante el comando “\$ python3 ASICA.py”. Se obtiene el siguiente resultado (*Figura 12*):

```

Speaker ID Accuracy
0 10150QL2_BASAL    60.48

-----
|EXECUTION TIME --> 308.2848460674286 secs |

-----

|EXECUTION TIME --> 5.138080767790476 min |

```

Figura 12: Prueba 1 - Funcionamiento ASICA.py

Donde se puede observar que, para el audio del sujeto “10150QL2_BASAL”, se ha obtenido un porcentaje de acierto del 60,48 % y el tiempo de ejecución ha sido aproximadamente de 5,14 minutos. Se podrá obtener un fichero “.txt” con los resultados de la ejecución mediante el comando “\$ python3 result_format.py”.

2. Evaluación de la capacidad del sistema para realizar RAH para un paciente determinado. Para ello, el programa cuenta con una serie de audios con los que se entrena, para la evaluación utiliza otro audio similar, y comprueba el grado de acierto comparando los resultados obtenidos y los resultados esperados (localizados en la carpeta “audio/test”). Para ejecutar esta prueba y una vez dentro de “/ASICAkaldiRecipe”, se ejecuta en el script de shell “\$ bash crossval_spk.sh + 001 123”, con lo que se evalúa a los sujetos 001 y 123, devolviendo el programa el porcentaje de acierto (exactitud) en el RAH para cada caso. Se obtiene como resultado la exactitud de sílabas reconocidas para cada sujeto (*Figura 13*) y una tabla con todos los resultados obtenidos (*Figura 14*):

```

Speaker ID Accuracy
0 CL001QL1_H54    86.36
2 CL001QL2_H54    86.05
4 CL001QL3_H54    83.33
6 CL001QL4_H54    86.36

-----
|EXECUTION TIME --> 314.75326681137085 secs |

-----

|EXECUTION TIME --> 5.2458877801895145 min |

```

Figura 13: Prueba 2.1 ASICA - Funcionamiento crossval_spk.sh

```

Guardo los resultados de 001
guardados en results/crossval_spk_001.txt
devuelvo los *.kal de 001 a su sitio
    Speaker ID Task Evaluation ... Responded Utterance Number of Syllables Syllable position
 0      CL001   QL1     H54 ...           ya te          2             1
 1      CL001   QL1     H54 ...           ya te          2             2
 2      CL001   QL1     H54 ...         rre lo lu        3             1
 3      CL001   QL1     H54 ...         rre lo lu        3             2
 4      CL001   QL1     H54 ...         rre lo lu        3             3
 ..
 523     ...     ...     H54 ...           ...          ...           ...
 524     CL001   QL4     H54 ...       se nyo ri da      4             2
 525     CL001   QL4     H54 ...       se nyo ri da      4             3
 526     CL001   QL4     H54 ...       se nyo ri da      4             4
 527     CL001   QL4     H54 ...           tu zo          2             1
                                         tu zo          2             2
[528 rows x 11 columns]

```

Figura 14: Prueba 2.2 ASICA - Funcionamiento crossval_spk.sh

Tras realizar estas dos pruebas, se verifica que la receta está instalada de manera correcta.

3.3 Obtención de datos para realizar la librería

Una vez que se entrena al programa con una base de archivos de audio de distintos sujetos y se realiza una prueba de RAH para un archivo de audio, se obtienen los resultados devueltos por el programa. El proyecto ASICA (Moreno-Torres Sánchez, 2019), cuenta con un script en su receta, llamado “result_format.py”, con el cuál se puede obtener un archivo de texto (“salida.txt”) que contiene múltiples parámetros relacionados con los resultados obtenidos para la prueba de RAH realizada; estos parámetros en el archivo de texto están separados por tabuladores (importante a la hora de leer este archivo en Python). Se puede observar un ejemplo de la vista de este archivo en la *Figura 15*. Los resultados obtenidos en él son:

- N
- Speaker ID
- Task
- Evaluation
- Target Response
- Hit
- Correct

- Target Utterance
- Responded
- Utterance
- Number of Syllables
- TargetC
- TargetV
- RespC
- RespV
- TargetCManner
- TargetCVoice
- TargetCPlace
- RespCanner
- RespVoice
- RespPlace
- CorrC
- CorrV
- CorrManner
- CorrVoice
- CorrPlace

N	Speaker	ID	Task	Evaluation	TargetC	Response	Hit	Correct	Target	Utterance	CorrManner	CorrVoice	Responded	Utterance	Number of Syllables	CorrPlace	TargetC	TargetV	RespC	RespV									
0	TargetChanner	TargetVoice	TargetCPlace	TargetC	1	ya	te	2	t	e	a	y	a	fric	Vless	Coronal	Coronal	1	1	1	1								
1	CL001	QL1	H54	te	te	c	1	ya	te	ya	te	2	t	e	t	e	Qcl	Vless	Coronal	Qcl	Vless	Coronal 1	1	1	1				
2	CL001	QL1	H54	***	***	I	0	***	no	lu	***	no	lu	lu	3	**	*	Qcl	e	Other	Other	Other	Aprox	Voiced	Coronal	0	0	0	
3	CL001	QL1	H54	no	lo	S	0	***	no	lu	***	no	lu	lu	3	n	o	l	o	nas	Voiced	Coronal	Aprox	Voiced	Coronal	0	1	0	
4	CL001	QL1	H54	lu	lu	C	1	***	no	lu	***	no	lu	lu	3	l	u	u	u	Aprox	Voiced	Coronal	Aprox	Voiced	Coronal	1	1	1	
5	CL001	QL1	H54	chu	chu	c	1	chu	lo	chu	lo	2	ch	u	ch	u	Aflic	Vless	Coronal	Aflic	Vless	Coronal	1	1	1	1			
6	CL001	QL1	H54	lo	lo	c	1	chu	lo	chu	lo	2	l	o	l	o	Aprox	Voiced	Coronal	Aprox	Voiced	Coronal	1	1	1	1			
7	CL001	QL1	H54	pa	pa	c	1	pa	ba	ri	no	pa	ba	ri	no	4	p	a	p	a	Qcl	Vless	Frontal	Qcl	Vless	Frontal	1	1	1
8	CL001	QL1	H54	ba	ba	c	1	pa	ba	ri	no	pa	ba	ri	no	4	b	a	b	a	Qcl	Voiced	Frontal	Qcl	Voiced	Frontal	1	1	1
9	CL001	QL1	H54	ri	ri	c	1	pa	ba	ri	no	pa	ba	ri	no	4	r	i	r	i	Aprox	Voiced	Coronal	Aprox	Voiced	Coronal	1	1	1
10	CL001	QL1	H54	no	no	c	1	pa	ba	ri	no	pa	ba	ri	no	4	n	o	n	o	nas	Voiced	Coronal	nas	Voiced	Coronal	1	1	1
11	CL001	QL1	H54	ko	ko	c	1	ko	pa	di	ko	pa	ri	3	k	o	k	o	Qcl	Vless	back	Qcl	Vless	back	1	1	1		
12	CL001	QL1	H54	pa	pa	c	1	ko	pa	di	ko	pa	ri	3	p	a	p	a	Qcl	Vless	Frontal	Qcl	Vless	Frontal	1	1	1		
13	CL001	QL1	H54	di	ri	S	0	ko	pa	di	ko	pa	ri	3	d	i	r	i	Qcl	Voiced	Coronal	Aprox	Voiced	Coronal	0	1	0		
14	CL001	QL1	H54	li	li	c	1	li	ga	li	ga	2	l	i	l	i	Aprox	Voiced	Coronal	Aprox	Voiced	Coronal	1	1	1				
15	CL001	QL1	H54	ga	ga	c	1	li	ga	li	ga	2	g	a	g	a	Qcl	Voiced	back	Qcl	Voiced	back	1	1	1				
16	CL001	QL1	H54	be	be	c	1	be	yo	lu	be	yo	lu	3	b	e	b	e	Qcl	Voiced	Frontal	Qcl	Voiced	Frontal	1	1	1		
17	CL001	QL1	H54	yo	yo	c	1	be	yo	lu	be	yo	lu	3	y	o	y	o	fric	Voiced	Coronal	fric	Voiced	Coronal	1	1	1		
18	CL001	QL1	H54	lu	lu	c	1	be	yo	lu	be	yo	lu	3	l	u	l	u	Aprox	Voiced	Coronal	Aprox	Voiced	Coronal	1	1	1		
19	CL001	QL1	H54	so	so	c	1	so	te	lo	so	te	lo	3	s	o	s	o	fric	Vless	Coronal	fric	Vless	Coronal	1	1	1		
20	CL001	QL1	H54	te	te	c	1	so	te	lo	so	te	lo	3	t	e	t	e	Qcl	Vless	Coronal	Qcl	Vless	Coronal	1	1	1		
21	CL001	QL1	H54	lo	lo	c	1	so	te	lo	so	te	lo	3	l	o	l	o	Aprox	Voiced	Coronal	Aprox	Voiced	Coronal	1	1	1		
22	CL001	QL1	H54	ma	ma	c	1	ma	re	xa	za	ma	re	xa	za	4	m	a	m	a	nas	Voiced	Frontal	nas	Voiced	Frontal	1	1	1
23	CL001	QL1	H54	re	re	c	1	ma	re	xa	za	ma	re	xa	za	4	r	e	r	e	Aprox	Voiced	Coronal	Aprox	Voiced	Coronal	1	1	1
24	CL001	QL1	H54	xa	xa	c	1	ma	re	xa	za	ma	re	xa	za	4	x	a	x	a	fric	Vless	back	fric	Vless	back	1	1	1
25	CL001	QL1	H54	za	za	c	1	ma	re	xa	za	ma	re	xa	za	4	z	a	z	a	fric	Vless	Coronal	fric	Vless	Coronal	1	1	1
26	CL001	QL1	H54	nu	mu	S	0	nu	me	ro	nu	me	ro	3	n	u	m	u	nas	Voiced	Coronal	nas	Voiced	Frontal	0	1	1		
27	CL001	QL1	H54	me	me	c	1	nu	me	ro	nu	me	ro	3	m	e	m	e	nas	Voiced	Frontal	nas	Voiced	Frontal	1	1	1		

Figura 15: Ejemplo de archivo de texto “salida.txt”

Para obtener este archivo, debe ejecutarse un experimento básico de la receta ASICA realizando los siguientes pasos (Moreno-Torres Sánchez, 2019):

1. Colocar los archivos de audio en la carpeta audio/experimento1 (si se prefiere otra ubicación, se debe especificarla en “config.py”).
2. Colocar los archivos de entrenamiento “.kal” en “info_user/control”.
3. Colocar los archivos “.kal” de prueba en “info_user/test”.
4. En la terminal, dentro de la carpeta “kaldi/egs/ASICAKaldiRecipe” se debe ejecutar “check_format.py”. Si hay errores, se deben editar los archivos “.kal”. Se llamará desde el terminal mediante “\$ python3 check_format.py”.
5. Ejecutar el script principal “ASICA.py”. Se llamará desde el terminal mediante “\$ Python 3ASICA.py”.
6. Ejecutar el script “result_format.py”. Se llamará desde el terminal mediante “\$ python3 result_format.py speakerID fileOut”, donde “speakerID” es el código de identificación del hablante, y “fileOut” el archivo que se ejecuta, un ejemplo de ejecución sería “python3 result_format.py CL001 CL001.csv”.

Realizando estos pasos, se obtendrá dentro de la carpeta “results”, el archivo de texto “salida.txt”, el cual cuenta con los resultados obtenidos del RAH para un archivo de audio, en base a estos resultados, se realizará la librería de evaluación Python.

4. Desarrollo de la librería

En este trabajo, se desarrollará la librería Python de evaluación de resultados obtenidos en problemas de RAH, esta librería se llamará “Lib_RAH_DGG.py”.

4.1 Conversión de datos

En este punto, se procesarán con Python los parámetros del archivo “salida.txt”, generado con la receta ASICA ejecutada en el software Kaldi (*Receta ASICA para Kaldi*). Este archivo de texto contiene los resultados obtenidos tras realizar el RAH, descrito en el apartado “*Obtención de datos para realizar la librería*”.

Para poder realizar la librería y probarla, se necesita importar las siguientes librerías: “os”, “numpy”, “pandas”, “math”, “seaborn”, “matplotlib”, “matplotlib.pyplot”, “colors” desde “matplotlib”, “norm” desde “scipy.stats” y “ListedColormap” desde “matplotlib”.

En primer lugar, se debe convertir el archivo de texto a un DataFrame (colección ordenada de columnas con nombres y tipos, donde una sola fila representa un único caso y las columnas representan atributos particulares (Abder-Rahman, 2016)) con el que poder trabajar en Python. Una vez convertido este archivo, también se cambia el nombre de la columna 'Number of Syllables' por el nuevo nombre 'NSyllables'. Se puede ver en *Tabla 1*.

Tabla 1: Transformación de "salida.txt" a un DataFrame en Python

```
import os
import numpy as np
import pandas as pd
import math
import seaborn as sn
import matplotlib.pyplot as plt
from matplotlib import colors
from scipy.stats import norm
```

```

from matplotlib import cm
from matplotlib.colors import ListedColormap

##### CONVERSION DE ARCHIVO .TXT A DATAFRAME PARA FACILITAR EL
TRABAJO EN PYTHON #####
#Sirve para establecer en qué carpeta se está trabajando (ruta)
working_path = os.getcwd()

#Se construye un dataframe importando el archivo .txt separado por tabuladores
texto_kaldi = pd.read_csv("salida.txt", delimiter="\t")

#Se cambia (acorta) el de la columna que indica el nº de sílabas para poder trabajar
con ella más fácilmente
texto_kaldi.rename(columns={'Number of Syllables': 'NSyllables'}, inplace = True)

```

4.2 Función de cálculo del porcentaje de sílabas reconocidas correctamente

Se desarrolla la función “*PorcentajeAciertos (df)*” (*Tabla 2*) con la que se calcula el porcentaje de sílabas que el software ha reconocido de manera correcta (porcentaje de aciertos). Esta función tiene como parámetro de entrada el dataframe con los datos obtenidos al aplicar la receta ASICA y calcula e imprime en pantalla el porcentaje de sílabas reconocidas correctamente de manera general y para palabras con dos, tres y cuatro sílabas de manera individual.

Se puede encontrar un ejemplo del funcionamiento de esta función en el apartado “*Prueba del porcentaje de sílabas reconocidas correctamente*”.

Tabla 2: Función de cálculo del porcentaje de acierto Python

```

##### CALCULO DE EL PORCENTAJE DE ACIERTO SEGUN EL NUMERO DE
SILABAS DE LA PALABRA #####
def PorcentajeAciertos (df):
    #Porcentaje de palabras de 2 sílabas

```

```

two_syl = round((len(df[df.NSyllables == 2])/ len(df))*100, 2)
#Porcentaje de palabras de 3 sílabas
three_syl = round((len(df[df.NSyllables == 3])/ len(df))*100, 2)
#Porcentaje de palabras de 4 sílabas
four_syl = round((len(df[df.NSyllables == 4])/ len(df))*100, 2)

#Calcula el % de total, crea un array en el que solo se queda con los elementos
que df.prueba == 1 y lo divide por el total, redondeo a dos decimales
Corr_Total = round(((len(df[df.Correct == 1])/ len(df)) * 100), 2)
#Calcula el % de acierto para palabras de 2 sílabas, redondeamos a dos
decimales
Corr_two = round(((len(df[(df.Correct == 1) & (df.NSyllables == 2)])/
len(df[df.NSyllables == 2])) * 100), 2)
#Calcula el % de acierto para palabras de 3 sílabas, redondeamos a dos
decimales
Corr_three = round(((len(df[(df.Correct == 1) & (df.NSyllables == 3)])/
len(df[df.NSyllables == 3])) * 100), 2)
#Calcula el % de acierto para palabras de 4 sílabas, redondeamos a dos
decimales
Corr_four = round(((len(df[(df.Correct == 1) & (df.NSyllables == 4)])/
len(df[df.NSyllables == 4])) * 100), 2)

#Imprime los resultados de la función
print("El porcentaje total correcto es:", Corr_Total , "% \n")
print("El porcentaje correcto con 2 sílabas es:", Corr_two, "% ", "Con un porcentaje
de palabras total de:", two_syl, "% \n")
print("El porcentaje correcto con 3 sílabas es:", Corr_three, "% ", "Con un
porcentaje de palabras total de:", three_syl, "% \n")
print("El porcentaje correcto con 4 sílabas es:", Corr_four, "% ", "Con un porcentaje
de palabras total de:", four_syl, "% "\n")

```

El diagrama de flujo asociado a esta función se encuentra en la *Figura 16*. La función tiene una variable de entrada de tipo dataframe, en este caso, el dataframe es el correspondiente al archivo “salida.txt”. En primer lugar, se calcula el porcentaje de palabras con 2, 3 y 4 sílabas y se guardan en variables tipo float llamadas “two_syl”, “three_syl” y “four_syl”; posteriormente, se calcula el porcentaje de palabras totales y el

porcentaje de palabras de 2, 3 y 4 sílabas reconocidas correctamente (valores guardados en variables tipo float). Por último, se imprimen en pantalla los resultados obtenidos.

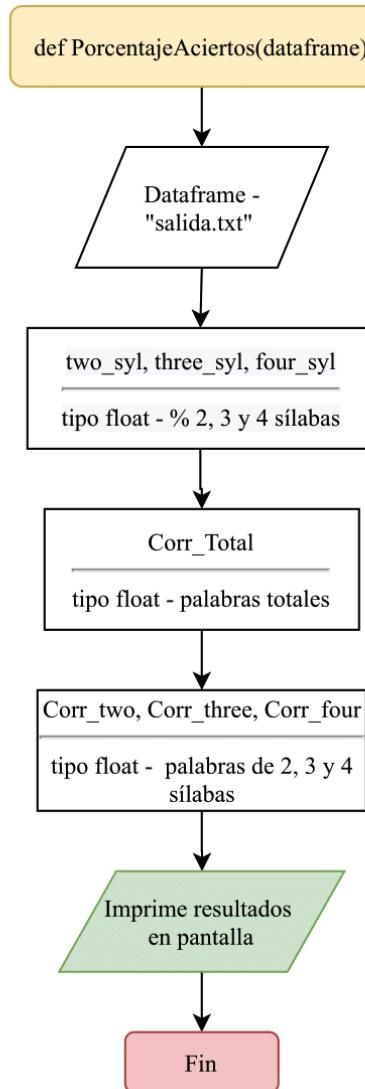


Figura 16: Diagrama de flujo de la función "PorcentajeAciertos (dataframe)"

4.3 Función de cálculo de matrices de confusión

Las funciones descritas en la *Tabla 3*, crean dos tipos de matriz de confusión, una con los datos totales y otra con los datos normalizados. Se han desarrollado cuatro funciones: “ConfusionMatrixVoc(df)”, “NormalConfusionMatrixVoc(df)”, “ConfusionMatrixCons(df)” y “NormalConfusionMatrixCons(df)”. Donde las dos

primeras calculan la matriz de confusión con los datos totales y normalizados para las vocales, y las dos segundas realizan los cálculos de la misma manera para las consonantes. Estas funciones obtienen las matrices de confusión en un dataframe y tienen como parámetro de entrada el dataframe obtenido con los datos obtenidos con la receta ASICA.

Se puede encontrar un ejemplo del funcionamiento de estas funciones en el apartado “*Prueba de matrices de confusión*”.

Tabla 3: Funciones de creación de matrices de confusión para vocales y consonantes

```
##### CALCULO DE LA MATRIZ DE CONFUSION TOTALES PARA VOCALES Y
CONSONANTES Y VISUALIZACION DE ESTAS MATRICES #####
#Matriz de confusión para VOCALES
def ConfusionMatrixVoc(df):
    conf_matV = pd.crosstab(df['TargetV'], df['RespV'], rownames=['Target'],
    colnames=['Response'], margins = True);
    #Elimina la fila All y *(no da información relevante) Vocales
    ncmV = conf_matV.drop(["All", "*"], axis = 0);
    #Obtiene la matriz de confusión total - Elimina la columna All y * (no da información relevante) Vocales
    confusion_matrixV = ncmV.drop(["All", "*"], axis = 1);
    return confusion_matrixV

#Matriz de confusión Normalizada para VOCALES
def NormalConfusionMatrixVoc (df):
    conf_matV = pd.crosstab(df['TargetV'], df['RespV'], rownames=['Target'],
    colnames=['Response'], margins = True);
    #Normalizacion de la matriz de confusión Vocales
    ncmV1 = conf_matV/conf_matV.max().astype(np.float64);
    #Elimina la fila All y * (no da información relevante) Vocales
    ncmV2 = ncmV1.drop(["All", "*"], axis = 0);
    #Elimina la columna All y *(no nos da información) Vocales
    ncmV3 = ncmV2.drop(["All", "*"], axis = 1);
    #Obtiene la Matriz de Confusión Normalizada - Redondea los datos a dos decimales en df Vocales
    normalize_confusion_matrixV = ncmV3.round(2)
```

```

return normalize_confusion_matrixV

#Matriz de confusión para CONSONANTES
def ConfusionMatrixCons(df):
    conf_matC = pd.crosstab(df['TargetC'], df['RespC'], rownames=['Target'],
    colnames=['Response'], margins = True);
    #Elimina la fila All y ** (no da información relevante) Consonantes
    ncmC = conf_matC.drop(["All", "***"], axis = 0);
    #Obtiene matriz de confusión total - Elimina las columnas All, **, gr y zr (no da
    información relevante) Consonantes
    confusion_matrixC = ncmC.drop(["All", "***","gr", "zr"], axis = 1);
    return confusion_matrixC

#Matriz de confusión Normalizada para CONSONANTES
def NormalConfusionMatrixCons(df):
    conf_matC = pd.crosstab(df['TargetC'], df['RespC'], rownames=['Target'],
    colnames=['Response'], margins = True);
    #Normalización de matriz de confusión Consonantes
    ncmC1 = conf_matC/conf_matC.max().astype(np.float64);
    #Eliminación de la fila All y ** (no da información relevante) Consonantes
    ncmC2 = ncmC1.drop(["All","***"], axis = 0);
    #Eliminacion las columnas All, **, gr y zr (no da información relevante)
    Consonantes
    ncmC3 = ncmC2.drop(["All", "***","gr", "zr"], axis = 1);
    #Obtención Matriz de Confusión Normalizada - Redondeo de los datos a dos
    decimales en df Consonantes
    normarlize_confusion_matrixC = ncmC3.round(2)
    return normarlize_confusion_matrixC

```

Se han realizado los diagramas de flujo de las funciones “*ConfusionMatrixVoc(df)*” y “*NormalConfusionMatrixVoc (df)*” como ejemplos de este apartado, se encuentran en la *Figura 17* y *Figura 18* respectivamente.

La función “*ConfusionMatrixVoc(dataframe)*” mostrada en el diagrama de flujo (*Figura 17*) lee entradas de tipo dataframe, en este caso, el dataframe es el correspondiente al archivo “salida.txt”. En primer lugar, se crea una variable de tipo dataframe (“*conf_matV*”) que guarda una matriz de confusión donde los valores predichos

corresponden a la columna “TargetV” y los valores observados corresponden a la columna “RespV” del dataframe de entrada. En segundo lugar, se crea una nueva variable (“confusion_matrixV”) de tipo dataframe que almacena una matriz similar a “conf_matV” pero sin las filas y columnas “All” y “*”, pues es información no relevante. La función devuelve la variable “confusion_matrixV”.

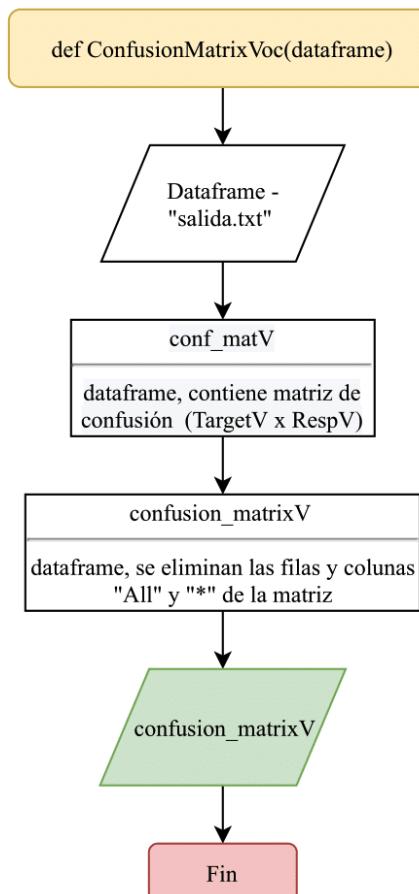


Figura 17: Diagrama de flujo de la función "ConfusionMatrixVoc (dataframe)"

La función “*NormalConfusionMatrixVoc(dataframe)*” mostrada en el diagrama de flujo (*Figura 18*) lee entradas de tipo dataframe, en este caso, el dataframe es el correspondiente al archivo “salida.txt”. Los pasos realizados en esta función son similares a los de la función “*ConfusionMatrixVoc(df)*”, pero normaliza los datos de la matriz (dataframe) antes de eliminar las filas y columnas “All” y “*”, los datos los devolverá con dos cifras decimales. La función devolverá la variable “*normalize_confusion_matrixV*”.

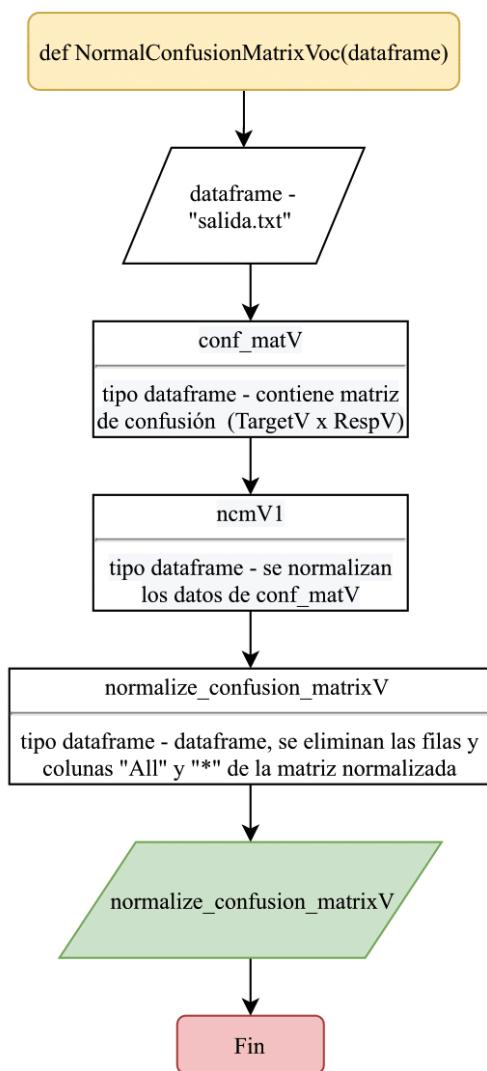


Figura 18: Diagrama de flujo de la función "NormalConfusionMatrixVoc (dataframe)"

4.4 Función de agrupación de consonantes

Las consonantes se pueden agrupar por diferentes criterios, tal y como se explica en el apartado “*Tipo de consonantes*”. Siguiendo estas pautas se realizan diez funciones para agrupar las consonantes en cada tipo según los criterios de Sonoridad, Lugar de articulación y modo de articulación y obtener su matriz de confusión, y otras diez para obtener estas mismas matrices de confusión de manera normalizada. Estas matrices se obtienen en una variable de tipo dataframe.

Las funciones para el cálculo de la matriz de confusión para estos grupos son: “CSonora(df)”, “CSorda(df)”, “CFrontal(df)”, “CCoronal(df)”, “CBack(df)”, “COclusiva(df)”, “CAfricada(df)”, “CFricativa(df)”, “CNasal(df)” y “CAproximante(df)”; las funciones para el cálculo de la matriz de confusión normalizada para estos grupos son: “NormalizeCSonora(df)”, “NormalizeCSorda(df)”, “NormalizeCFrontal(df)”, “NormalizeCCoronal(df)”, “NormalizeCBack(df)”, “NormalizeCOclusiva(df)”, “NormalizeCAfricada(df)”, “NormalizeCFricativa(df)”, “NormalizeCNasal(df)” y “NormalizeCAproximante(df)”.

Para realizar estas funciones (*Tabla 4*), se ha tomado como parámetro de entrada el dataframe con los datos obtenidos con la receta ASICA. Se utiliza la función “ConfusionMatrixCons(df)” para obtener una variable que almacene un dataframe con todas las consonantes, el cual posteriormente se filtra para obtener las letras deseadas en función del tipo de agrupación que se haga.

Se puede encontrar un ejemplo del funcionamiento de estas funciones en el apartado “Prueba de agrupación de consonantes”.

Tabla 4: Funciones de creación de matrices de confusión para las distintas agrupaciones de consonantes

```
##### AGRUPACIÓN DE CONSONANTES POR CRITERIO DE SONORIDAD,
LUGAR DE ARTICULACION Y MODO DE ARTICULACION #####
# 1. SONORIDAD: Sordas o Sonoras #
def CSonora(df):
    #Creación de un dataframe con las consonantes Sonoras
    ncmC1 = ConfusionMatrixCons(df)
    CSonora1 = ncmC1.filter(["b","d","g","l","m","n","ny","r","rr","y"], axis=1)
    Sonoridad_Sonora = CSonora1.filter(["b","d","g","l","m","n","ny","r","rr","y"], axis=0)
    return Sonoridad_Sonora

def CSorda(df):
    #Creación de un dataframe con las consonantes Sonoras
    ncmC1 = ConfusionMatrixCons(df)
    CSorda1 = ncmC1.filter(["ch","f","k","p","s","t","x","z"], axis=1)
```

```

Sonoridad_Sorda = CSorda1.filter(["ch","f","k","p","s","t","x","z"], axis=0)
return Sonoridad_Sorda

# 1.1 Sonoridad Normalizadas
def NormalizeCSonora(df):
    #Creación de un dataframe con las consonantes Sonoras normalizadas
    ncmC1 = NormalConfusionMatrixCons(df)
    CSonora1 = ncmC1.filter(["b","d","g","l","m","n","ny","r","rr","y"], axis=1)
    Sonoridad_Sonora = CSonora1.filter(["b","d","g","l","m","n","ny","r","rr","y"], axis=0)
    return Sonoridad_Sonora

def NormalizeCSorda(df):
    #Creaación de un dataframe con las consonantes Sonoras normalizadas
    ncmC1 = NormalConfusionMatrixCons(df)
    CSorda1 = ncmC1.filter(["ch","f","k","p","s","t","x","z"], axis=1)
    Sonoridad_Sorda = CSorda1.filter(["ch","f","k","p","s","t","x","z"], axis=0)
    return Sonoridad_Sorda

# 2. LUGAR DE ARTICULACION: Frontal, Coronal o Back #
def CFrontal(df):
    #Creación de un dataframe con las consonantes Frontal
    ncmC1 = ConfusionMatrixCons(df)
    CFrontal1 = ncmC1.filter(["b","f","m","p"], axis=1)
    Lugar_Frontal = CFrontal1.filter(["b","f","m","p"], axis=0)
    return Lugar_Frontal

def CCoronal(df):
    #Creación de un dataframe con las consonantes Coronal
    ncmC1 = ConfusionMatrixCons(df)
    CCoronal1 = ncmC1.filter(["ch","d","l","n","ny","r","rr","s","t","y","z"], axis=1)
    Lugar_Coronal = CCoronal1.filter(["ch","d","l","n","ny","r","rr","s","t","y","z"], axis=0)
    return Lugar_Coronal

def CBack(df):
    #Creación de un dataframe con las consonantes Back
    ncmC1 = ConfusionMatrixCons(df)
    CBack1 = ncmC1.filter(["g","k","x"], axis=1)
    Lugar_Back = CBack1.filter(["g","k","x"], axis=0)

```

```

    return Lugar_Back

# 2.1 Lugar de articulacion Normalizadas

def NormalizeCFrontal(df):
    #Creación de un dataframe con las consonantes Frontal normalizadas
    ncmC1 = NormalConfusionMatrixCons(df)
    CFrontal1 = ncmC1.filter(["b","f","m","p"], axis=1)
    Lugar_Frontal = CFrontal1.filter(["b","f","m","p"], axis=0)
    return Lugar_Frontal

def NormalizeCCoronal(df):
    #Creación de un dataframe con las consonantes Coronal normalizadas
    ncmC1 = NormalConfusionMatrixCons(df)
    CCoral1 = ncmC1.filter(["ch","d","l","n","ny","r","rr","s","t","y","z"], axis=1)
    Lugar_Coronal = CCoral1.filter(["ch","d","l","n","ny","r","rr","s","t","y","z"], axis=0)
    return Lugar_Coronal

def NormalizeCBack(df):
    #Creación de un dataframe con las consonantes Back normalizadas
    ncmC1 = NormalConfusionMatrixCons(df)
    CBack1 = ncmC1.filter(["g","k","x"], axis=1)
    Lugar_Back = CBack1.filter(["g","k","x"], axis=0)
    return Lugar_Back

# 3. MODO DE ARTICULACION: Oclusiva, Africada, Fricativa, Nasal, Aproximantes

def COclusiva(df):
    #Creación de un dataframe con las consonantes Oclusivas
    ncmC1 = ConfusionMatrixCons(df)
    COclusiva1 = ncmC1.filter(["b","d","g","k","p","t"], axis=1)
    Modo_Oclusiva = COclusiva1.filter(["b","d","g","k","p","t"], axis=0)
    return Modo_Oclusiva

def CAfricada(df):
    #Creación de un dataframe con las consonantes Africadas
    ncmC1 = ConfusionMatrixCons(df)
    CAfricada1 = ncmC1.filter(["ch"], axis=1)
    Modo_Africada = CAfricada1.filter(["ch"], axis=0)
    return Modo_Africada

```

```

def CFricativa(df):
    #Creación de un dataframe con las consonantes Fricativa
    ncmC1 = ConfusionMatrixCons(df)
    CFricativa1 = ncmC1.filter(["f","s","x","y","z"], axis=1)
    Modo_Fricativa = CFricativa1.filter(["f","s","x","y","z"], axis=0)
    return Modo_Fricativa

def CNasal(df):
    #Creación de un dataframe con las consonantes Nasal
    ncmC1 = ConfusionMatrixCons(df)
    CNasal1 = ncmC1.filter(["m","n","ny"], axis=1)
    Modo_Nasal = CNasal1.filter(["m","n","ny"], axis=0)
    return Modo_Nasal

def CAproximante(df):
    #Creación de un dataframe con las consonantes Aproximante
    ncmC1 = ConfusionMatrixCons(df)
    CAproximante1 = ncmC1.filter(["l","r","rr"], axis=1)
    Modo_Aproximante = CAproximante1.filter(["l","r","rr"], axis=0)
    return Modo_Aproximante

# 2.1 Modo de articulacion Normalizadas

def NormalizeCOclusiva(df):
    #Creación de un dataframe con las consonantes Oclusivas normalizadas
    ncmC1 = NormalConfusionMatrixCons(df)
    COclusiva1 = ncmC1.filter(["b","d","g","k","p","t"], axis=1)
    Modo_Oclusiva = COclusiva1.filter(["b","d","g","k","p","t"], axis=0)
    return Modo_Oclusiva

def NormalizeCAfricada(df):
    #Creación de un dataframe con las consonantes Africadas normalizadas
    ncmC1 = NormalConfusionMatrixCons(df)
    CAfricada1 = ncmC1.filter(["ch"], axis=1)
    Modo_Africada = CAfricada1.filter(["ch"], axis=0)
    return Modo_Africada

def NormalizeCFricativa(df):

```

```

#Creación de un dataframe con las consonantes Fricativa normalizadas
ncmC1 = NormalConfusionMatrixCons(df)
CFricativa1 = ncmC1.filter(["f","s","x","y","z"], axis=1)
Modo_Fricativa = CFricativa1.filter(["f","s","x","y","z"], axis=0)
return Modo_Fricativa

def NormalizeCNasal(df):
    #Creación de un dataframe con las consonantes Nasal normalizadas
    ncmC1 = NormalConfusionMatrixCons(df)
    CNasal1 = ncmC1.filter(["m","n","ny"], axis=1)
    Modo_Nasal = CNasal1.filter(["m","n","ny"], axis=0)
    return Modo_Nasal

def NormalizeCAproximante(df):
    #Creación de un dataframe con las consonantes Aproximante normalizadas
    ncmC1 = NormalConfusionMatrixCons(df)
    CAproximante1 = ncmC1.filter(["l","r","rr"], axis=1)
    Modo_Aproximante = CAproximante1.filter(["l","r","rr"], axis=0)
    return Modo_Aproximante

```

Se ha realizado el diagrama de flujo de la función “*CSonora(dataframe)*” como ejemplo de este apartado, se encuentran en la *Figura 19*. Esta función tiene como parámetro de entrada una variable de tipo dataframe, en este caso, el dataframe es el correspondiente al archivo “salida.txt”. Posteriormente, se llama a la función “*ConfusionMatrixCons(dataframe)*” para obtener la matriz de confusión correspondiente a todas las consonantes (se guarda en una variable de tipo dataframe llamada “ncmC1”). Por último, se obtiene la matriz de confusión para las consonantes sonoras y se guarda en la variable de tipo dataframe “Sonoridad_Sonora”; la función devolverá esta variable.

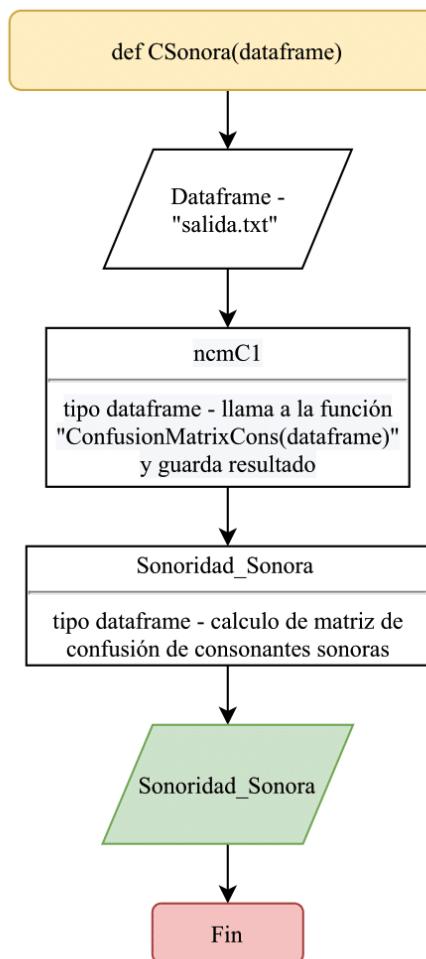


Figura 19: Diagrama de flujo de la función "CSonora(df)"

4.5 Función para graficar y colorear matrices normalizadas

Una forma de simplificar visualmente la detección de errores en la matriz de confusión por parte del usuario es graficar esta matriz coloreada. Debe ser posible diferenciar claramente tanto los valores correctos como los errados por el software en el RAH para así, diferenciar en qué letras se dan los errores de detección y en cuales se confunden.

Tras la evaluación de los posibles modos de colorear la matriz (*"II.II ANEXO II: Elección de gradación de color para graficar matrices"*), se ha optado por realizar las dos funciones "*"PlotMatOrangeBlue(normalize_confusion_matrix, nombre)"*" y "*"PlotMatViridis(normalize_confusion_matrix, nombre)"*" (*Tabla 5*), que utilizan una

gradación de colores naranja-azul y viridis respectivamente. Ambas funciones tienen como parámetros de entrada el dataframe de la matriz de confusión que se quiera graficar y colorear y, el nombre que se quiera asignar a la figura resultante (escrito entre comillas).

En ambas funciones, el primer paso realizado es definir la gradación de color que se utiliza en cada caso; esta gradación se ha definido mediante la función “*get_cmap()*” de la librería “*matplotlib*”, la cual realiza un mapa de color. Esta se ha asignado en la función “*PlotMatOrangeBlue(normalize_confusion_matrix, nombre)*” como un vector de escala de 128 valores para la gradación naranja y 128 valores para la azul, asignándose para la función “*PlotMatViridis(normalize_confusion_matrix, nombre)*” un vector de escala de 256 valores para la gradación viridis.

Posteriormente, se realiza para ambas funciones un bucle “if”, donde si la matriz tiene una dimensión mayor o igual a 7x7 (filas x columnas), se creará una figura con un tamaño 15x10 y, si tiene un tamaño menor a 7x7, se creará una figura con un tamaño 8x4. Se ha estimado que estos tamaños son los más adecuados para una mejor visualización.

Por último, se usa la función “*heatmap()*” de la librería “*seaborn*” que crea un mapa de calor, al que se le asignan como argumentos la matriz que se va a graficar y la gradación de color usada para unos valores determinados. Tal y como se explica en el “II.II ANEXO II: Elección de gradación de color para graficar matrices”, se ha optado por utilizar una escala logarítmica para definir la gradación de color. La figura resultante, se guardará en formato PNG en la carpeta donde se ejecute el script que llama a esta función. Se ha realizado el diagrama de flujo (*Figura 20*) de la función “*PlotMatOrangeBlue(normalize_confusion_matrix, nombre)*” como ejemplo de este apartado.

Se puede encontrar un ejemplo del funcionamiento de estas funciones en el apartado “Prueba graficar y colorear matrices normalizadas”.

Tabla 5: Funciones para graficar y colorear matrices de confusión

```
##### IMPRIMIR MATRICES #####
#Función para imprimir las matrices de confusión normalizadas, utilizando un mapa de calor con diferentes gradaciones de color
#Gradaciones de color diferentes y con una escala logarítmica divididas en intervalos

#Escala de color Naranja - Azul
def PlotMatOrangeBlue(normalize_confusion_matrix, nombre):

    #Se define el color, en una escala naranja-azul
    topO = cm.get_cmap('Oranges_r', 128)
    #En una escala de colores, el naranja representa la mitad 128/256
    bottomB = cm.get_cmap('Blues', 128)
    #En una escala de colores, el naranja representa 128/256
    newcolorsOB = np.vstack((topO(np.linspace(0, 1, 128)),
                             bottomB(np.linspace(0, 1, 128))))
    newcmpOB = ListedColormap(newcolorsOB, name='OrangeBlue')

    if len(normalize_confusion_matrix) >= 7:
        plt.figure(figsize=(15, 10))
        #Mapa de calor, dependiendo la escala de colores que se quiera (escala creada u otra ya existente),
        #asigna la escala a cmap; en este caso, cmap = newcmpOB (escala Orange y Blue)
        sn.heatmap(normalize_confusion_matrix,           cmap=newcmpOB,
                   annot_kws={"size": 13},
                   norm=colors.LogNorm(vmin=0.01,vmax=1),      vmin=0.01,      vmax=1,
                   annot=True);
        plt.title("Mapa de calor Matriz de confusión - " + nombre)
        plt.tight_layout() #Ajusta la figura (funciona también sin este parámetro)
        plt.grid(which='minor', alpha=0.1) #Para hacer el grid de un tono más claro y no tape datos
        plt.grid(which='major', alpha=0.2) #Para hacer el grid de un tono más claro y no tape datos
        plt.savefig(nombre) #Guarda la figura creada en la carpeta donde se encuentre el fichero de ejecución
```

```

else:
    plt.figure(figsize=(8, 4))
        #Mapa de calor, dependiendo la escala de colores que se quiera (escala creada
u otra ya existente),
        # asigna la escala a cmap; en este caso, cmap = newcmpOB (escala Orange y
Blue)
        sn.heatmap(normalize_confusion_matrix,           cmap=           newcmpOB,
annot_kws={"size": 20},
           norm=colors.LogNorm(vmin=0.01,vmax=1),           vmin=0.01,           vmax=1,
annot=True);
        plt.title("Mapa de calor Matriz de confusion - " + nombre)
        plt.tight_layout() #Ajusta la figura (funciona también sin este parámetro)
        plt.grid(which='minor', alpha=0.1) #Para hacer el grid de un tono más claro y no
tape datos
        plt.grid(which='major', alpha=0.2) #Para hacer el grid de un tono más claro y no
tape datos
        plt.savefig(nombre) #Guarda la figura creada en la carpeta donde se encuentre
el fichero de ejecución

return plt.show()

#Escala de color Viridis
def PlotMatViridis(normalize_confusion_matrix, nombre):

    #Se define el color Viridis con una escala de 256 datos
    viridis = cm.get_cmap('viridis', 256)

    if len(normalize_confusion_matrix) >= 7:
        plt.figure(figsize=(15, 10))
            #Mapa de calor, dependiendo la escala de colores que se quiera (escala creada
u otra ya existente),
            #asigna la escala a cmap; en este caso, cmap = newcmpOB (escala Orange y
Blue)
            sn.heatmap(normalize_confusion_matrix,  cmap=  viridis,  annot_kws={"size":
13},
               norm=colors.LogNorm(vmin=0.01,vmax=1),           vmin=0.01,           vmax=1,
annot=True);

```

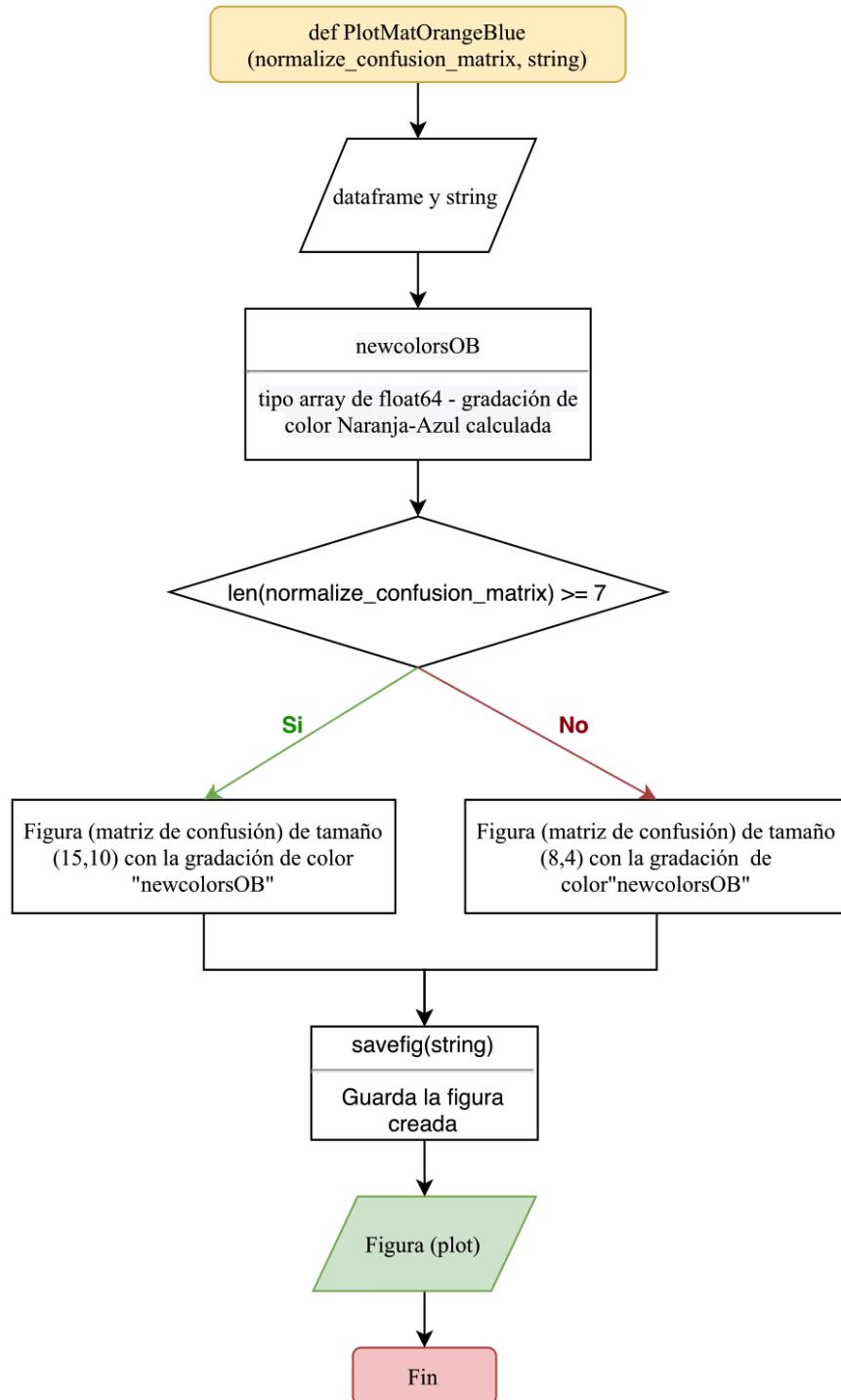
```

plt.title("Mapa de calor Matriz de confusión - " + nombre)
plt.tight_layout() #Ajusta la figura (funciona también sin este parametro)
plt.grid(which='minor', alpha=0.1) #Para hacer el grid de un tono más claro y no
tape datos
plt.grid(which='major', alpha=0.2) #Para hacer el grid de un tono más claro y no
tape datos
plt.savefig(nombre) #Guarda la figura creada en la carpeta donde se encuentre
el fichero de ejecución

else:
    plt.figure(figsize=(8, 4))
    #Mapa de calor, dependiendo la escala de colores que se quiera (escala creada
u otra ya existente),
    #asigna la escala a cmap; en este caso, cmap = newcmpOB (escala Orange y
Blue)
    sn.heatmap(normalize_confusion_matrix, cmap= viridis, annot_kws={"size":
20},
                norm=colors.LogNorm(vmin=0.01,vmax=1),           vmin=0.01,           vmax=1,
annot=True);
    plt.title("Mapa de calor Matriz de confusión - " + nombre)
    plt.tight_layout() #Ajusta la figura (funciona también sin este parametro)
    plt.grid(which='minor', alpha=0.1) #Para hacer el grid de un tono más claro
    plt.grid(which='major', alpha=0.2) #Para hacer el grid de un tono más claro
    plt.savefig(nombre) #Guarda la figura creada en la carpeta donde se encuentre
el fichero de ejecución

return plt.show()

```



*Figura 20: Diagrama de flujo de la función
`"PlotMatOrangeBlue(normalize_confusion_matrix, nombre)"`*

4.6 Función para transformar las matrices de confusión totales en submatrices de tamaño 2x2

Tal y como se explica en el apartado “*Matriz de confusión*”, una matriz de confusión con múltiples variables MxM, se puede transformar en M matrices 2x2. Cada una de estas matrices reducidas, corresponderían a una letra en particular. Esta forma de representar las letras contando con los datos VP, FP, FN y VP en la matriz 2x2, permite evaluar el acierto en el RAH del software para una letra concreta de una manera más sencilla, sabiendo además el tipo de errores que se cometan en el reconocimiento y pudiendo aplicar también los parámetros estadísticos de evaluación para matrices de confusión de este tamaño.

Se ha creado la función “*Matriz2x2(confusion_matrix)*” (*Tabla 6*) con la que se puede obtener una lista de las matrices de confusión 2x2 para cada letra; recibe como parámetro de entrada la matriz de confusión MxM que se quiera transformar.

Un ejemplo del funcionamiento de esta función se encuentra en el apartado “*Prueba para de la función de transformación de matriz de confusión MxM en 2x2*”.

Tabla 6: Función para transformar matrices de confusión grandes en submatrices 2x2

```
##### TRANSFORMACION A MATRICES 2X2 #####
#VOCALES
def Matriz2x2 (confusion_matrix):
    #Definición de qué es un verdadero positivo, falso positivo, falso negativo y
    #verdadero negativo
    #Definición de verdaderos positivos de la matriz
    verdaderos_positivos = np.diag(confusion_matrix)
    #Definicion de los falsos positivos de la matriz
    falsos_positivos = confusion_matrix.sum(axis=0) - verdaderos_positivos
    #Definicion de los falsos negativos de la matriz
    falsos_negativos = confusion_matrix.sum(axis=1) - verdaderos_positivos
    #Definicion de los verdaderos negativos de la matriz
    verdaderos_negativos = (confusion_matrix.to_numpy().sum() -
    (verdaderos_positivos + falsos_positivos + falsos_negativos))
```

```

#Lista con Dataframe de matrices de confusión 2x2
Matrices2x2 = []
#Creación de una lista (de dataframes) en la que se guardan la matriz de confusión
de cada letra
for dato, vp, fp, vn, fn in zip(
    confusion_matrix.columns,
    verdaderos_positivos, falsos_positivos,
    verdaderos_negativos, falsos_negativos):

    #Creación del Dataframe de la matriz de confusión 2x2 para cada letra,
    #cuenta con una tercera fila con el nombre de la letra, para poder identificar el
df
frame = pd.DataFrame.from_dict(
    {"Positive(1)": (vp, fp), "Negative(0)": (fn, vn), "Name": (dato, "-")},
    orient="Index",
    columns=("Positive(1)", "Negative(0)")
)
frame.index.name = f'Predicted Values for "{dato}"'
frame.columns.name = f'Actual values for "{dato}"'
Matrices2x2.append(frame) #Se va rellenando la lista con los dataframes
obtenidos
return Matrices2x2

```

El diagrama de flujo asociado a esta función se encuentra en la *Figura 21*. En primer lugar, lee el dataframe correspondiente a la matriz de confusión que se quiera simplificar; posteriormente, se definen dónde se encuentran los valores de la matriz correspondientes a VP, FP, FN y VN y se guardan en variables de tipo series; también se crea una lista (“Matrices2x2”) que almacenará las nuevas matrices creadas para cada letra. Para llenar esta letra, se crea un bucle “for” que recorre el tamaño de las variables tipo series y crea la nueva matriz 2x2 en una variable de tipo dataframe (“frame”), por último, se añade este dataframe correspondiente a la matriz de confusión a la lista Matrices2x2. La función devuelve la lista de matrices 2x2.

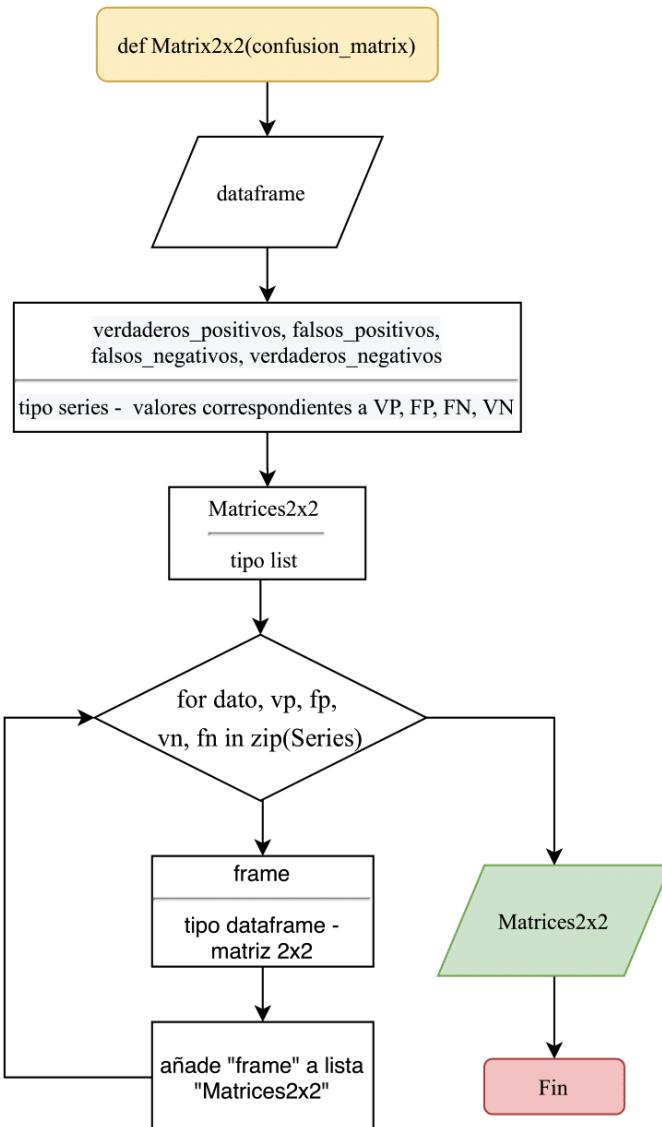


Figura 21: Diagrama de flujo de la función "Matrix2x2(confusion_matrix)"

4.7 Funciones estadísticas

En este apartado, se describirán las funciones realizadas para calcular cada parámetro estadístico en matrices de confusión 2x2 (para cada letra de manera individual). Se encuentra una descripción más detallada de los parámetros estadísticos utilizados en el apartado “Fundamentos”.

Estas funciones, en general, no serán usadas de manera individual, sino que serán llamadas por otras funciones que calculen estos parámetros para el dataframe de una

matriz de confusión determinada y generen diferentes tablas (funciones creadas en los apartados 4.8 y 4.9). Aun así, estas funciones también podrán ser utilizadas de manera aislada pasándoles como parámetros de entrada los VP, FN, FP y VN de la matriz de la letra que se quiera analizar.

4.7.1 Sensibilidad

Para calcular la Sensibilidad de la matriz de confusión para una letra concreta se utiliza la “*Ecuación I*”, desarrollándose de esta manera la función “*sensibilidad(VP, FN, FP, VN)*” (*Tabla 7*).

Se pueden ver ejemplos de los resultados que devuelve esta función en el apartado “*Prueba del resultado de la tabla de funciones estadísticas*” y en la “*Figura 36*”.

Tabla 7: Función para realizar el cálculo de la Sensibilidad

```
#Definición de la función que calcula la sensibilidad de la muestra/matriz 2x2
def sensibilidad(VP, FN, FP, VN):
    #Cálculo de la sensibilidad, redondeo 3 decimales
    resultado_sen = round((VP)/(VP + FN), 3);
    return (resultado_sen)
```

Se ha realizado el diagrama de flujo de la función “*sensibilidad(VP, FN, FP, VN)*” como ejemplo de este apartado, se encuentran en la *Figura 22*. En primer lugar, la función recibe como parámetros de entrada cuatro enteros (tipo int) correspondientes a los valores de VP, FN, FP y VN de la letra que se quiera analizar. Posteriormente, se calcula el resultado la sensibilidad del RAH de la letra a analizar mediante la “*Ecuación I*”, redondeado a 3 decimales, y se guarda en una variable de tipo float llamada “*resultado_sen*”. La función devuelve esta última variable.

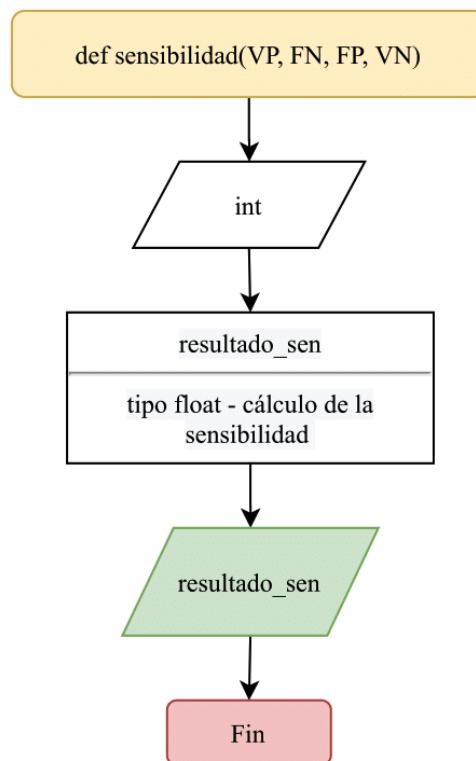


Figura 22: Diagrama de flujo de la función "sensibilidad(VP, FN, FP, VN)"

4.7.2 Especificidad

Para calcular la Especificidad de la matriz de confusión para una letra concreta se utiliza la “Ecuación 2”, desarrollándose de esta manera la función “*especificidad(VP, FN, FP, VN)*” (Tabla 8). Devuelve como resultado del cálculo el valor numérico obtenido redondeado a tres cifras decimales. Se pueden ver ejemplos de los resultados que devuelve esta función en el apartado “*Prueba del resultado de la tabla de funciones estadísticas*” y en la “*Figura 37*”.

Tabla 8: Función para realizar el cálculo de la Especificidad

```

#Definición de la función que calcula la especificidad de la muestra/matriz 2x2
def especificidad(VP, FN, FP, VN):
    #Cálculo de la especificidad, redondeo 3 decimales
    resultado_espec = round((VN)/(VN + FP), 3);
    return (resultado_espec)

```

4.7.3 Exactitud

Para calcular la Exactitud de la matriz de confusión para una letra concreta se utiliza la “Ecuación 3”, desarrollándose de esta manera la función “*exactitud(VP, FN, FP, VN)*” (*Tabla 9*). Devuelve como resultado del cálculo el valor numérico obtenido redondeado a tres cifras decimales. Se pueden ver ejemplos de los resultados que devuelve esta función en el apartado “*Prueba del resultado de la tabla de funciones estadísticas*” y en la “*Figura 38*”.

Tabla 9: Función para realizar el cálculo de la Exactitud

```
#Definición de la función que calcula la exactitud de la muestra/matriz 2x2
def exactitud(VP, FN, FP, VN):
    #Cálculo de la exactitud, redondeo 3 decimales
    resultado_exact = round((VP + VN)/(VP + FN + FP + VN), 3);
    return (resultado_exact)
```

4.7.4 Precisión

Para calcular la Precisión de la matriz de confusión para una letra concreta se utiliza la “Ecuación 4”, desarrollándose de esta manera la función “*precision(VP, FN, FP, VN)*” (*Tabla 10*). Devuelve como resultado del cálculo el valor numérico obtenido redondeado a tres cifras decimales. Se pueden ver ejemplos de los resultados que devuelve esta función en el apartado “*Prueba del resultado de la tabla de funciones estadísticas*” y en la “*Figura 39*”.

Tabla 10: Función para realizar el cálculo de la Precisión

```
#Definición de la función que calcula la precisión de la muestra/matriz 2x2
def precision(VP, FN, FP, VN):
    #Cálculo de la especificidad, redondeo 3 decimales
    resultado_precis = round((VP)/(VP + FP), 3);
    return (resultado_precis)
```

4.7.5 Error Medio

Para calcular el Error Medio de la matriz de confusión para una letra concreta se utiliza la “Ecuación 5”, desarrollándose de esta manera la función “*errorMedio(VP, FN, FP, VN)*” (*Tabla 11*). Devuelve como resultado del cálculo el valor numérico obtenido redondeado a tres cifras decimales. Se pueden ver ejemplos de los resultados que devuelve esta función en el apartado “*Prueba del resultado de la tabla de funciones estadísticas*” y en la “*Figura 40*”.

Tabla 11: Función para realizar el cálculo del Error Medio

```
#Definición de la función Misclassification Ratio or Mean error (MRC) - Error Medio
def errorMedio (VP, FN, FP, VN):
    #Fórmula error medio, redondeo 3 decimales
    MCR = round((FP + FN) / (VP + FN + FP + VN), 3);
    return (MCR)
```

4.7.6 Valor-F

Para calcular el Valor-F de la matriz de confusión para una letra concreta se utiliza la “Ecuación 6”, desarrollándose de esta manera la función “*valorF(VP, FN, FP, VN)*” (*Tabla 12*). Devuelve como resultado del cálculo el valor numérico obtenido redondeado a tres cifras decimales. Se pueden ver ejemplos de los resultados que devuelve esta función en el apartado “*Prueba del resultado de la tabla de funciones estadísticas*” y en la “*Figura 41*”.

Tabla 12: Función para realizar el cálculo del Valor-F

```
#Definición del F-score (Valor F) = Precisión x Sensibilidad
def valorF(VP, FN, FP, VN):
    #Cálculo del valor F, redondeo 3 decimales
    resultado_valorF = round(2*(sensibilidad(VP, FN, FP, VN) * precision(VP, FN, FP,
    VN))/
        (sensibilidad(VP, FN, FP, VN) + precision(VP, FN, FP, VN)), 3);
    return (resultado_valorF)
```

4.7.7 Ratio de Falsos Positivos

Para calcular el Ratio de Falsos Positivos de la matriz de confusión para una letra concreta se utiliza la “*Ecuación 7*”, desarrollándose de esta manera la función “*falsePositive(VP, FN, FP, VN)*” (*Tabla 13*). Devuelve como resultado del cálculo el valor numérico obtenido redondeado a tres cifras decimales. Se pueden ver ejemplos de los resultados que devuelve esta función en el apartado “*Prueba del resultado de la tabla de funciones estadísticas*” y en la “*Figura 42*”.

Tabla 13: Función para realizar el cálculo del Ratio de Falsos Positivos

```
#Definición del ratio de falsos positivos - false positive rate (FPR)
def falsePositive(VP, FN, FP, VN):
    #Cálculo FPR, redondeo 3 decimales
    resultado_FPR = round((FP)/(FP + VN), 3);
    return (resultado_FPR)
```

4.7.8 Coeficiente de correlación de Matthews

Para calcular el Coeficiente de correlación de Matthews de la matriz de confusión para una letra concreta se utiliza la “*Ecuación 8*”, desarrollándose de esta manera la función “*matthew(VP, FN, FP, VN)*” (*Tabla 14*). Devuelve como resultado del cálculo el valor numérico obtenido redondeado a tres cifras decimales. Se pueden ver ejemplos de los resultados que devuelve esta función en el apartado “*Prueba del resultado de la tabla de funciones estadísticas*” y en la “*Figura 43*”.

Tabla 14: Función para realizar el cálculo del Coeficiente de correlación de Matthews

```
#Definición de la función que calcula el Coeficiente de Matthew
def matthew(VP,FN, FP, VN):
    #Fórmula Coeficiente Matthew, redondeo a 3 decimales
    resultado_matthew = round(((VP * VN) - (FP * FN)) / (math.sqrt((VP + FP) * (VP
+ FN) * (VN + FP) * (VN + FN)))), 3);
    return (resultado_matthew)
```

4.7.9 Índice de Jaccard

Para calcular el Índice de Jaccard de la matriz de confusión para una letra concreta se utiliza la “*Ecuación 10*”, desarrollándose de esta manera la función “*jaccard(VP, FN, FP, VN)*” (*Tabla 15*). Devuelve como resultado del cálculo el valor numérico obtenido redondeado a tres cifras decimales. Se pueden ver ejemplos de los resultados que devuelve esta función en el apartado “*Prueba del resultado de la tabla de funciones estadísticas*” y en la “*Figura 44*”.

Tabla 15: Función para realizar el cálculo del Índice de Jaccard

```
#Definición de la función que calcula el Índice de Jaccard
def jaccard(VP, FN, FP, VN):
    #Fórmula Índice Jaccard, redondeo 3 decimales
    resultado_jaccard = round((VP) / (VP + FP + FN), 3);
    return (resultado_jaccard)
```

4.7.10 Parámetro D'

Para calcular el Parámetro D’ la matriz de confusión para una letra concreta se utiliza la “*Ecuación 11*”, desarrollándose de esta manera la función “*parametroD(VP, FN, FP, VN)*” (*Tabla 16*). Devuelve como resultado del cálculo, el valor numérico obtenido redondeado a tres cifras decimales. La función calcula en primer lugar las probabilidades condicionales de acierto y fallo posteriormente, aplica la función normal de distribución normal inversa para cada una de ellas (en Python la función “*norm.ppf()*” de la librería “*scipy.stats*”), por último, se calcula la diferencia entre ambos resultados obteniéndose un resultado comprendido entre $-\infty$ y $+\infty$.

Para realizar esta función, se han subdividido los pasos para el cálculo del Parámetro D’ y se han comprobado los posibles resultados que devolvería la función (*II.III ANEXO III: Comprobación de resultados del Parámetro D’*). Se pueden ver ejemplos de los resultados que devuelve esta función en el apartado “*Prueba del resultado de la tabla de funciones estadísticas*” y en la “*Figura 45*”.

Tabla 16: Función para realizar el cálculo del Parámetro D'

```
#Definimos la función que calcula el parámetro d'  
def parametroD(VP, FN, FP, VN):  
    #Definición de función para calcular el Parámetro D', redondeo a 3 decimales  
    H = VP/(VP+FN) #Probabilidad condicional de acierto  
    FA = FP/(FP+VN) #Probabilidad condicional de fallo  
    zH = norm.ppf(H) #Función normal de distribución inversa para la probabilidad de  
    acierto  
    zFA = norm.ppf(FA)#Función normal de distribución inversa para la probabilidad  
    de fallo  
    parametro_d = zH -zFA #Calculo del parámetro d'  
    return (round(parametro_d, 3))
```

4.8 Creación de tabla de funciones estadísticas

Para poder visualizar de manera conjunta todas las funciones estadísticas vistas en el apartado “*Funciones estadísticas*” para todas las letras de una matriz de confusión, se ha desarrollado la función “*FuncionesEstadisticas(confusion_matrix)*” (*Tabla 17*). Esta función recibe como parámetro de entrada la matriz de confusión del conjunto de letras de las que se quieran conocer los resultados de los parámetros estadísticos y devuelve un dataframe en forma de tabla estos resultados para cada letra.

Se puede encontrar un ejemplo del funcionamiento de esta función en el apartado “*Prueba del resultado de la tabla de funciones estadísticas*”.

Tabla 17: Función para imprimir una tabla con todos los parámetros estadísticos calculados para cada letra de una matriz de confusión de entrada

```
##### APlicacion DE PARÁMETROS ESTADÍSTICO A VOCALES Y  
CONSONANTES #####  
  
#Cálculos de las funciones estadísticas previamente definidas  
def FuncionesEstadisticas(confusion_matrix):  
    #Definición de VP, FP, FN y VN para cada letra de la matriz  
    verdaderos_positivos = np.diag(confusion_matrix)
```

```

falsos_positivos = confusion_matrix.sum(axis=0) - verdaderos_positivos
falsos_negativos = confusion_matrix.sum(axis=1) - verdaderos_positivos
verdaderos_negativos      =      (confusion_matrix.to_numpy().sum() -
(verdaderos_positivos + falsos_positivos + falsos_negativos))

#Lista con Dataframe de matrices de confusión 2x2
Matrices2x2 = []
for dato, vp, fp, vn, fn in zip(
    confusion_matrix.columns,
    verdaderos_positivos, falsos_positivos,
    verdaderos_negativos, falsos_negativos):

    #Creación del Dataframe de la matriz de confusión 2x2 para cada letra,
    # cuenta con una tercera fila con el nombre de la letra, para poder identificar el
df
frame = pd.DataFrame.from_dict(
    {"Positive(1)": (vp, fp), "Negative(0)": (fn, vn), "Name": (dato, "-")},
    orient="Index",
    columns=("Positive(1)", "Negative(0)")
)
frame.index.name = f'Predicted Values for "{dato}"'
frame.columns.name = f'Actual values for "{dato}"'
Matrices2x2.append(frame)

#Creación de listas para almacenar los cálculos de los datos
#Creación de una lista para almacenar los cálculos de la sensibilidad
lista_sensibilidad = []
#Creación de una lista para almacenar los datos de la especificidad de cada una
de las letras
lista_Especificidad = []
#Creación de una lista para almacenar los datos de la exactitud de cada una de
las letras
lista_Exactitud = []
#Creación de una lista para almacenar los datos de la precisión de cada una de
las letras
lista_Precision = []
#Creación de una lista para almacenar los cálculos del Coeficiente de Matthew
lista_Matthews = []
#Creación de una lista para almacenar los cálculos del Índice de Jaccard

```

```

lista_Jaccard = []
#Creación de una lista para almacenar los cálculos del Parámetro d'

lista_ParametroD = []
#Creación de una lista para almacenar los cálculos del error medio cada una de las letras

lista_ErrorMedio = []
#Creación de una lista para almacenar los cálculos del valor F cada una de las letras

lista_ValorF = []
#Creación de una lista para almacenar los cálculos del ratio de falsos positivos para letras

lista_FPR = []
#Creación de una lista, que va a contener los nombres de las letras, cuyas posiciones van a ser las mismas que ocupan en las demás listas

#Es decir, si se quiere saber qué letra es la posición 4 de una lista estadística, se va a la lista nombre, se mira la posición 4 y esa será la letra correspondiente

lista_Letra_Posicion = []

#Bucle para calcular los Parámetros Estadísticos para cada letra
for i in range(0,len(Matrices2x2)):

    VP = Matrices2x2[i]['Positive(1)'][0] #Definición de la posición en la que se encuentra el Verdadero Positivo
    FN = Matrices2x2[i]['Positive(1)'][1] #Definición de la posición en la que se encuentra el Verdadero Positivo
    FP = Matrices2x2[i]['Negative(0)'][0] #Definición de la posición en la que se encuentra el Verdadero Positivo
    VN = Matrices2x2[i]['Negative(0)'][1] #Definición de la posición en la que se encuentra el Verdadero Positivo

    Nombre_Letra = Matrices2x2[i]['Positive(1)'][2] #Sabe el nombre de la letra

    sensibilidadf = sensibilidad(VP, FN, FP, VN) #Cálculo de la sensibilidad
    especificidadf = especificidad(VP, FN, FP, VN) #Cálculo de la especificidad
    exactitudf = exactitud(VP, FN, FP, VN) #Cálculo de la exactitud
    precisionf = precision(VP, FN, FP, VN) #Cálculo de la precisión
    matthewf = matthew(VP, FN, FP, VN) #Cálculo del Coeficiente de Matthews
    jaccardf = jaccard(VP, FN, FP, VN) #Cálculo del Índice de Jaccard
    parametroDf = parametroD(VP, FN, FP, VN) #Cálculo del Parámetro d'

```

```

errorMedf = errorMedio(VP, FN, FP, VN) #Cálculo del error medio para las
Consonantes
    valorFf = valorF(VP, FN, FP, VN) #Cálculo del valor para las Consonantes
    fprf = falsePositive(VP, FN, FP, VN) #Cálculo del valor FPR para las
Consonantes

    lista_sensibilidad.append(sensibilidaddf) #Se rellena la lista de las sensibilidades
    lista_Especificidad.append(especificidaddf) #Se rellena la lista de la especificidad
    lista_Exactitud.append(exactitudf) #Se rellena la lista de la exactitud
    lista_Precision.append(precisionf) #Se rellena la lista de la precisión
    lista_Matthews.append(matthewf) #Se rellena la lista del coeficiente de
Matthews
    lista_Jaccard.append(jaccardf) #Se rellena la lista del Índice de Jaccard
    lista_ParametroD.append(parametroDf) #Se rellena la lista del Parámetro d'
    lista_ErrorMedio.append(errorMedf) #Se rellena la lista del error medio
    lista_ValorF.append(valorFf) #Se rellenas la lista del valor F
    lista_FPR.append(fprf) #Se rellena la lista del FPR
    lista_Letra_Posicion.append(Nombre_Letra) #Nombre de las letras iteradas

#Se transforma el índice de la matriz/dataframe numérico por un incide que refleja
la letra estudiada
Indices = lista_Letra_Posicion

#Se crea un dataframe con los resultados obtenidos para todos los parámetros
estadisticos calculados
Resultado_Estadistico = pd.DataFrame({"Sensibilidad": lista_sensibilidad,
"Especificidad": lista_Especificidad, "Exactitud": lista_Exactitud, "Precision": lista_Precision, "Error_Medio": lista_ErrorMedio, "Valor_F": lista_ValorF, "Ratio_FP": lista_FPR, "Coeficiente_Matthews": lista_Matthews, "Indice_Jaccard": lista_Jaccard, "Parametro_d": lista_ParametroD}, index = Indices)

#Se imprime la tabla (dataframe) con los resultados totales
print("\n Los resultados estadisticos son: \n \n", Resultado_Estadistico, "\n")

#La funcion devuelve el dataframe "Resultado_Estadistico", con los parámetros
estadísticos calculados
return Resultado_Estadistico

```

El diagrama de flujo asociado a esta función se encuentra en la *Figura 23*. Esta función tiene como parámetro de entrada la variable de tipo dataframe que contenga la matriz de confusión de las letras que se quieran analizar. En primer lugar, se define en qué posición dentro de este dataframe se encuentran los valores de VP, FP, FN y VN para cada letra y se almacenan en variables de tipo series (“verdaderos_positivos”, “falsos_positivos”, “falsos_negativos” y “verdaderos_negativos”). Posteriormente, se crea una lista llamada “Matrices2x2” donde se almacenarán los dataframes correspondientes a la matriz 2x2 generada para cada letra mediante el bucle “for” que recorre el tamaño de las variables series. Una vez obtenidas esta lista con las matrices 2x2 de las letras que se quieren analizar, se puede proceder al cálculo de los parámetros estadísticos para cada una de las letras (parámetros explicados en el apartado “*Fundamentos*”).

En primer lugar, se crean listas donde se almacenarán cada uno de los resultados estadísticos para cada letra (“lista_sensibilidad”, “lista_Especificidad”, “lista_Exactitud”, etc.); para llenar estas listas, se utiliza un bucle “for” que recorre todo el tamaño de la lista “Matrices2x2” (todas las letras a analizar). En primer lugar dentro del bucle, se define la posición que ocupa el valor de VP, FP, FN y VN para cada letra dentro de los dataframes contenidos en la lista recorrida, posteriormente, se calculan los resultados de las variables estadísticas mediante las funciones vistas en el apartado “*Funciones estadísticas*” para cada letra y se guardan en variables de tipo float (“sensibilidaddf”, “especificidaddf”, “exactituddf”, etc.) ; por último, se añaden los resultados de cada variable estadística correspondiente a cada letra a las listas creadas anteriormente para almacenar cada uno de los resultados; este proceso se repite para todas las letras.

Posteriormente se crea una lista llamada “Indices” que guarda el índice correspondiente a cada letra en formato alfabetico, así se podrá asignar estos índices al dataframe que se crea a continuación obteniendo dataframe más intuitivo (para referirse a una letra, el índice correspondiente es la letra en sí). Se crea una tabla con los resultados (dataframe) llamada “Resultado_Estadistico” que almacenará todos los resultados estadísticos obtenidos y guardados en las listas correspondientes a cada parámetro estadístico para todas las letras contenidas en la matriz de confusión analizada (asignando los índices guardados en la lista “Indices”). Por último, se imprime en pantalla la tabla creada. La función devuelve el dataframe “Resultado_Estadistico”.

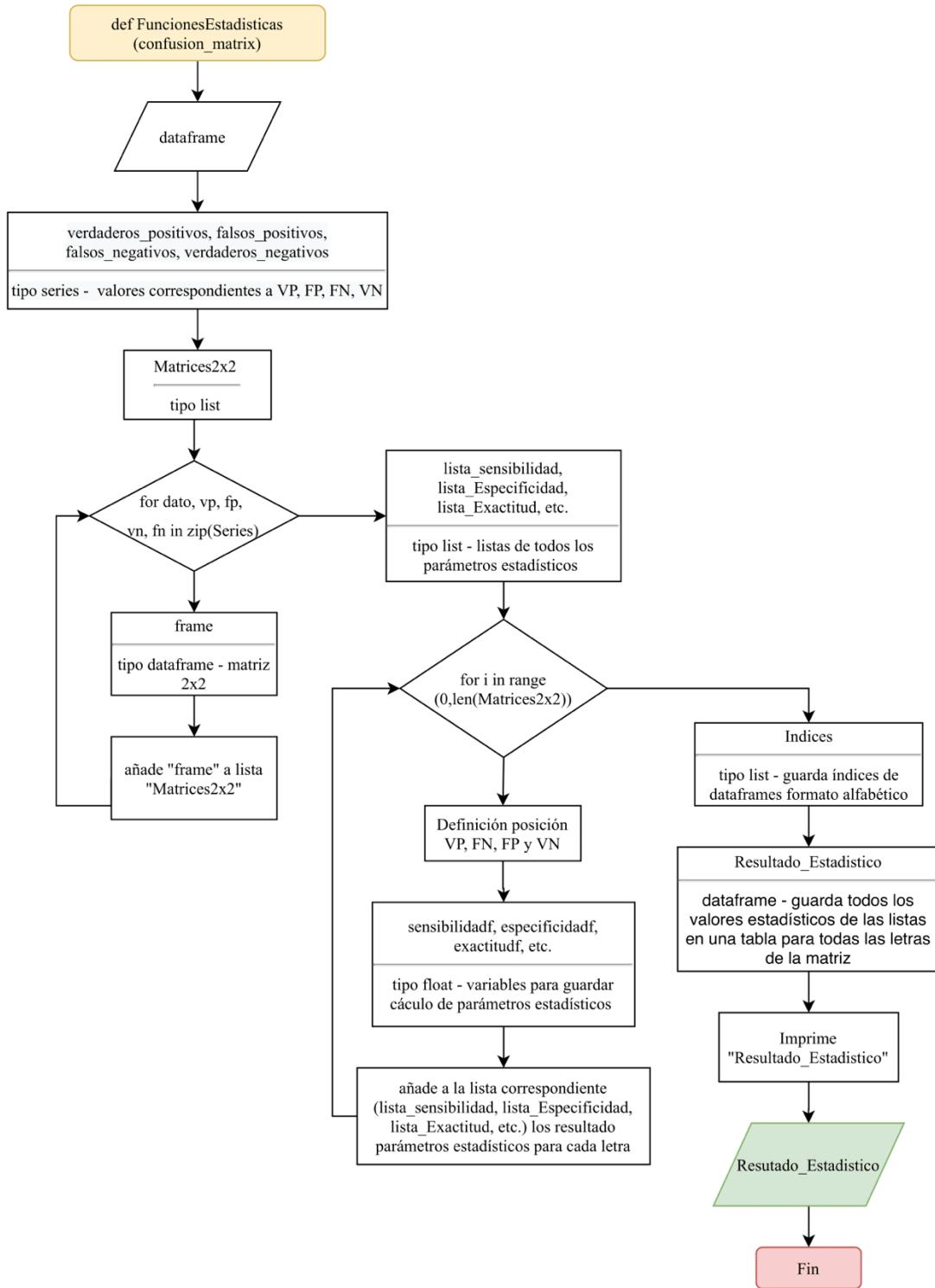


Figura 23: Diagrama de flujos de la función "FuncionesEstadisticas(confusion_matrix)"

4.9 Cálculo y creación de una tabla para las funciones estadísticas de manera individual en una matriz de confusión de letras

Con frecuencia, para diversos estudios que se puedan realizar, solo es necesario utilizar un parámetro estadístico concreto (o un número reducido), por ello se ha decidido crear funciones que devuelvan un parámetro concreto (*Tabla 18*). Devuelve como resultado un dataframe que conforma una tabla con el valor del parámetro estadístico concreto para cada letra de la matriz e imprime los resultados en pantalla.

Se encuentra un ejemplo del funcionamiento de estas funciones en el apartado “*Prueba del cálculo y creación de una tabla con funciones estadísticas de manera individual de una matriz de confusión de letras*”.

Tabla 18: Funciones para el cálculo de parámetros estadísticos concretos para una matriz de confusión de letras

```
##### PARAMETROS ESTADISTICOS INDIVIDUALES #####
#Sensibilidad
def EstatSensibilidad(matriz):
    sensib = pd.DataFrame(data = FuncionesEstadisticas(matriz).Sensibilidad);
    print("La Sensibilidad es: \n \n", sensib)
    return sensib

#Especificidad
def EstatEspecificidad(matriz):
    espec = pd.DataFrame(data = FuncionesEstadisticas(matriz).Especificidad);
    print("La Especificidad es: \n \n", espec)
    return espec

#Exactitud
def EstatExactitud(matriz):
    exact = pd.DataFrame(data = FuncionesEstadisticas(matriz).Exactitud);
    print("La Exactitud es: \n \n", exact)
    return exact
```

```

#Precisión
def EstatPrecision(matriz):
    precis= pd.DataFrame(data = FuncionesEstadisticas(matriz).Precision);
    print("La Precision es: \n \n", precis)
    return precis

#Error Medio
def EstatError_Medio(matriz):
    errorm = pd.DataFrame(data = FuncionesEstadisticas(matriz).Error_Medio);
    print("El Error Medio es: \n \n", errorm)
    return errorm

#Valor F
def EstatValor_F(matriz):
    vf = pd.DataFrame(data = FuncionesEstadisticas(matriz).Valor_F);
    print("El Valor-F es: \n \n", vf)
    return vf

#Ratio de Falsos Positivos
def EstatRatio_FP(matriz):
    ratFP = pd.DataFrame(data = FuncionesEstadisticas(matriz).Ratio_FP);
    print("El Ratio de Falsos Positivos es: \n \n", ratFP)
    return ratFP

#Coeficiente de Matthews
def EstatMatthews(matriz):
    matt = pd.DataFrame(data = FuncionesEstadisticas(matriz).Coeficiente_Matthews);
    print("El Coeficiente de Matthews es: \n \n", matt)
    return matt

#Índice de Jaccard
def EstatJaccard(matriz):
    jacc = pd.DataFrame(data = FuncionesEstadisticas(matriz).Indice_Jaccard);
    print("El Indice de Jaccard es: \n \n", jacc)
    return jacc

#Parámetro D'

```

```

def EstatParamD(matriz):
    parD = pd.DataFrame(data = FuncionesEstadisticas(matriz).Parametro_d);
    print("El Parametro D es: \n \n", parD)
    return parD

```

Se ha realizado el diagrama de flujo de la función “*EstatSensibilidad(matriz)*” como ejemplo de este apartado, se encuentran en la *Figura 24*. Como variable de entrada, esta función recibe la matriz de confusión que se quiera analizar (tipo dataframe). Para calcular la sensibilidad de todas las letras de la matriz, llama a la función “*FuncionesEstadisticas(confusion_matrix)*” y obtiene los datos correspondientes al parámetro estadístico requerido de la matriz a analizar, guardando el resultado en la variable llamada “*sensib*”, de tipo dataframe. Por último, se imprime en pantalla el resultado de las sensibilidades para cada una de las letras de la matriz de confusión analizada y se devuelve como resultado el dataframe “*sensib*”.

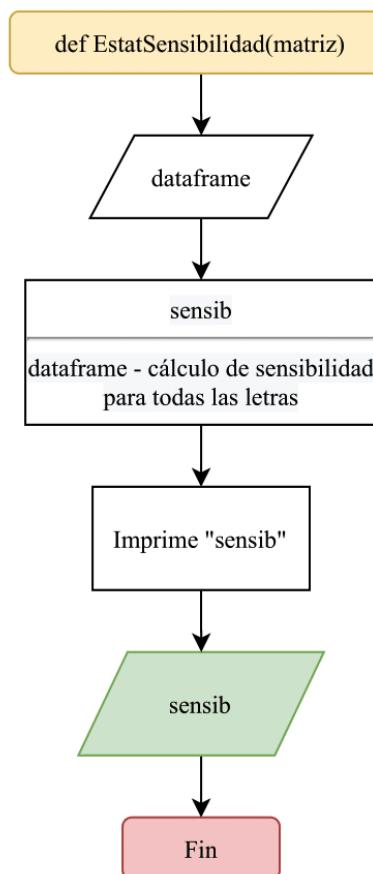


Figura 24: Diagrama de flujo de la función "EstatSensibilidad(matriz)"

4.10 Función para guardar dataframe en formato .xlsx

Para guardar los dataframes con los resultados obtenidos en formato “.xlsx” se ha creado la función “*dfToExcel(dataframe, nombre”*) (*Tabla 19*). Esta función recibe como parámetros de entrada el dataframe que se quiera guardar y el nombre que se desee asignar al archivo resultante, un string (el nombre debe estar escrito entrecomillado, y con la extensión “.xlsx”; por ejemplo, si se quiere nombrar al archivo como “Dataframe_Vocales”, se deberá escribir “Dataframe_Vocales.xlsx”) para que la función se ejecute correctamente.

El archivo obtenido se guardará en la misma carpeta donde se ejecute (y esté guardado) el script de Python que utiliza esta función.

Tabla 19: Función para guardar los dataframes obtenidos en formato .xlsx

```
#Función para guardar los dataframes obtenidos en formato .xlsx
def dfToExcel(dataframe, nombre):
    dataframe.to_excel(nombre)
```

El diagrama de flujo asociado a esta función se encuentra en la *Figura 25*, la función utiliza la función predefinida “to_excel” para guardar el dataframe dado en formato “.xlsx”

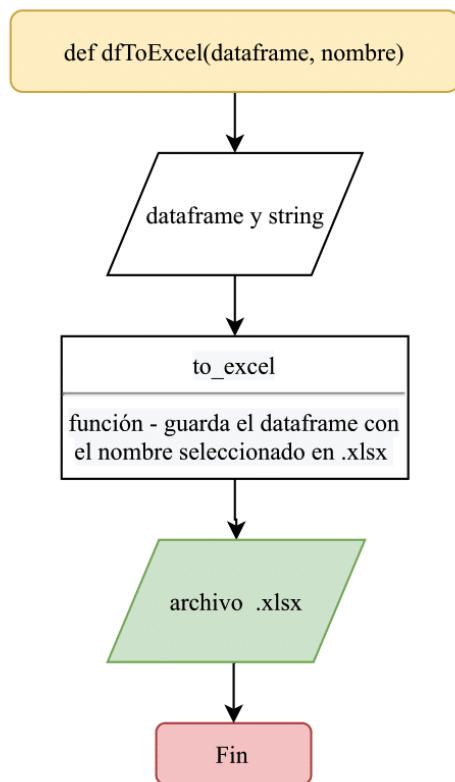


Figura 25: Diagrama de flujo de la función "dfToExcel(dataframe, nombre)"

4.11 Subir librería a GitHub

Para completar el desarrollo, se ha subido a la plataforma GitHub vista en el apartado “*Git: Sistema de control de versiones*” tanto la librería creada (“*Lib_RAH_DGG.py*”), como los ficheros de prueba (“*Prueba.Lib.DGG.py*” y “*salida.txt*”), el archivo “*README.md*” explicativo de la librería y una carpeta con imágenes de los resultados que se explican en “*README.md*” (“*Imagenes_Ejemplo_Resultados_Librería*”). El enlace al repositorio donde se encuentran estos ficheros es:

https://github.com/davidgg97/Python_Statistics_Library_for_Kaldi_RAH_DGG

5. Prueba de las funciones

Las pruebas para comprobar el correcto funcionamiento de la librería se realizarán con el archivo devuelto al ejecutar el script “*result_format.py*” de la receta ASICA para audio del sujeto CL001, llamado “*salida.txt*”. Para ello se ha creado un script llamado “*Prueba.Lib.DGG.py*”, en el que se ejecutan las pruebas que llaman a las funciones contenidas en la librería creada.

5.1 Prueba del porcentaje de sílabas reconocidas correctamente

Al ejecutar la función “*PorcentajeAciertos (df)*” de la librería con el dataframe “*texto_kaldi*” como entrada, se obtienen los resultados de la *Figura 26*. El resultado de sílabas reconocidas correctamente es del 85,04% para el total de palabras (general); un 84,21% para palabras de dos sílabas (las cuales representan un 35,98% respecto al total); un 84,78% para palabras de tres sílabas (las cuales representan un 26,14% respecto al total); y un 86,00% para palabras de cuatro sílabas (las cuales representan un 37,88% respecto al total). Se consideran correctos los cálculos.

El dataframe “*texto_kaldi*” es el creado en el apartado “*Conversión de datos*” al transformar el archivo “*salida.txt*”.

```
In [96]: ld.PorcentajeAciertos(texto_kaldi)
El porcentaje total correcto es: 85.04 %

El porcentaje correcto con 2 sílabas es: 84.21 % Con un porcentaje de palabras total de: 35.98 %
El porcentaje correcto con 3 sílabas es: 84.78 % Con un porcentaje de palabras total de: 26.14 %
El porcentaje correcto con 4 sílabas es: 86.0 % Con un porcentaje de palabras total de: 37.88 %
```

Figura 26: Resultado del porcentaje de sílabas reconocidas correctamente para "salida.txt"

5.2 Prueba de matrices de confusión

Para reducir la extensión de las pruebas para matrices de confusión y, puesto que la prueba tiene características similares repetitivas para todas las matrices, solo se considerará en este documento la prueba de obtención de la matriz de confusión para las consonantes.

En primer lugar, se ejecuta la función “*ConfusionMatrixCons(df)*” de la librería con el dataframe “*texto_kaldi*” como entrada para obtener la matriz de confusión de las consonantes, los resultados se visualizan en la *Figura 27*. Se observa que la función ha creado la matriz de manera correcta, enfrentando los resultados obtenidos (valores predichos) en las filas con los resultados conocidos (valores observados) en las columnas y teniendo en la diagonal principal la mayoría de los valores (aciertos).

Target	b	ch	d	f	g	k	l	m	n	ny	p	r	rr	s	t	x	y	z
b	38	0	3	0	1	0	0	0	0	0	1	0	0	0	0	0	0	
ch	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
d	0	0	30	0	1	0	0	0	0	0	0	3	1	0	1	0	0	
f	0	0	0	21	0	0	0	0	0	0	0	0	0	0	0	0	9	
g	1	0	0	0	13	0	1	0	0	0	0	0	0	0	0	0	0	
k	0	0	0	0	0	43	0	0	0	0	0	0	0	0	0	0	0	
l	0	0	1	0	0	0	41	0	1	0	0	3	0	0	0	0	0	
m	1	0	0	0	1	0	0	43	1	0	0	0	1	0	0	0	0	
n	0	0	3	0	0	0	3	7	19	1	0	5	1	0	0	0	0	
ny	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	
p	0	0	0	0	0	0	0	0	0	0	33	0	0	0	2	0	0	
r	0	0	0	0	0	0	1	0	0	0	0	16	1	0	1	0	0	
rr	0	0	0	0	0	0	0	0	0	0	0	0	13	0	0	0	0	
s	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	0	0	
t	0	0	0	0	0	4	0	0	0	0	0	0	0	46	0	0	3	
x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	0	1	
y	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	23	0	
z	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	25	

Figura 27: Resultado matriz de confusión de consonantes para "salida.txt"

En segundo lugar, se ejecuta la función “*NormalConfusionMatrixCons(df)*” de la librería con el dataframe “*texto_kaldi*” como entrada para obtener la matriz de confusión normalizada de las consonantes, los resultados se visualizan en la *Figura 28*. Se observa que la función ha creado la matriz normalizada de manera correcta, obteniendo los datos de la matriz normalizados y redondeados a dos decimales.

Target	b	ch	d	f	g	k	l	m	n	ny	p	r	rr	s	t	x	y	z
b	0.95	0	0.08	0	0.06	0	0	0	0	0	0.03	0	0	0	0	0	0	
ch	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
d	0	0	0.81	0	0.06	0	0	0	0	0	0	0.11	0.06	0	0.02	0	0	
f	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0.24	
g	0.02	0	0	0	0.76	0	0.02	0	0	0	0	0	0	0	0	0	0	
k	0	0	0	0	0	0.91	0	0	0	0	0	0	0	0	0	0	0	
l	0	0	0.03	0	0	0	0.85	0	0.05	0	0	0.11	0	0	0	0	0	
m	0.02	0	0	0	0.06	0	0	0.86	0.05	0	0	0	0.06	0	0	0	0	
n	0	0	0.08	0	0	0	0.06	0.14	0.9	0.12	0	0.18	0.06	0	0	0	0	
ny	0	0	0	0	0	0	0	0	0	0.88	0	0	0	0	0	0	0	
p	0	0	0	0	0	0	0	0	0	0	0.97	0	0	0	0.04	0	0	
r	0	0	0	0	0	0	0.02	0	0	0	0	0.57	0.06	0	0.02	0	0	
rr	0	0	0	0	0	0	0	0	0	0	0	0	0.72	0	0	0	0	
s	0	0	0	0	0	0	0	0	0	0	0	0	0.89	0	0	0	0	
t	0	0	0	0	0	0.09	0	0	0	0	0	0	0	0	0.92	0	0.08	
x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0.03	
y	0	0	0	0	0	0	0.02	0	0	0	0	0.04	0	0	0	1	0	
z	0	0	0	0	0	0	0	0	0	0	0	0	0.11	0	0	0	0.66	

Figura 28: Resultado matriz de confusión normalizada de consonantes para "salida.txt"

5.3 Prueba de agrupación de consonantes

Para reducir la extensión de las pruebas para la agrupación de consonantes y, puesto que la prueba tiene características similares repetitivas para todas las matrices, solo se considerará en este documento la prueba de obtención de la matriz de confusión y matriz de confusión normalizada para las consonantes sonoras.

Primeramente, se ejecuta la función “*CSonora(df)*” de la librería con el dataframe “*texto_kaldi*” como entrada, se obtienen los resultados de la *Figura 29*. Se observa que la función agrupa correctamente las consonantes sonoras, realizando un dataframe con la matriz de confusión para esta agrupación.

Target	b	d	g	l	m	n	ny	r	rr	y
b	38	3	1	0	0	0	0	0	0	0
d	0	30	1	0	0	0	0	3	1	0
g	1	0	13	1	0	0	0	0	0	0
l	0	1	0	41	0	1	0	3	0	0
m	1	0	1	0	43	1	0	0	1	0
n	0	3	0	3	7	19	1	5	1	0
ny	0	0	0	0	0	0	7	0	0	0
r	0	0	0	1	0	0	0	16	1	0
rr	0	0	0	0	0	0	0	0	13	0
y	0	0	0	1	0	0	0	1	0	23

Figura 29: Resultado matriz de confusión de consonantes sonoras para "salida.txt"

Segundamente, se ejecuta la función “*CSonora(df)*” de la librería con el dataframe “*texto_kaldi*” como entrada, se obtienen los resultados de la Figura 30. Donde se obtiene una matriz resultante similar a la Figura 29, pero con los datos normalizados.

Target	b	d	g	l	m	n	ny	r	rr	y
b	0.95	0.08	0.06	0	0	0	0	0	0	0
d	0	0.81	0.06	0	0	0	0	0.11	0.06	0
g	0.02	0	0.76	0.02	0	0	0	0	0	0
l	0	0.03	0	0.85	0	0.05	0	0.11	0	0
m	0.02	0	0.06	0	0.86	0.05	0	0	0.06	0
n	0	0.08	0	0.06	0.14	0.9	0.12	0.18	0.06	0
ny	0	0	0	0	0	0	0.88	0	0	0
r	0	0	0	0.02	0	0	0	0.57	0.06	0
rr	0	0	0	0	0	0	0	0	0.72	0
y	0	0	0	0.02	0	0	0	0.04	0	1

Figura 30: Resultado matriz de confusión normalizada de consonantes sonoras para "salida.txt"

5.4 Prueba graficar y colorear matrices normalizadas

De igual manera que en los dos apartados anteriores, se utilizará la matriz de confusión de las consonantes obtenidas en el archivo “salida.txt”.

En primer lugar, se ejecuta la función con gradación naranja.azul “*PlotMatOrangeBlue* (*Normal_Matriz_Confusion_C*, “Consonantes”), donde se tiene como parámetros de entrada el dataframe para las consonantes calculado en el apartado “Prueba de matrices de confusión”, y un string con el nombre asignado a esta figura (“Consonantes”). Tras la ejecución de esta función se obtiene la *Figura 31*.

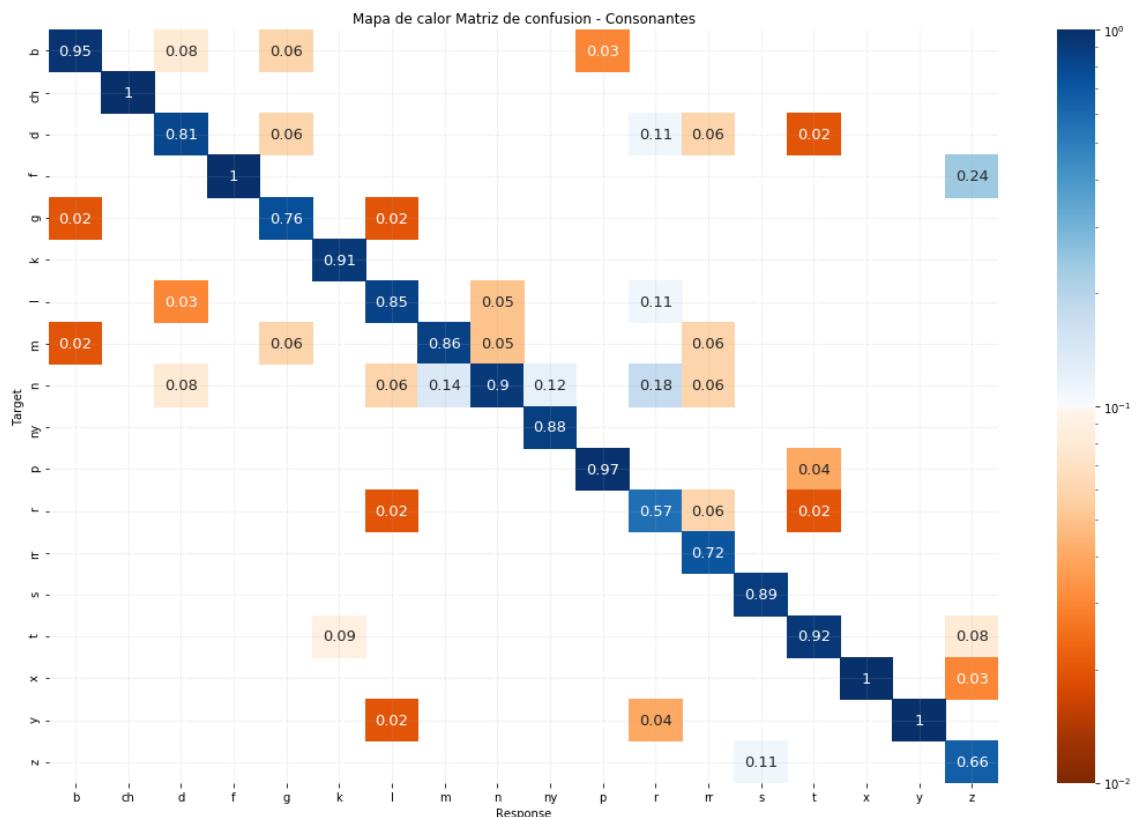


Figura 31: Matriz de confusión coloreada en Naranja-Azul para las consonantes

En segundo lugar, se ejecuta la función con gradación viridis “*PlotMatViridis* (*Normal_Matriz_Confusion_C*, “Consonantes”), donde se tiene como parámetros de entrada el dataframe para las consonantes calculado en el apartado “Prueba de matrices de confusión”, y un string con el nombre asignado a esta figura (“Consonantes”). Tras la ejecución de esta función, se obtiene como resultado la *Figura 32*.

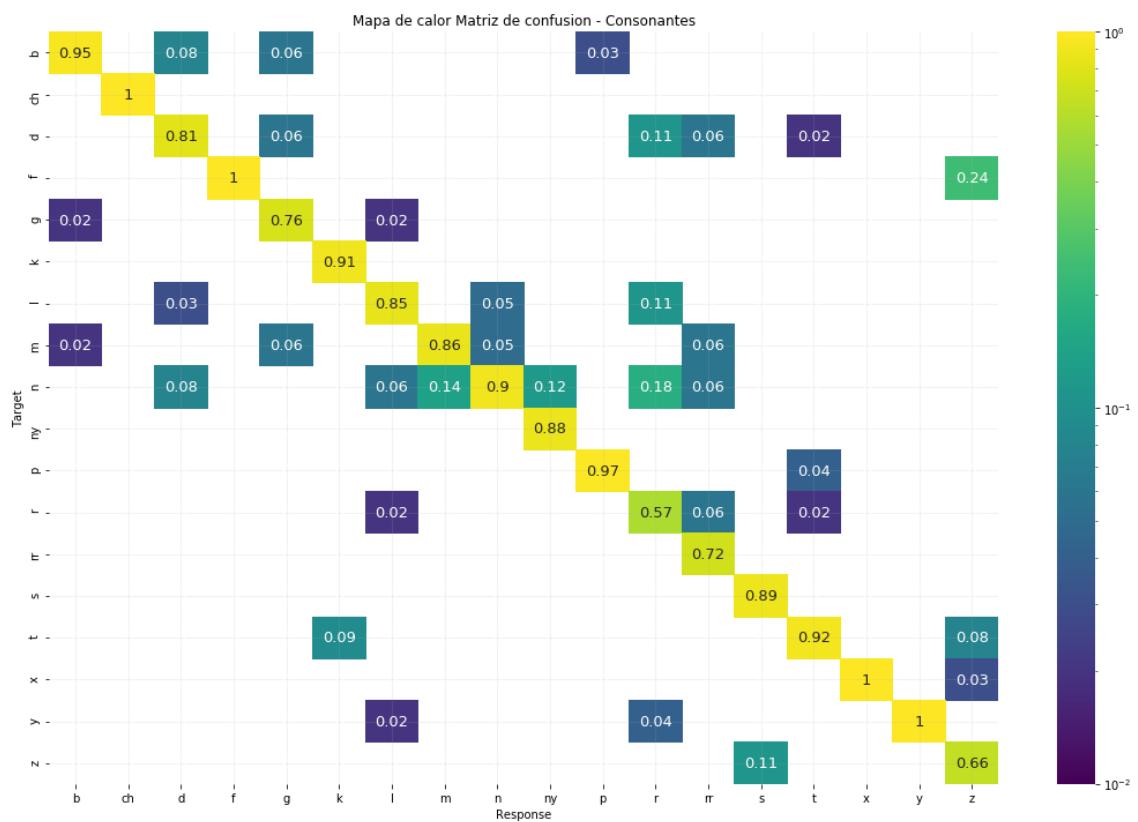


Figura 32: Matriz de confusión coloreada en Viridis para las consonantes

5.5 Prueba para la función de transformación de matriz de confusión MxM en 2x2

En este caso y, puesto que la función realiza todas las matrices 2x2 de la misma manera, se ha elegido transformar la matriz de confusión de las vocales de una prueba, debido a que ocupa una menor extensión, con lo que se simplifica la visualización de la prueba.

En la Figura 33, se observa la matriz de confusión para las vocales; al aplicar la función “*Matriz2x2(Matriz_Confusion_V)*”, se obtiene una lista con todas las matrices 2x2 para cada letra, se imprimen en pantalla para poder visualizar estas matrices (Figura 34).

Index	a	e	i	o	u
a	157	2	0	2	2
e	2	106	2	3	0
i	4	3	85	6	5
o	5	5	9	122	3
u	6	1	1	2	67

Figura 33: Matriz de confusión Vocales

```
In [66]: print(Matrices2x2_Vocales)
[Actual values for "a" Positive(1) Negative(0)
Predicted Values for "a"
Positive(1)          157      17
Negative(0)          6        420
Name                a        -, Actual values for "e" Positive(1) Negative(0)
Predicted Values for "e"
Positive(1)          106      11
Negative(0)          7        476
Name                e        -, Actual values for "i" Positive(1) Negative(0)
Predicted Values for "i"
Positive(1)          85       12
Negative(0)          18       485
Name                i        -, Actual values for "o" Positive(1) Negative(0)
Predicted Values for "o"
Positive(1)          122      13
Negative(0)          22       443
Name                o        -, Actual values for "u" Positive(1) Negative(0)
Predicted Values for "u"
Positive(1)          67       10
Negative(0)          10      513
Name                u        -]
```

Figura 34: Matrices de confusión 2x2 para Vocales

5.6 Prueba del resultado de la tabla de funciones estadísticas

Para realizar la prueba de la función “*FuncionesEstadisticas(confusion_matrix)*” para crear una tabla con todas las variables estadísticas de manera conjunta, vista en el apartado “*Creación de tabla de funciones estadísticas*”, se utilizará como variable de entrada un dataframe que contiene todas las consonantes recogidas en el archivo “salida.txt”. Como resultado de la ejecución de esta función, se obtiene la tabla con el resultado de las funciones estadísticas para las consonantes, se muestra en la *Figura 35*.

Index	Sensibilidad	Especificidad	Exactitud	Precision	Error_Medio	Valor_F	Ratio_FP	Coefficiente_Matthews	Indice_Jaccard	Parametro_d
b	0.884	0.996	0.987	0.95	0.013	0.916	0.004	0.909	0.844	3.831
ch	1	1	1	1	0	1	0	1	1	inf
d	0.833	0.986	0.975	0.811	0.025	0.822	0.014	0.809	0.698	3.154
f	0.7	1	0.983	1	0.017	0.824	0	0.829	0.7	inf
g	0.867	0.994	0.99	0.812	0.01	0.839	0.006	0.834	0.722	3.628
k	1	0.992	0.992	0.915	0.008	0.956	0.008	0.952	0.915	inf
l	0.891	0.987	0.979	0.872	0.021	0.881	0.013	0.87	0.788	3.472
m	0.915	0.985	0.979	0.86	0.021	0.887	0.015	0.876	0.796	3.549
n	0.487	0.996	0.958	0.905	0.042	0.633	0.004	0.646	0.463	2.608
ny	1	0.998	0.998	0.875	0.002	0.933	0.002	0.935	0.875	inf
p	0.943	0.998	0.994	0.971	0.006	0.957	0.002	0.954	0.917	4.449
r	0.842	0.976	0.971	0.571	0.029	0.681	0.024	0.68	0.516	2.983
rr	1	0.992	0.992	0.765	0.008	0.867	0.008	0.871	0.765	inf
s	1	0.994	0.994	0.893	0.006	0.943	0.006	0.942	0.893	inf
t	0.868	0.991	0.979	0.92	0.021	0.893	0.009	0.882	0.807	3.502
x	0.917	1	0.998	1	0.002	0.957	0	0.956	0.917	inf
y	0.92	1	0.996	1	0.004	0.958	0	0.957	0.92	inf
z	0.893	0.974	0.969	0.658	0.031	0.758	0.026	0.751	0.61	3.18

Figura 35: Tabla que recoge todas las funciones estadísticas para la matriz de confusión de las vocales del archivo "salida.txt"

5.7 Prueba del cálculo y creación de una tabla con funciones estadísticas de manera individual de una matriz de confusión de letras

Para realizar la prueba de funcionamiento de las funciones escritas en el apartado “Cálculo y creación de una tabla para las funciones estadísticas de manera individual en una matriz de confusión de letras”, se ha utilizado la matriz de confusión de vocales vista en la Figura 33. Los resultados obtenidos son:

1. Para la función que calcula la Sensibilidad (“EstatSensibilidad(matriz)”) se obtiene la Figura 36. Se puede observar que la Sensibilidad de los resultados obtenidos por el software en el RAH es mayor a 0.8 para todas las vocales, en general, los resultados dados por el sistema para reconocer VP son buenos, aunque mejorables.

La Sensibilidad es:

	Sensibilidad
a	0.963
e	0.938
i	0.825
o	0.847
u	0.870

Figura 36: Sensibilidad de las vocales para el archivo "salida.txt"

2. Para la función que calcula la Especificidad (“EstatEspecificidad(matriz)”) se obtiene la *Figura 37*. Se puede observar que la Especificidad de los resultados obtenidos por el software en el RAH es mayor a 0.95 para todas las vocales, por lo que el sistema si tiene una buena capacidad para reconocer VN.

La Especificidad es:

	Especificidad
a	0.961
e	0.977
i	0.976
o	0.971
u	0.981

Figura 37: Especificidad de las vocales para el archivo "salida.txt"

3. Para la función que calcula la Exactitud (“EstatExactitud(matriz)”) se obtiene la *Figura 38*. Se puede observar que la Exactitud de los resultados obtenidos por el software en el RAH es mayor a 0.9 para todas las vocales, por lo que el sistema tiene poca incertidumbre.

La Exactitud es:

	Exactitud
a	0.962
e	0.970
i	0.950
o	0.942
u	0.967

Figura 38: Exactitud de las vocales para el archivo "salida.txt"

4. Para la función que calcula la Precisión (“EstatPrecision(matriz)”) se obtiene la *Figura 39*. Se puede observar que la Precisión de los resultados obtenidos por el software en el RAH es mayor a 0.85 para todas las vocales, es un valor razonable, aunque indica que tiene un pequeño porcentaje de fallos.

La Precision es:

	Precision
a	0.902
e	0.906
i	0.876
o	0.904
u	0.870

Figura 39: Precisión de las vocales para el archivo "salida.txt"

5. Para la función que calcula el Error Medio (“EstatError_Medio(matriz)”) se obtiene la *Figura 40*. Se puede observar que el Error Medio de los resultados obtenidos por el software en el RAH es menor a 0.06 para todas las vocales, por lo que el porcentaje de error en el reconocimiento de vocales es siempre menor al 6%.

El Error Medio es:

	Error_Medio
a	0.038
e	0.030
i	0.050
o	0.058
u	0.033

Figura 40: Error Medio de las vocales para el archivo "salida.txt"

6. Para la función que calcula el Valor-F (“EstatValor_F(matriz)”) se obtiene la *Figura 41*. Se puede observar que el Valor-F de los resultados obtenidos por el software en el RAH es mayor a 0.85 para todas las vocales, por lo que aún falta afinar más el sistema para que este valor sea más cercano a 1 (valor ideal).

El Valor-F es:

	Valor_F
a	0.932
e	0.922
i	0.850
o	0.875
u	0.870

Figura 41: Valor-F de las vocales para el archivo "salida.txt"

7. Para la función que calcula el Ratio de FP (“EstatRatio_FP(matriz)”) se obtiene la *Figura 42*. Se puede observar que el Ratio de Falsos Positivos de los resultados obtenidos por el software en el RAH es menor a 0.05 para todas las vocales, es decir, menor al 5%.

El Ratio de Falsos Positivos es:

	Ratio_FP
a	0.039
e	0.023
i	0.024
o	0.029
u	0.019

Figura 42: Ratio de FP de las vocales para el archivo "salida.txt"

8. Para la función que calcula el Coeficiente de Matthews (“EstatMatthews(matriz)”) se obtiene la *Figura 43*. Se puede observar que el Coeficiente de Matthews de los resultados obtenidos por el software en el RAH es mayor a 0.8 para todas las vocales, el resultado indica que el programa predice de una manera aceptable los resultados, aunque no de manera perfecta (el resultado sería 1).

El Coeficiente de Matthews es:

	Coeficiente_Matthews
a	0.906
e	0.903
i	0.821
o	0.837
u	0.851

Figura 43: Coeficiente de Matthews de las vocales para el archivo "salida.txt"

9. Para la función que calcula el Índice de Jaccard (“EstatJaccard(matriz)”) se obtiene la *Figura 44*. Se puede observar que el Índice de Jaccard de los resultados obtenidos por el software en el RAH es mayor a 0.7 para todas las vocales, por lo que hay un cierto grado de similitud entre los conjuntos estudiados, aunque el resultado se debe mejorar para obtener un reconocimiento perfecto o cercano a la perfección.

El Índice de Jaccard es:

Indice_Jaccard	
a	0.872
e	0.855
i	0.739
o	0.777
u	0.770

Figura 44: Índice de Jaccard de las vocales para el archivo "salida.txt"

10. Para la función que calcula el Parámetro D’ (“EstatParamD(matriz)”) se obtiene la *Figura 45*. Se puede observar que el Parámetro D’ de los resultados obtenidos por el software en el RAH es cercano a 3 para todas las vocales, el resultado es positivo, lo que indica que hay mayor probabilidad de acierto que de fallo y, al ser cercano a 3, se puede concluir que la probabilidad de fallo es relativamente pequeña (aunque no es 0, donde el resultado de este parámetro sería $+\infty$, lo que indica un reconocimiento perfecto).

El Parámetro D es:

Parametro_d	
a	3.553
e	3.542
i	2.910
o	2.928
u	3.199

Figura 45: Parámetro D' de las vocales para el archivo "salida.txt"

6. Conclusiones y líneas futuras

Al realizar este Trabajo de Fin de Grado, se han estudiado distintos parámetros estadísticos con los que poder evaluar el correcto funcionamiento de un software que trabaje con el RAH, desarrollando con estos parámetros una librería en el lenguaje de programación Python con la que poder evaluar los posibles errores al implementar la receta ASICA en el software Kaldi, la cual se puede utilizar también con otras recetas para el RAH que devuelvan parámetros similares a los que utiliza la librería para realizar la evaluación estadística.

Gracias a esta librería, los investigadores podrán obtener de una manera sencilla y rápida mediante el lenguaje Python los resultados estadísticos relacionados con su estudio, por lo que será más fácil identificar qué parámetros del entrenamiento o de la receta utilizada en el software Kaldi (u otro software de RAH con el que se obtengan resultados similares) se deben corregir para mejorar la eficacia del programa.

Sería interesante continuar este trabajo con la creación de más variables estadísticas para la evaluación de los resultados obtenidos en distintos softwares de RAH. Pudiendo además transcribir la librería en otros lenguajes de programación o la utilizar otros parámetros de entrada distintos para realizar diversas pruebas estadísticas.

7. Bibliografía

- Abder-Rahman, A. (24 de Octubre de 2016). *Introducing Pandas*. Obtenido de envatotuts+: <https://code.tutsplus.com/tutorials/introducing-pandas--cms-26514>
- Andrews, L. C. (1998). *Special functions of mathematics for engineers*. SPIE Press.
- Apple. (2020). Acerca de los scripts de shell en Terminal en el Mac. Obtenido de Apple: <https://support.apple.com/es-es/guide/terminal/apd53500956-7c5b-496b-a362-2845f2aab4bc/mac>
- Ariza-López, F. J.-A.-F. (2018). *Control estricto de matrices de confusión por medio de distribuciones multinomiales*. Obtenido de GeoFocus. Revista Internacional de Ciencia y Tecnología de la Información Geográfica, (21), 215-226.
- Armenteros, A. M. (2010). *Error, incertidumbre, precisión y exactitud, términos asociados a la calidad espacial del dato geográfico*. Obtenido de In Catastro: formación, investigación y empresa: Selección de ponencias del I Congreso Internacional sobre catastro unificado y multipropósito: https://d1wqtxts1xzle7.cloudfront.net/38594565/Informacion_de_Metrologia.pdf?1440729554=&response-content-disposition=inline%3B+filename%3DERROR_INCERTIDUMBRE_PRECISION_Y_EXACTITU.pdf&Expires=1591319520&Signature=X~ilg3Rj6tRVQ2Q8Ag3NDsfASPUm0bkTy2WMn2fSkA
- Bagwell, C. (2013). *Sox*. Obtenido de sox.sourceforge: <http://sox.sourceforge.net/sox.html>
- Burke, D. S. (1988). *Measurement of the false positive rate in a screening program for human immunodeficiency virus infections*. Obtenido de The New England journal of medicine, 319(15), 961–964.: <https://doi.org/10.1056/NEJM198810133191501>
- Chacon, S. (24 de Diciembre de 2014). *git everything is local*. Obtenido de git-scm: <https://git-scm.com/book/en/v2>
- Comber, A. F. (07 de Octubre de 2012). *Spatial analysis of remote sensing image classification accuracy*. Obtenido de ELSEVIER: <https://doi.org/10.1016/j.rse.2012.09.005>
- Commons, C. (2007). *Librerías estáticas y dinámicas*. Obtenido de Chuidiang: <http://www.chuidiang.org/linux/herramientas/librarias.php>

- Contributors, T. S. (2018). *Spyder*. Obtenido de Spyder - The Scientific Python Development Environment: <https://www.spyder-ide.org/>
- DiCanio, C. (15 de Junio de 2013). *La fonética segmental: punto y manera de articulación*. Obtenido de Laboratorio de Haskins: http://www.acsu.buffalo.edu/~cdicarlo/pdfs/Fonetica_segmental.pdf
- docs.python. (20 de Abril de 2020). *15.16. errno — Standard errno system symbols*. Obtenido de Python: <https://docs.python.org/2/library/errno.html>
- Duxans Barrobés, H. R.-j. (2012). *Reconocimiento automático del habla*. Barcelona: Universitat Oberta de Catalunya.
- Evin, D. A. (2011). *Incorporación de información suprasegmental en el proceso de reconocimiento automático del habla*. Obtenido de Universidad de Buenos Aires. Facultad de Ciencias Exactas y Naturales: https://bibliotecadigital.exactas.uba.ar/download/tesis/tesis_n4920_Evin.pdf
- Ganeshan, A. (18 de Julio de 2019). *Articulación de las consonantes*. Obtenido de Pressbooks: <https://linghisintro.pressbooks.com/chapter/articulacion-de-las-consonantes/#:~:text=El%20modo%20de%20articulaci%C3%B3n,-El%20modo%20de&text=Las%20consonantes%20%2F%2C%20%2Fb%2F%2C%20%2Ft%CC%AA,que%20crea%20una%20fricci%C3%B3n%20audible>.
- Glen, S. (10 de Octubre de 2015). *Inverse Normal Distribution*. Obtenido de Statistics How To: <https://www.statisticshowto.com/inverse-normal-distribution/>
- GNU. (05 de Mayo de 2016). *Autoconf*. Obtenido de GNU: <https://www.gnu.org/software/autoconf/autoconf.html>
- GNU. (03 de Septiembre de 2018). *Automake*. Obtenido de GNU: <https://www.gnu.org/software/automake/>
- GNU. (30 de Octubre de 2019). *GNU Wget*. Obtenido de GNU: <https://www.gnu.org/software/wget/>
- GNU. (07 de Mayo de 2020). *GCC, the GNU Compiler Collection*. Obtenido de gcc.gnu: <https://gcc.gnu.org/>
- Greelane. (26 de Junio de 2019). *Saber la diferencia entre consonantes sonoras y sordas*. Obtenido de Greelane: <https://www.greelane.com/es/idiomas/ingles-como-segundo-lenguaje/voiced-and-voiceless-consonants-1212092/#:~:text=Consonantes%20sonoras%20requieren%20el%20uso,modificar%20a%C3%AAn%20m%C3%A1s%20el%20habla>.

- Green, D. M. (1966). *Signal Detection Theory and Psychophysics*. Nueva York - Londres: John Wiley and Sons, Inc.
- Home, A. (2020). *Anaconda Documentation - Anaconda Navigator*. Obtenido de Anaconda : <https://docs.anaconda.com/anaconda/navigator/>
- Inc., A. (2020). *Acerca de los scripts de shell en Terminal en el Mac*. Obtenido de Support Apple: <https://support.apple.com/es-es/guide/terminal/apd53500956-7c5b-496b-a362-2845f2aab4bc/mac>
- Le, D. L. (20 de Agosto de 2017). *Automatic Paraphasia Detection from Aphasic Speech: A Preliminary Study*. Obtenido de University of Michigan, Ann Arbor, MI 48109, USA Computer Science and Engineering, University Center for Language and Literacy:
http://web.eecs.umich.edu/~emilykmp/EmilyPapers/2017_Le_Interspeechb.pdf
- Le, D. L. (2018). Automatic quantitative analysis of spontaneous aphasic speech. . *Elsevier*, Speech Communication, 100, 1-12.
- MacMillan, N., & Creelman, C. (2005). *Detection Theory: A User's Guide*. Lawrence Erlbaum Associates.
- Matthews, B. W. (20 de Octubre de 1975). *Comparison of the Predicted and Observed Secondary Structure of T4 Phage Lysozyme*. Obtenido de National Library of Medicine: <https://pubmed.ncbi.nlm.nih.gov/1180967/>
- Montoro, A. F. (2012). *Python 3 al descubierto*. Obtenido de RC Libros:
https://d1wqxts1xzle7.cloudfront.net/54161186/cap1.PHYTON.pdf?1502942016=&response-content-disposition=inline%3B+filename%3DPython_3_al_descubierto.pdf&Expires=1591154748&Signature=WXsQx4N8bqq63jjCHayqqg8cBJSn6usAxBpcPP~4Zx2M~nDspOZUDNjHU7IHaEafgVUb3qRjy
- Moreno-Torres Sánchez, I. (28 de Agosto de 2019). *ASICA Kaldi Recipe*. Obtenido de GitHub: https://github.com/Caliope-SpeechProcessingLab/ASICA_KaldiRecipe_old/blob/master/README.md
- Nancy Helm-Estabrooks, M. L. (2005). *Manual de la afasia y de terapia de la afasia. Segunda Edición*. Editorial Panamericana.
- Nvie. (24 de Agosto de 2014). *times 0.7*. Obtenido de pypi: <https://pypi.org/project/times/>
- Pierre Rebours, T. M. (2006). *Misclassification Rate*. Obtenido de ScienceDirect: <https://www.sciencedirect.com/topics/engineering/misclassification-rate>

- Pita Fernández, S. P. (2003). *Pruebas diagnósticas: Sensibilidad y especificidad*. Obtenido de Unidad de Epidemiología Clínica y Bioestadística. Complexo Hospitalario Universitario de A Coruña (España): https://www.fisterra.com/mbe/investiga/pruebas_diagnosticas/pruebas_diagnosticas.asp#Tabla%201
- Povey, D. a. (Diciembre de 2011). *The Kaldi Speech Recognition Toolkit*. (IEEE Signal Processing Society) Obtenido de Kaldi: <https://kaldi-asr.org/>
- Python. (2020). *Why was Python created in the first place?* Obtenido de General Python FAQ - Python 3.8.3 documentation: <https://docs.python.org/3/faq/general.html#why-was-python-created-in-the-first-place>
- Python, D. (2020). *History and License — Python 3.8.3 documentation*. Obtenido de Python Documentation: <https://docs.python.org/3/license.html>
- Rafael González V., A. H.-H. (2014). *Afasia: una perspectiva clínica*. Chile: Departamento de Neurología y Neurocirugía, HCUCH.
- Real, R. (1999). *Tables of significant values of Jaccard's index of similarity*. Obtenido de Miscellania Zoologica, 22(1), 29-40: https://www.researchgate.net/profile/Raimundo_Real/publication/254472759_Tables_of_significant_values_of_Jaccard's_index_of_similarity/links/00b7d525f9c1961b89000000.pdf
- Reddy, V. R. (Septiembre de 2014). *Person identification in natural static postures using kinect*. Obtenido de European Conference on Computer Vision (pp. 793-808). Springer, Cham: https://link.springer.com/chapter/10.1007/978-3-319-16181-5_60
- Reitz, K. (2016). *Instalando Python 3 en Mac OS X*. Obtenido de The Hitchhiker's Guide to Python: <https://python-guide.es.readthedocs.io/es/latest/starting/install3/osx.html>
- Reyes, P. R.-F. (2009). *Diversidad, distribución, riqueza y abundancia de condrictios de aguas profundas a través del archipiélago patagónico austral, Cabo de Hornos, Islas Diego Ramírez y el sector norte del paso Drake*. Obtenido de Revista de biología marina y oceanografía: <https://dx.doi.org/10.4067/S0718-19572009000100025>
- Rico Schmidt, E. (2018). *shutil — Operaciones de archivo de alto nivel*. Obtenido de rico-schmidt: <https://rico-schmidt.name/pymotw-3/shutil/>

- Rodríguez, D. (15 de Junio de 2018). *Guardar y leer archivos CSV con Python*. Obtenido de Analyticslane: <https://www.analyticslane.com/2018/06/15/guardar-y-leer-archivos-csv-con-python/#:~:text=Uno%20de%20los%20formatos%20m%C3%A1s,registros%20delimitados%20por%20un%20separador>.
- Sabri Boughorbel, F. J.-A. (2 de Junio de 2017). *Optimal Classifier for Imbalanced Data Using Matthews Correlation Coefficient Metric*. Obtenido de National Library of Medicine: <https://pubmed.ncbi.nlm.nih.gov/28574989/>
- Sobrino, D. C. (Marzo de 2018). *¿Cómo funciona el reconocimiento automático del habla?* Obtenido de Medium: <https://medium.com/soldai/c%C3%B3mo-funciona-el-reconocimiento-autom%C3%A1tico-del-habla-eb038ecfe72e>
- SRI. (24 de Septiembre de 2019). *SRILM - The SRI Language Modeling Toolkit*. Obtenido de SRI International: <http://www.speech.sri.com/projects/srilm/#:~:text=SRILM%20is%20a%20toolkit%20for,AND%20Research%20Laboratory%20since%201995>.
- SRILM, D. (24 de Septiembre de 2019). *Downloading and Building SRILM*. Obtenido de SRI International: <http://www.speech.sri.com/projects/srilm/download.html>
- Terpstra, B. (25 de Diciembre de 2009). *Homebrew, the perfect gift for command line lovers.* Obtenido de Engadget: https://www.engadget.com/2009/12/25/homebrew-the-perfect-gift-for-command-line-lovers/?guccounter=1&guce_referrer=aHR0cHM6Ly9lcy53aWtpcGVkaWEub3JnLw&guce_referrer_sig=AQAAAE-GssQLDdrvYeP4EMW-yuEbEsDHqUx1BMWltgmKAUxBqEmTCFGqfcT6yej9_-6eQ-zsjI5X_0puUKWkVG9u
- Torvalds, L. (07 de 04 de 2005). *"So I'm writing some scripts to try to track things a whole lot faster."*. Obtenido de <https://marc.info/?l=linux-kernel&m=111288700902396>
- Tóth, L. G. (2015). *Automatic Detection of Mild Cognitive Impairment from Spontaneous Speech Using ASR*. Desden, Germany: ISCA Archive.
- tutorialspoint. (2020). *make - Unix, Linux Command*. Obtenido de tutorialspoint: https://www.tutorialspoint.com/unix_commands/make.htm
- uniwebsidad. (2020). *10.1. Módulos de sistema*. Obtenido de uniwebsidad: <https://uniwebsidad.com/libros/python/capitulo-10/modulos-de-sistema>

Vaughan, G. V. (04 de Junio de 2016). *GNU Libtool - The GNU Portable Library Tool*.

Obtenido de GNU: <https://www.gnu.org/software/libtool/>

Wikipedia. (16 de Mayo de 2020). *Biblioteca (informática)*. Obtenido de Wikipedia: [https://es.wikipedia.org/wiki/Biblioteca_\(inform%C3%A1tica\)](https://es.wikipedia.org/wiki/Biblioteca_(inform%C3%A1tica))

Wikipedia. (6 de Mayo de 2020). *Homebrew (gestor de paquetes)*. Obtenido de Wikipedia: [https://es.wikipedia.org/wiki/Homebrew_\(gestor_de_paquetes\)](https://es.wikipedia.org/wiki/Homebrew_(gestor_de_paquetes))

Xando. (16 de Noviembre de 2013). *subprocess.run 0.08*. Obtenido de pypi: <https://pypi.org/project/subprocess.run/>

I. Glosario

- RAH – Reconocimiento Automático del Habla
- TFG – Trabajo de Fin de Grado
- CWI - Centrum Wiskunde & Informatica
- GCC – Compilador GNU de C
- GUI – Interfaz Gráfica de Usuario (Graphical User Interface)
- LDC – Linguistic Data Consortium
- AVC - Accidente Cerebro Vascular
- TEC – Traumatismo Encefalocraneano
- TU – Tumor
- CDC – Conjunto de Datos bajo Control
- CDR – Conjunto de Datos de Referencia
- VP – Verdaderos Positivos
- VN – Verdaderos Negativos
- FP – Falsos Positivos
- FN – Falsos Negativos
- SO – Sistema Operativo
- TDS – Teoría de Detección de Señales

II. Anexos

II.I

ANEXO I: Instalación de Kaldi y Receta ASICA

Tabla 20: Instalación Kaldi en macOS

```
brew install automake
brew install wget
brew install sox
brew install gcc
brew install libtool

git clone https://github.com/kaldi-asr/kaldi.git kaldi --origin upstream
cd kaldi

cd tools
chmod +x ./extras/check_dependencies.sh
./extras/check_dependencies.sh
make

cd ..
 
cd src
./configure --shared
make depend -j 8
make -j 8
extras/install_irstlm.sh
```

Tabla 21: Instalación Receta ASICA

```
cd ~
cd kaldi
cd egs
git clone https://github.com/Caliope-SpeechProcessingLab/ASICAKaldiRecipe.git
```

II.II ANEXO II: Elección de gradación de color para graficar matrices

Para realizar la función con la que se grafiquen y coloreen las matrices de confusión normalizadas (vista en el apartado “*Función para graficar y colorear matrices normalizadas*”), es necesario determinar la gradación de color más adecuada con la que colorear las matrices. Para ello se han realizado varias pruebas, los ejemplos de las pruebas se realizan con la matriz de confusión de las consonantes, puesto que es una matriz de un mayor tamaño con lo que se pueden ver diferencias más significativas:

En primer lugar, la matriz de confusión para las consonantes se representa utilizando un mapa de calor con una gradación lineal del color azul (‘Blues’). Se puede observar que, de esta manera, las diferencias de color entre valores bajos son pequeñas, por lo que un error de 0.08 es muy parecido visualmente a un error de 0.02 (*Figura 46*).

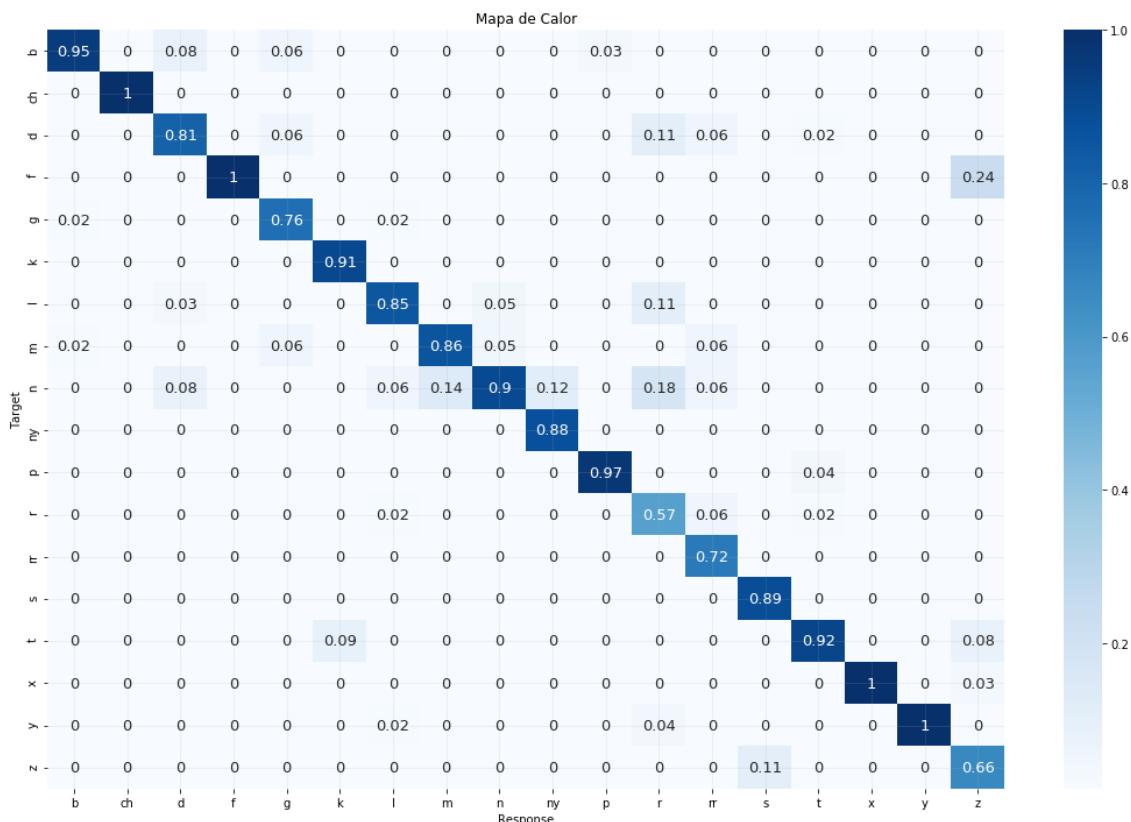


Figura 46: Matriz de confusión coloreada en una gradación azul de manera lineal

Para solucionar este problema de falta de contraste entre los valores, se opta por usar una escala logarítmica para representar la gradación de colores. De esta manera, se obtiene una figura donde el contraste entre resultados cercanos es bastante más notable (*Figura 47*). De esta manera se establece que, para la gradación de color en la visualización de matrices de confusión, se debe usar preferentemente una escala logarítmica.

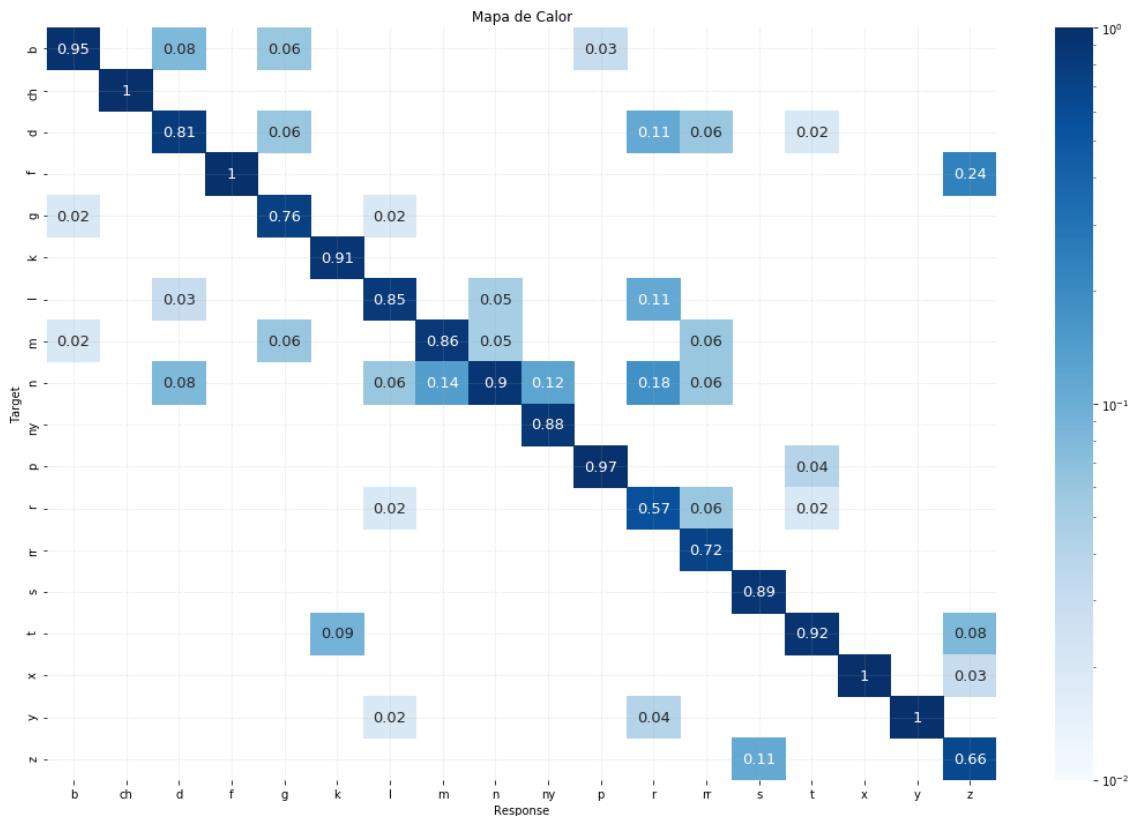


Figura 47: Matriz de confusión coloreada en una gradación azul en escala logarítmica

Se estudia también la posibilidad de combinar escalas para dos colores, una escala para los valores situados entre 0 y 0.5, y otra escala para valores entre 0.5 y 1. De esta manera surge la función “*PlotMatOrangeBlue(normalize_confusion_matrix, nombre)*” explicada en el apartado “*Función para graficar y colorear matrices normalizadas*”, donde se utiliza una gradación de naranja para valores entre 0 y 0.5, y una de azul para valores entre 0.5 y 1.

La librería “*matplotlib*” tiene graduaciones de colores predeterminados, entre los cuales se encuentra la escala “*viridis*”. Se ha estudiado la posibilidad de añadir esta gradación para representar los colores de la figura resultante de las matrices de confusión

normalizadas y, finalmente, se ha decidido añadir esta opción a la librería como la función “*PlotMatViridis(normalize_confusion_matrix, nombre)*”, explicada en el apartado “Función para graficar y colorear matrices normalizadas”.

Por último, se ha estudiado la posibilidad de añadir un factor más en la visualización de las matrices de confusión normalizadas, se probó a colorear de rosa los valores en los que el programa acertase en el 100% de los casos (valor igual a 1) (*Figura 48*). Finalmente, esta idea se descartó puesto que se considera que no es necesaria para la correcta visualización de la matriz e, incluso, podría distorsionar la percepción de los valores (letras) para los que el software comete errores en el reconocimiento.

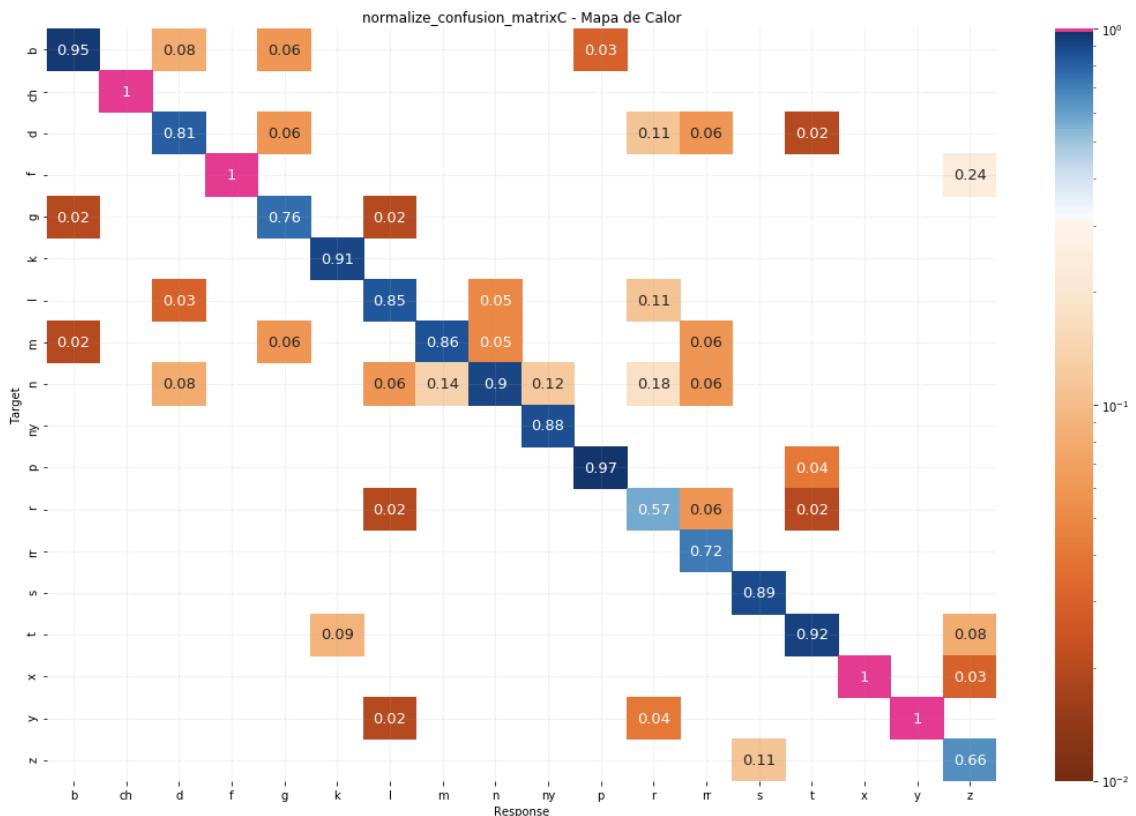


Figura 48: Matriz de confusión coloreada en una graduación naranja-azul-rosa en escala logarítmica

II.III ANEXO III: Comprobación de resultados del Parámetro D'

En una matriz de confusión 2x2, se puede calcular el Parámetro D' para cada elemento (letra) utilizando la fórmula (*Ecuación 11*):

$$d' = z(H) - z(FA)$$

Ecuación 14: Cálculo del Parámetro D' para matrices de confusión

Donde H es la probabilidad condicional de acierto, es decir, la probabilidad de que se reconozca una letra conociendo de antemano la letra que es; FA es la probabilidad condicional de fallo, es decir, la probabilidad de que se reconozca una letra cuando realmente no lo es. Una vez conocidas estas dos probabilidades condicionales, se debe aplicar la función normal de distribución inversa $z(q)$, definida en el apartado de la memoria “*Parámetro D’*”, para cada una de ellas y realizar la diferencia de ambos resultados.

Al calcularse la probabilidad condicional, se puede obtener un resultado entre 0 y 1, siendo 0 la probabilidad nula y 1 el acierto absoluto. A este resultado, se le debe aplicar la función normal de distribución inversa, en Python se calcula mediante la función `norm.ppf()`, con lo que obtendremos un resultado que variará entre $-\infty$ y $+\infty$. Para comprobar el correcto funcionamiento de esta función normal de distribución inversa en Python, se creará un programa con el que se generan un millón de números aleatorios entre 0 y 1 (posibles resultados del cálculo de la probabilidad condicional), a los que se les aplica la $z(q)$, y se procede a dibujar los resultados obtenidos *Tabla 22*. La gráfica resultante se puede observar en la *Figura 49*. Si se calcula la diferencia para dos resultados del cálculo de la probabilidad condicional (Parámetro D’), la solución será de igual manera un número entre $-\infty$ y $+\infty$.

Tabla 22: Comprobación de la función normal de distribución inversa

#Se generan 1.000.000 valores aleatorios entre 0 y 1. “random” genera valores aleatorios, al añadir “.rand” los genera entre 0 y 1

```
prueba = np.random.rand(1000000)
```

#Se obtiene la función normal de distribución inversa de la probabilidad condicional de acierto H y se imprime respecto a los números aleatorios correspondientes entre 0 y 1 (Porcentajes aleatorios creados para la prueba) que supuestamente corresponden a la probabilidad condicional de acierto $H = VP/(VP+FN)$

```
zH = norm.ppf(prueba)
plt.title("Resultado FNDF para 1.000.000 valores aleatorios entre 0 y 1")
plt.xlabel("Valores entre 0 y 1")
plt.ylabel("Funcion Normal de Distribucion Inversa")
plt.plot(prueba, zH, 'o')
plt.grid()
```

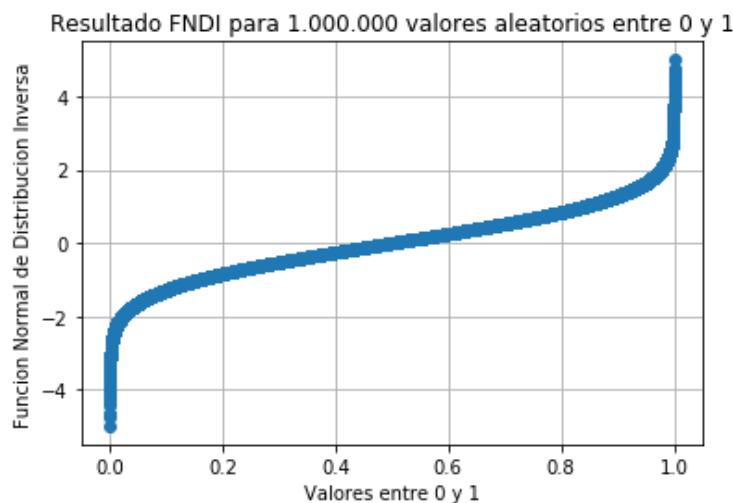


Figura 49: Función normal de distribución inversa para un millón de valores entre 0 y 1



UNIVERSIDAD
DE MÁLAGA

uma.es

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga