FACULTAD DE INGENIERÍA Y CIENCIAS APLICADAS

CLASIFICACIÓN SONORA DE INSTRUMENTOS ANDINOS A TRAVÉS DE MACHINE LEARNING

Autor

Aníbal Fernando Bonilla Ambrossi

Año

2019

FACULTAD DE INGENIERÍA Y CIENCIAS APLICADAS

CLASIFICACIÓN SONORA DE INSTRUMENTOS ANDINOS A TRAVÉS DE MACHINE LEARNING

Trabajo de Titulación presentado en conformidad con los requisitos establecidos para optar por el título de Ingeniero de Sonido y Acústica

Profesor Guía

Msc. Paúl Adrián Cabezas Yánez

Autor

Aníbal Fernando Bonilla Ambrossi

Año

2019

DECLARACIÓN DEL PROFESOR GUÍA

“Declaro haber dirigido el trabajo, “Clasificación Sonora de Instrumentos Andinos A Través de Machine Learning”, del estudiante, Aníbal Fernando Bonilla Ambrossi, en el semestre 201910, orientando sus conocimientos y competencias para un eficiente desarrollo del tema escogido y dando cumplimiento a todas las disposiciones vigentes que regulan los Trabajos de Titulación”.

_____

Paúl Adrián Cabezas Yánez
Master en Industrias Creativas en Música y Sonido
CI: 171918954-8

DECLARACIÓN DEL PROFESOR CORRECTOR

"Declaro haber revisado este trabajo, “Clasificación Sonora de Instrumentos Andinos A Través de Machine Learning”, del estudiante, Aníbal Fernando Bonilla Ambrossi, en el semestre 201910, dando cumplimiento a todas las disposiciones vigentes que regulan los Trabajos de Titulación".

_____

Luis Alberto Bravo Moncayo
Doctor en Ingeniería Acústica
CI: 171171060-6

DECLARACIÓN DE AUTORÍA DEL ESTUDIANTE

“Declaro que este trabajo es original, de mi autoría, que se han citado las fuentes correspondientes y que en su ejecución se respetaron las disposiciones legales que protegen los derechos de autor vigentes.”

_____

Aníbal Fernando Bonilla Ambrossi
CI: 100393615-8

AGRADECIMIENTOS

A mis padres y familia, quienes me han ayudado a superar todos los obstáculos que se han ido dando durante este proceso de aprendizaje constante. A mis amigos, los cuales han compartido experiencias que han enriquecido y complementado todas las enseñanzas inducidas durante este largo período. Finalmente, a mis compañeros y futuros colegas por su gran sinceridad y humildad, siendo en sí un apoyo fundamental para mi vida profesional.

DEDICATORIA

A mis padres e hijos, por ser la razón de mi esfuerzo y dedicación. Cada uno de los gestos que me han brindado y han perdurado en los recuerdos desde que tengo uso de razón, durante mi educación y hasta el final de los días.

RESUMEN

El objetivo planteado en el actual documento es proponer una metodología para clasificar sonidos andinos a través de una herramienta que desarrolle Machine Learning. Para la investigación realizada se desarrolló una base de datos mediante un software llamado "*NN-XT Sampler"*, el cual tiene la función de samplear sonidos que permitan editar y diseñar audios con los requerimientos óptimos para un entrenamiento de aprendizage automático con la herramienta llamada *Wekinator*. Para el desarrollo de la validación del programa se pretende clasificar sonidos a través de varias muestras de entrenamiento y unas pocas para la respectiva prueba o evaluación de la precisión en los clasificadores que posee el software *Wekinator* mediante una técnica estadística de validación cruzada. Finalmente, los resultados dieron que para la clasificación de instrumentos de viento andinos cumpliendo dos clases de analogías detalladas en el documento existen tres de cinco clasificadores que tienen un porcentaje de precisión superiores al 84%, siendo estos los de Árbol de decisión, AdaBost y SVM, superando a los algoritmos de K-Neigbor y Naive Boyes.

ABSTRACT

The objective that I propose in the real document is propose a Methodology to Classify Andean sounds through a tool that develops Machine Learning. For the realized investigation, it developed a data base through a software called "*NN-XT Sampler"* which function is allowed edit and design sounds with good quality for the Machine Learning processing. The development process for the validation program starts with various training samplers and test samplers. These samplers used two phases of the investigation. The first phase is the training process and the second is the testing of the program using a software called *Wekinator*. Finally, the results show that three of the five classifiers have superior 84% of precision. These classifiers are the Tree Decision, AdaBost and SVM, exceeding the K-Neigbor and Naïve Boyes algorithms.

ÍNDICE

1. CAPITULO I: INTRODUCCIÓN

1. 1 Antecedentes

La síntesis musical basada en redes neuronales ha sido un tema de investigación activo en las últimas dos décadas. Al simular la propagación de ondas, los enfoques basados en modelaje de redes neuronales pueden producir sonidos muy reales. Las cuales pueden ser utilizadas en instrumentos de cuerda evaluando sus parámetros como técnicas de ejecución y así abrir campo a la aplicación del modelo en instrumentos de viento (Liang, 2000). Es adecuado indicar la evolución de modelos de Machine Learning para crear el aprendizaje mediante un algoritmo, cada uno de ellos tiene sus ventajas y desventajas, siendo la mayoría de ellas una apertura para nuevas ramas de investigación en cada uno de los parámetros que se analizan, siendo este dato una buena noticia para el investigador que pretende abrir aún más las posibilidades de esta herramienta.

Es fácil entender que un nuevo campo en la inteligencia artificial puede tener resultados ambiciosos, no solamente siendo el ML una aplicación para la robótica, juegos, medicina, finanzas, etc, sinó también la utilización del ML en el audio; es así que han existido recientes desarrollos en el campo de las redes neuronales artificiales, abriendo la puerta a una amplia variedad de usos creativos. Como por ejemplo las arquitecturas de Google, WaveNet generando una composición de piano u na muestra a la vez en tiempo real, simulando una inteligencia artificial como si un pianista profesional estuviera ejecutando el instrumento. Y la arquitectura NSynth igualmente de Google, la cual usa clases de instrumentos para generar timbres específicos y nuevos, interpolando diferentes tipos de clases de instrumentos (Colonel, 2017).

En los últimos años se han desarrollado nuevos sistemas de redes neuronales como por ejemplo el autoencoder, las cuales capturan las características espectrales de las señales de audio en el dominio del tiempo y la frecuencia, construyendo bases de datos que son generadas por entradas en base al conocimiento adquirido durante la fase de entrenamiento de los algoritmos ML. Es así que el generar nuevos timbres mediante muestras de instrumentos toma fuerza al momento en el que algoritmos obtienen interpolaciones de dos fuentes para obtener señales mixtas, estos resultados dan apertura a una nueva literatura de procesamiento de señales para la obtención de una nueva síntesis de sonidos novedosos de dos o más fuentes diferentes de sonido. Esta nueva literatura conoce varias técnicas con nuevas definiciones o términos como el análisis de síntesis, síntesis cruzada, codificación, mortificación o hibridación (Gabrielli Y Cella, 2018).

El año 2015 se desarrolló una aplicación que permitía la autentificación de Usuarios a través de la biometría del comportamiento, también llamada como “biometría conductual” mediante algoritmos de clasificación de aprendizaje automático multicapa “Multilayer Machine Learning’’, una aplicación muy importante para la seguridad en teléfonos inteligentes. La metodología de este estudio realizado en New York en la Escuela de ciencias y computación de Seidenberg muestra resultados casi perfectos, pero conclusiones muy polémicas debido a que dieron un cambio total a la posible finalidad de esta aplicación. Estas conclusiones se basan en los resultados, los cuales indican que el sistema de aprendizaje mediante movimientos en los censores de los teléfonos obteniendo una precisión de autenticación del 90% y en el estudio completo con más muestras una precisión del 93%, siendo excelentes resultados pero las conclusiones muestran un criterio diferente y la razón es que debido a que las investigaciones núnca obtuvieron 100% de las precisiones, los resultados no respaldan la utilización para ser un método de autenticación ya que pueden lograr que algún tipo de usuario pueda desbloquear algún tipo de dispositivo en algún momento. Es así que tendría

más sentido aprovechar la biometría del comportamiento para mantener el teléfono desbloqueado durante un uso prolongado. Finalizando con una conclusión muy convincente y coherente (Maghsoudi, 2017).

Una estudio realizado en la Universidad Técnica de Gdansk, en la Facultad de Electrónica, Telecomunicaciones e Informática desarrollado en 1998 indica la descripción de un sistema basado en el aprendizaje automático para un reconocimiento y predicción de la música, siendo un adelanto interesante para la época y dar la apertura a algoritmos más avanzados como los utilizados en las aplicaciones como Shazam, prediciendo canciones con algoritmos de aprendizaje automático y redes neuronales. Estas mejoras avanzaron gracias a que el estudio tuvo conclusiones muy interesantes y fueron tomadas en cuenta. Estas conclusiones mencionan que el reconocimiento de la melodía requiere la inclusión de información sobre el ritmo, también que el reconocimiento y la predicción del ritmo es más difícil cuando no hay puntos de referencias claros y que la predicción de los experimentos de melodía muestra que la información del ritmo no ayuda al sistema de aprendizaje automático para predecir la melodía. Siendo estos tres puntos claves para crear mayor robustez en el algoritmo y mejorar los resultados que no fueron 100% estables (Szczerba, 1999).

También existe una investigación para clasificar géneros musicales poniendo en contexto la explicación de las estructuras musicales y al entendimiento del análisis percibido por los humanos. Tratando de resolver los problemas que se enfocan en la recuperación de información musical y especialmente en la clasificación automática de géneros musicales en plataformas interactivas, las sus bases de datos crecen exponencialmente, este artículo presenta un amplio enfoque en el aprendizaje automático utilizando las señales de audio proponiendo dos tipos de algoritmos de clasificación, el primero son los vectores de características o también llamados máquinas con vectores de

soporte con función de núcleo polinómico y otros algoritmos de ML los cuales son SVM y AdaBoost. Las características que se probarán en sus resultados son los de dominio de frecuencia, dominio temporal, dominio cepstral y frecuencia de modulación de dominio de audio. Los resultados son que SVM actúa con un fuerte rendimiento en base al aprendizaje de AdaBoost, concluyendo que SVM no puede mejorar utilizando el método de impulso, consiguiendo una precisión de clasificación de género del 78% y 81% en el género musical GTZAN sobre los demás diez géneros (Chathuranga & Jayaratne, 2013).

Similarmente se habla de una nueva aplicación desarrollada en el año 2010 en la misma Universidad Tecnológica de Gdansk en Polonia, la cual desarrollan una investigación que continúa con la publicada en 1999. Este artículo presenta un análisis exhaustivo de la clasificación automática aplicada a las señales musicales, basada en un conjunto elegido de algoritmos de ML. Su investigación constó de una base de datos de 60 compositores e intérpretes de música con el propósito de recolectar de 15 a 20 piezas de música, divididas en 20 segmentos para después dividirlas en parámetros de entrada. Sus resultados demuestran que los clasificadores que son capaces de obtener una separación de clases no lineal tenían más éxito en el reconocimiento. De los cuatro tipos de algoritmos el SMCC tuvo mejores resultados con un promedio de 72.33% de eficiencia, además al reducir la cantidad de géneros musicales al 15% se aumentó la precisión a un 88%. Finalmente se podría decir que es una aplicación para el reconocimiento de metadatos en los archivos de audio para clasificarlos automáticamente en una base de datos y que su objetivo general es obtener una eficiencia robusta y quedarse con el algoritmo con mejores resultados (Zwan & Bozena, 2011).

En el año 2015 en la Universidad de Kansas cuya su principal contribución es proponer un marco para modelar las tareas múltiples en un tiempo constante.

Siendo una investigación aplicada para redes neuronales en línea, lo cual es un verdadero desafío para el algoritmo debido a que el número de tareas múltiples es grande a comparación del número de muestras, el cual debe cumplir que todas las tareas que están relacionadas seleccionen un conjunto común de características. En ese documento se propone funciones de aprendizaje para representar las tareas relacionadas finalizando con funciones de aprendizaje más rápidas y menos costosas. Un tema un poco complejo para entenderlo ya que se realiza matematice muy avanzada para este tipo de algoritmos, pero se pueden entender teóricamente para obtener conclusiones convincentes al final de la investigación. Finalmente se demostró que el ML convencional no siempre mejora el rendimiento, siendo más específico si las tareas no se relacionan entre sí, el rendimiento del ML se deteriora. Es así que se desarrolló un algoritmo que agrupará la tarea relevante en conjunto relacionando las otras menos relevantes al seleccionar características comunes. Así mismo se demostró con éxito que su algoritmo mejora el rendimiento cuando la cantidad de tareas y funciones es más grande. A comparación de otras investigaciones coinciden los resultados al aumentar la cantidad de muestras (Mishra, 2015).

El intento de encontrar una mejora de la herramienta del ML ha sido uno de los mayores objetivos, un claro ejemplo de esto se encuentra en una investigación desarrollada en la Universidad Purdue en West Lafayette (EE. UU), la cual utiliza el ámbito de la semántica ontológica, utilizando técnicas basadas en ML para hallar patrones en datos más significativos. Esto significa que al tener entradas ya sean explicitas o implícitas, deben contener conceptos, relaciones y literales que ayuden a diferenciar datos más significativos y mejorar la idea del ML simple. Concluyendo que si se propusiera algún tipo de aplicación con datos de entrada muy amplios en un sistema exitoso MBML sería superior al de un sistema ML convencional (Falk y Stuart, 2016).

1. 2 Objetivos

1.2.1 Objetivo General

Proponer una metodología de clasificación de instrumentos de viento andinos a travéz de algoritmos de Machine Learning.

1.2.2 Objetivos Específicos

Analizar las precisiones obtenidas por cada algoritmo de Machine Learning utilizadas para clasificar instrumentos de viento.

Diseñar una base de datos adecuada para entrenar un sistema de clasificación de instrumentos de viento andinos a través de la extracción de información musical.

1.3 Alcance

Con la aplicación de la investigación se busca encontrar resultados de clasificación para un cierto número de instrumentos andinos de viento, mediante la clasificación diseñada para este tipo de sonidos. Para esto es necesario tener claro el planteamiento del problema, encontrar los instrumentos que tengan la capacidad de emitir los datos necesarios que se introducirán a la red para el correcto desarrollo de su entrenamiento y finalmente obtener como mínimo un 80% de exactitud a una clasificación real con la inducción de sonidos de otros instrumentos que tengan características semejantes; y así comparar los resultados con la interpretación óptima del desarrollador plasmada en sus conclusiones.

1.4 Justificación

En el Ecuador se han registrado aproximadamente 300 instrumentos elaborados por las distintas culturas ecuatorianas, las cuales tienen distintas

variaciones y han sido utilizadas por un sin número de expresiones musicales y culturales. Muchos de estos instrumentos se encuentran en la clasificación de instrumentos de viento y no se los ha dado un estudio exhaustivo por investigaciones objetivas que muestren las características acústicas y sonoras de cada uno de ellos. Es más no existe ningún registro de alguna investigación a través del uso de un programa con aplicaciones ML que sirvan para clasificar por sí mismo dichos instrumentos.

El algoritmo a ser desarrollado brindará un primer paso a la investigación de la programación de algoritmos ML y a la aplicación de clasificar instrumentos de viento simulando el proceso que un ser humano realizaría fácilmente en su vida cotidiana.

1. 5 Hipótesis

El resultado que se busca en la investigación es poder clasificar instrumentos de viento mediante un software ya existente que utilice algoritmos de Machine Learning, con ayuda de herramientas que permitan diseñar una base de datos óptima y finalmente poder validar los resultados de precisión de la clasificación con un método estadístico llamado validación cruzada.

2. CAPITULO II: MARCO TEÓRICO

La siguiente investigación conforma una cierta cantidad de información, la cual debe ser comprendida muy a fondo, el cual consiste en conformar una de las aplicaciones en el tema de la clasificación de sonidos mediante algoritmos ML. Es un tema que sin duda tendrá mucha importancia y significa un gran avance para la inteligencia artificial en los últimos años, así como avances en la extracción de su información con el objetivo futuro de realizar aplicaciones más complejas que realicen síntesis en tiempo real o el reconocimiento de sonidos en tiempo real. Finalmente, las aplicaciones desarrolladas en el mundo que

utiliza inteligencia artificial el mayor reto es poder desarrollar programas que tengan una propia interpretación de lo que se está clasificando y tomar decisiones respecto a sus resultados, logrando un avance muy importante que permitirá al mundo abrir puertas a un nuevo mundo de entornos inteligentes.

2.1 Machine Learning

Machine Learning puede ser ampliamente definida como métodos computacionales que usan la experiencia para mejorar el desempeño de las predicciones, logrando ser estas más precisas. Cuando se referimos a experiencias se habla específicamente de la información histórica recolectada que se utiliza para los procesos de entrenamiento (Mehryar et al, 2012).

Es así que se indican las aplicaciones que puede extenderse en el mundo del ML:

- Clasificación: La idea central de la aplicación de la técnica es la identificación de clases o tipos de pertenencia en la entrada del algoritmo, aquellos ejemplos aplicados en la clasificación de documentos, imágenes, diagnóstico médico, sonidos, etc.
- Regresión: Realiza predicciones en valores reales para cada ítem, como por ejemplo predicciones de demandas, stocks de inventarios, variables económicas, tasas, etc.
- Ranking: Se la utiliza para ordenar variables basadas en algún criterio, por ejemplo, búsquedas web.
- Clustering: Muy utilizado en aplicaciones basadas en procesos comerciales, con objetivos de segmentar clientes y productos de tal manera que faciliten los procesos de decisiones referentes a qué productos vender y a qué tipo de target vender.
- Reducción de Dimensionalidad: Realiza una transformación basada en la optimización de espacio, manteniendo las propiedades iniciales que

representan los datos de entrada, un ejemplo tenemos en el reprocesamiento de imágenes digitales, o la optimización de audio.

Tenemos otras tareas donde se encuentra factible la aplicación de ML, las cuales pueden ser el procesamiento de lenguaje natural, reconocimiento de sonido, Reconocimiento de Caracteres Ópticos (OCR), reconocimiento de rostro, etc.

Existe una gran variedad de algoritmos de clasificación, como por ejemplo los basados en lógica y árboles de decisión (*Decision Trees*), aprendizaje estadístico (*Neural Networks*) o distancia (*k-Nearest Neighbors*), redes neuronales, clustering (*k-means*), máquinas de vectores de soporte (*Support Vector Machines*), etc. En este trabajo explicaremos algunos de los ejemplos de Redes Neuronales para una ampliación a los algoritmos que utiliza Wekinator para clasificar sonidos.

2.2 Aprendizaje Automático o Machine Learning (ML)

El Aprendizaje Automático o Machine Learning tiene como objetivo el desarrollo de algoritmos que permitan crear aprendizaje sobre un conjunto o una serie de grupos de observaciones (*inputs*), de tal modo que sea posible establecer hipótesis generales o predicciones sobre nuevas prospecciones, para de este modo tomar decisiones automáticas, ungiendo así un sistema de Inteligencia Artificial (Mitchel, 1997).

El entrenamiento consiste en ser una fase de recolección de observaciones o ejemplos hacia el sistema, siendo estos los que proveerán de conocimiento y aprendizaje al programa. Es así que surgen dos tipos de aprendizaje del que se

explicará cómo fase introductoria para los algoritmos utilizados en la investigación:

- Aprendizaje Automático Supervisado.
- Aprendizaje Automático No Supervisado.

El primero consiste en proporcionar al sistema una serie de datos que son fáciles de identificar y el segundo tipo de Aprendizaje Automático no Supervisado no tienen etiqueta y terminan teniendo un comportamiento aleatorio pero inteligente, concretando algoritmos de agrupamiento o *clustering* que logran ensamblar nuevos patrones de clasificación hacia un nuevo conjunto de ejemplos que serían utilizados como ejemplos de prueba en la fase de evaluación. En el presente trabajo se utilizará únicamente el aprendizaje supervisado, el cual proporciona una etiqueta o una serie de etiquetas que serán utilizadas durante la fase de entrenamiento (Kuri, 2015).

2.3 Fase de prueba de un algoritmo ML

La fase de evaluación se realiza después de que el programa haya sido entrenado. Esta segunda fase del sistema consiste en proporcionar al sistema una pequeña muestra de datos para que el sistema entrenado realice su trabajo y enviar datos con una interpretación propia del comportamiento del algoritmo, así se crea un nuevo sistema de inteligencia artificial. Identificando las etiquetas que proporcionan los ejemplos de prueba, podremos identificar e interpretar los resultados de la evaluación, distinguiendo los aciertos y fallos que tiene el programa. De esta manera nacen dos tipos de interpretaciones en la estadística utilizada en los resultados de probabilidad, estas interpretaciones sirven para observar dos tipos de punto de vista en la aplicación de clasificación de las redes neuronales, estos puntos de vista son:

- Falsa Alarma: probabilidad que tiene una determinada clase de aceptar erróneamente datos correspondientes a otra clase.

- Falso Rechazo: probabilidad que tiene una determinada clase de rechazar erróneamente datos correspondientes a dicha clase.

Tratándose de un programa supervisado de ML de reconocimiento de espectros sonoros, requiere de algún tipo de extracción con características físicas o acústicas relevantes (*Feature Extraction*), esta extracción de características depende de la base de datos que se escoja para la fase de entrenamiento y la fase de prueba. A esto se le puede reducir como extraer las huellas digitales de cada clase de sonido que se quiera clasificar, un ejemplo de huella digital sería el espectrograma de frecuencia, el cual termina siendo una imagen que refleja las características más relevantes de los sonidos.

En la figura 1 se visualiza un esquema basado en una tesis magistral que indica el desarrollo y análisis de clasificadores de señales de audio, el cual refleja el proceso general del desarrollo de un clasificador y que describe un estándar de clasificación para obtener buenos resultados. El primer paso es determinar una base de datos (*database*). Después de definir la base de datos, se procesa cada uno de los ejemplos para extraer las características más significativas (*feature extraction*). El conjunto de *features* utilizadas, conformarán una observación de la base de datos en la primera fase denominada (*fase de entrenamiento*), la calidad de la base de datos determinará la eficiencia del proceso de entrenamiento del programa.

En Wekinator tenemos un conjunto de clasificadores, los cuales se explicarán en una siguiente fase, así que es conveniente elegir adecuadamente el algoritmo de clasificación a utilizar o saber combinar una serie de ellos para obtener mayor precisión, esa es la razón por la cual esta investigación es viable, ya que brindarán resultados a partir de la selección de cada uno de los clasificadores. Igualmente se procede a reservar una parte pequeña de la base de datos para definir un conjunto de evaluación, llegando a la siguiente fase

(*fase de prueba*) para caracterizar el clasificador. Finalmente, se reflejan los resultados del clasificador. En caso de obtener malos resultados, se tendría que proponer la adquisición de una nueva manera de obtener una base de datos de mejor calidad o con menos ruido y definición, estas nuevas *features* lograrán obtener información más relevante con el objetivo de mejorar los resultados de los clasificadores.

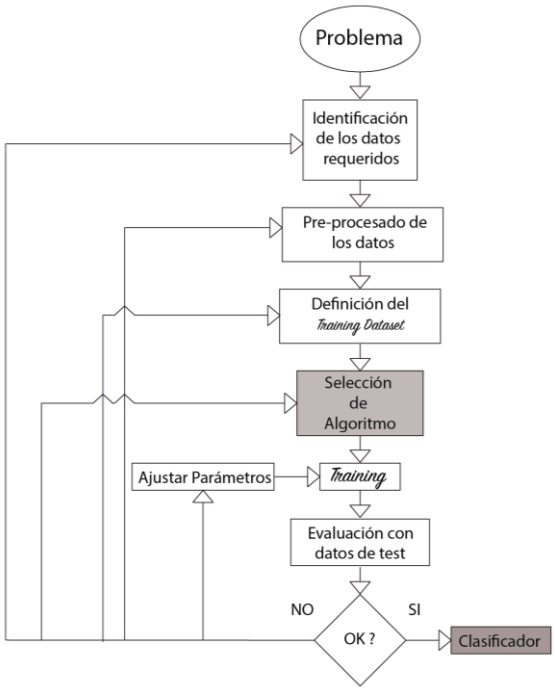


Figura 1. Flujo de trabajo de un clasificador de ML.
Adaptada de Aguirre, 2017.

2.4 Pre-Procesado: Feature Extraction

Como se había mencionado anteriormente nuestro programa necesita de un *pre-procesado* el cual debería estar disponible en función a la taxonomía que elijamos para analizar los ejemplos de entrenamiento. En el caso de características ligadas al audio, existen distintos tipos de taxonomías, las cuales únicamente serán mencionadas ya que ninguna de ellas será utilizada en el desarrollo del programa, las razones por las cuales considero que no se utilizan las diferentes taxonomías son:

- Optimizan mucha información que un procesador de cómputo convencional si puede manejar y procesar en tiempo real, como en el caso utilizado induce 16 000 entradas proporcionadas por el programa desarrollado en Processing y que no necesitan de optimización o promediado de características sonoras de los datos.

- La única rama de taxonomía que se asemeja a un espectrograma y que de hecho se basa en él, es en de Centroide Espectral (*Spectral Centroid*), siendo este un tipo de *feature* que aporta información acerca de la forma del espectro, concretamente define el centro de gravedad del espectro, es decir, considera el espectro como una distribución de probabilidad, la cual viene determinada por la amplitud del espectro para cada frecuencia, obteniendo así la frecuencia media ponderada en amplitud. Sin embargo; este tipo de taxonomía implica la extracción previa de una frecuencia fundamental, así como de sus armónicos, eliminando información valiosa para el entrenamiento reduciendo la resolución de datos los cuales en el audio son muy necesarios debido a que su distribución no es lineal.

Las taxonomías orientadas al audio, se pueden resumir en: temporales, de energía, espectrales, armónicas y perceptuales. En el siguiente apartado se explicará la base de las taxonomías, denominadas como Recuperación de Información Musical (*Music Information Retrieval*), el cual indicará en base a qué necesidades o aplicaciones se pretende realizar clasificaciones en base a características físicas de canciones o sonidos más complejos que existen en mercados o bases de datos con costo, como también se explicará un poco de la recopilación de los datos de entrenamiento utilizados en Wekinator para el desarrollo del sistema localizado, es decir se hablará de la recuperación de la información diseñado para clasificar instrumentos sin la necesidad de utilizar taxonomías.

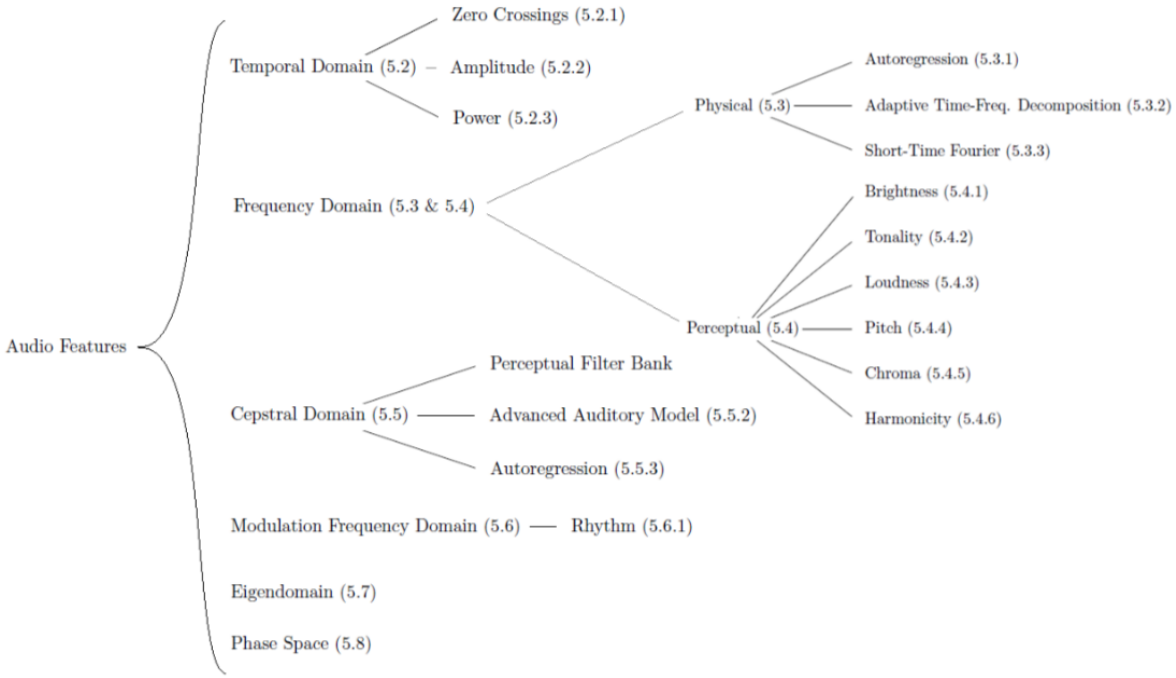


Figura 2. Taxonomías para Características de Audio.

2.5 Music Information Retrieval (MIR)

El MIR, es una nueva ciencia interdisciplinaria encargada de recuperar información de la música. Siendo un ascendente campo de investigación con muchas aplicaciones en el mundo real. Estas aplicaciones vienen siendo sistemas de recomendación, separación de pistas y reconocimiento de instrumentos, transcripción automática de música, categorización automática y generación musical. Está basada en metadatos que quedan registrados en los archivos de audio y en la información musical y tonal de los mismos.

2.5.1 Sistemas de recomendación musical (SDR)

En primer lugar, los SDR están basados en técnicas de recuperación de información, las cuales son varias. Estas técnicas se componen de una laboriosa recopilación de datos para después realizar semejanzas de reproducciones entre usuarios, también llamados sistemas de filtrado

colaborativos. Es decir que no existe ningún proceso de clasificación como es el caso del ML que utilizan algunas plataformas iterativas de reproducción musical. Una de las marcas que utiliza SDR es Pandora, el cual ha utilizado expertos para etiquetar la música con cualidades específicas como “cantante femenina” o “fuerte línea de bajo”. Otros sistemas utilizan en los historiales de reproducción de música en los usuarios para filtrar y sugerir nueva música. Estas nuevas técnicas están empezando a ocuparse como parte básica de las plataformas iterativas y no solamente con música, sino también con películas, series y documentales.

Es así que su uso principal está basado en el ámbito comercial, donde su tarea principal es encontrar elementos que el usuario estaría interesado en comprar, considerando que, si el sistema de recomendación es más eficiente, existirán mayores ganancias o terminará siendo un factor fundamental para aumentar su tendencia y demanda.

2.5.2 Identificación musical

Una de las aplicaciones más populares y comunes es la identificación musical, siendo este el enfoque basado en la información interna del archivo; es decir la función extraída, almacenada e indexada que lleva dentro, a esto se lo denomina *tabla de contenido*. Esta función es muy específica ya que representa las posiciones de inicio y longitud de las pistas en el disco, siendo esta característica fundamental ya que es muy difícil que un álbum coincida la misma longitud de pistas en el mismo orden que con otro álbum o canción.

Otro de los ejemplos que han optado por esta idea es Shazam (Wang, 2006), pero no solamente utilizan una tabla de contenido proporcionado por los metadatos, sino que está diseñado para adaptarse a cualquier ambiente con un ruido de fondo alto. La información que es utilizada para identificar música por Shazam son las diferencias de picos que tienen ciertos intervalos de tiempo de

la pista. Estos intervalos de los tracks evaluados se los denominan muestras, y no hacen falta muchas muestras para identificar canciones en estos casos, ya que es imposible que existan similitudes entre dos o más canciones canciones en las diferencias de amplitudes en cierto lapso de tiempo. En la figura 5 muestra el comportamiento del programa Shazam, el cual indica las muestras tomadas en lapsos de tiempo y sus diferencias de distancia.

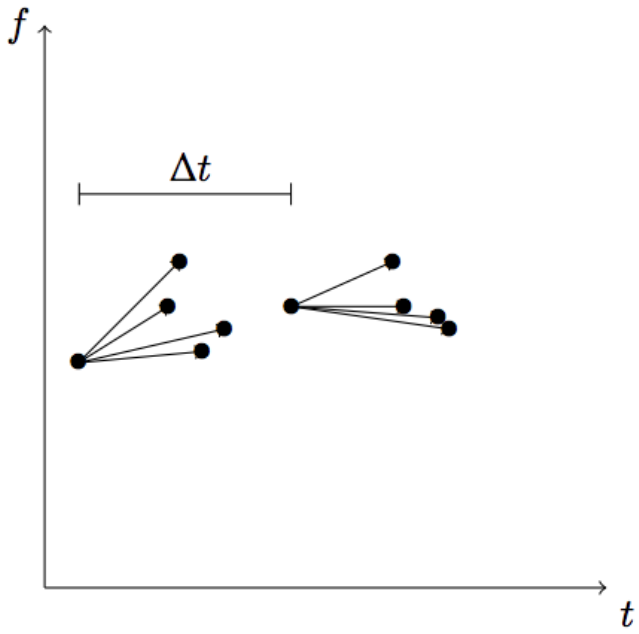


Figura 3. Gráfica de identificación musical de Shazam.

2.5.3 Análisis Musical

En los apartados anteriores del MIR se han explicado varias aplicaciones que muestran viarias maneras de extracción en la huella digital de cada canción para varias actividades. Existen otras ideas a través de un análisis distinto que son utilizadas para otro tipo de aplicaciones experimentadas y desarrolladas en actividades diarias del ser humano. Un avance de este nuevo análisis musical está dividido en dos partes la segmentación y la transcripción.

2.5.3.1 Segmentación

La segmentación es un tipo de análisis musical, el cual consiste en dividir coherentemente datos. Como por ejemplo se segmentan partes de una canción, en un caso serían los inicios y finales de frases o secciones importantes de dinámica utilizados en las máquinas de karaoke. Por otro lado se analiza los segmentos iniciales de las canciones, en los cuales se evalúan las diferencias significativas de los niveles bajos, expresándose así como procesamiento de señales. Es necesario conocer los niveles de la señal para poder evaluar la segmentación; es decir si el oyente puede escuchar la transducción o no.

En la siguiente figura se puede ver la segmentación que podría tener una frase o segmento de una canción, en donde existen barras separando beats y describiendo los eventos que suceden en la sección.

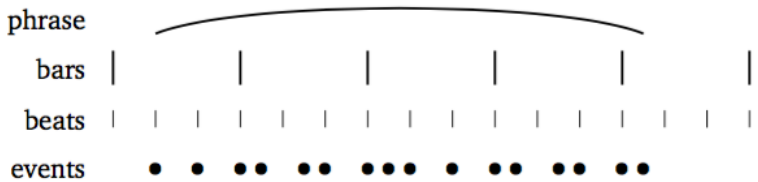


Figura 4. Segmentación de una sección musical.

Es así que existen enfoques en aplicaciones como la detección de pulsos o los beats por segundo que tiene una canción (*bpm*). Uno de los enlaces que se quiere lograr se encuentra en la detección de latidos. Estos algoritmos de detección pueden llegar a ser muy precisos debido a que si se escogen audios con elementos de percusión altos y fuertes se podría lograr una detección perfecta.

2.5.3.2 Transcripción

El poder transcribir la notación musical de una señal de audio de manera detallada es uno de los objetivos más complejos que tiene el MIR, ya que requiere de mucha complejidad y de una gran cantidad de pruebas, fallas y errores para llegar a buenos resultados. En el caso de esta investigación, encontrar la manera de transcribir características acústicas como el vibrato sin afectar a los resultados sería acercarse al 100% de efectividad en la función de la aplicación.

Se debe tomar en cuenta que todos los instrumentos que componen a una canción son totalmente diferentes, su contenido acústico es diferente al de un simple tono puro. Como se verá en la sección de ML basado en audio, la idea básica en el análisis de audio monofónico es identificar picos de intensidad en el espectrograma y compararlos con sus adyacentes hasta llegar a relacionarlos con su fundamental para obtener datos que se puedan interpretarlos.

Es así que para realizar una transcripción automática a un nivel de precisión similar a la de un humano entrenado en la música polifónica, no solamente se necesita de mayores recursos computacionales, sino se requiere una refinada codificación que se adapte a las expectativas previas.

2.5.3.3 Espectrograma frecuencial de audio

Las ondas sonoras son literalmente unidimensionales; es decir que en cada lugar del tiempo se encontraría un valor numérico basado en la altura o el nivel que se encuentre la onda (*muestra).*

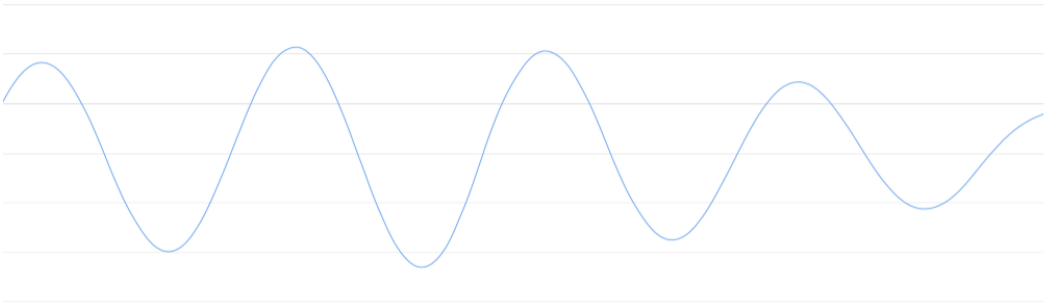


Figura 5. Onda sonora.

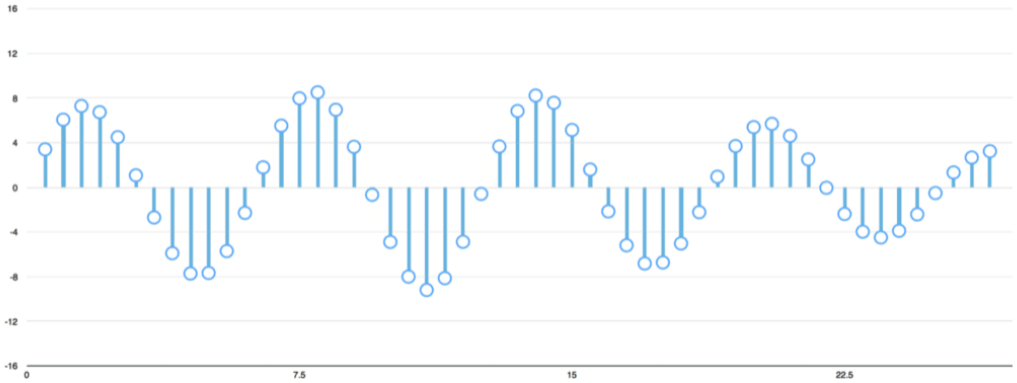


Figura 6. Onda Muestreada.

El audio digital en una buena calidad estándar se muestrea a una frecuencia de 44.1 kHz (44.100 lecturas por segundo). Así mismo es suficiente utilizar una frecuencia de muestreo de 16 kHz (16,000 muestras por segundo), las cuales son suficientes para cubrir un rango de frecuencia del habla humana.

Como ejemplo en la figura 9 se puede utilizar las 100 primeras muestras de una onda de sonido “hola” registrada o grabada. Siendo cada número representando la amplitud de onda en intervalos de 1/16000 segundos, cada dígito representa una cantidad de amplitud en función a la frecuencia para construir una huella digital de ese sonido.

```
[-1274, -1252, -1160, -986, -792, -692, -614, -429, -286, -134, -57, -41, -169, -456, -450, -541, -761, -1067, -1231, -1047, -952, -645, -489, -448
, -397, -212, 193, 114, -17, -110, 128, 261, 198, 390, 461, 772, 948, 1451, 1974, 2624, 3793, 4968, 5939, 6057, 6581, 7302, 7640, 7223, 6119, 5461,
 4820, 4353, 3611, 2740, 2004, 1349, 1178, 1085, 901, 301, -262, -499, -488, -707, -1406, -1997, -2377, -2494, -2605, -2675, -2627, -2500, -2148, -
1648, -970, -364, 13, 260, 494, 788, 1011, 938, 717, 507, 323, 324, 325, 350, 103, -113, 64, 176, 93, -249, -461, -606, -909, -1159, -1307, -1544]
```

Figura 7. 100 primeras muestras de un audio que dice “hola”.

Ahora al tener una muestra, la cual simplemente es única en el mundo, casi imposible de copiar por la cantidad de datos recopilados, como una huella digital sería muy difícil de recopilar información acerca de sus armónicos, texturas, etc. Para tenerlo más claro en la figura 10 tenemos un ejemplo más detallado el cual tiene 20 milisegundos de audio, el cual al tener más tiempo de reproducción tendrá más datos y será un sonido con más información.

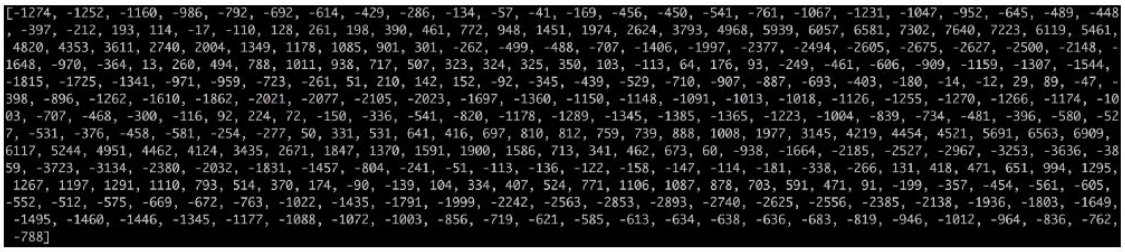


Figura 8. 320 muestras del audio “hola”.

Dándonos como resultado una aproximación de la onda original de sonido durante ese periodo:

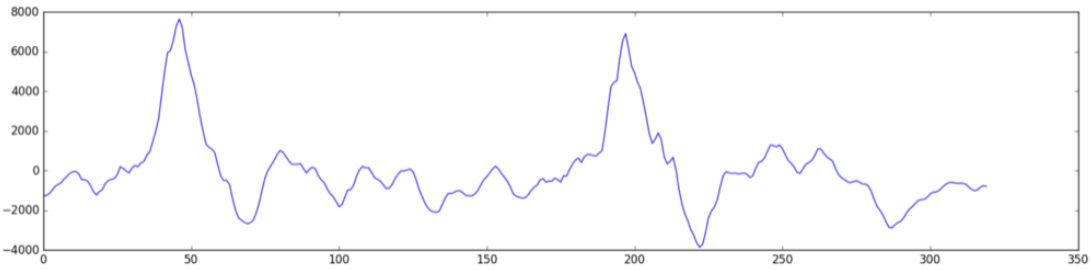


Figura 9. Onda Aproximada del audio “hola” en un periodo de 20 milisegundos.

Al resumirlo tenemos una relación de 1:50 de un segundo de tiempo. Se podría decir que es un registro literalmente insignificante, pero es un caso complejo de muchas frecuencias de sonido interactuando entre sí para generar un timbre único (*huella digital),* y simplemente con el método de medir amplitudes es imposible de identificar.

Para poder introducir los datos a una red neuronal y poder alimentarla se debe desglosar las partes acústicas tonales en donde se puedan diferenciar todas

las frecuencias y sus amplitudes en función al tiempo. En otras palabras, debemos realizar una operación matemática llamada "*transformada de Fourier*" para identificar la cantidad de energía que existe en cada una de las bandas de frecuencia.

Imaginamos una grabación de un instrumento andino de viento que tenga la capacidad de reproducir un acorde mayor como por ejemplo Do mayor, el cual tiene la combinación de tres notas musicales, Do, Mi y Sol (C, E, G), todas interactuando generan un sonido complejo que con el método anterior de amplitud sería imposible de identificar. Es así que para dividir debemos romper la compleja onda de sonido para contemplar sus componentes. Como resultado tendríamos una base de datos similar a la figura 12 pero con la diferencia que se obtiene una puntuación representando la cantidad de energía que hay en cada banda de 50 Hz en un clip de audio de 20 milisegundos.

Figura 10. Clip de audio de 20 milisegundos.

Gráficamente no solamente se obtendría una simple frecuencia de 50 Hz sino de todo un rango de frecuencias, las cuales brindará una información valiosísima al clasificar instrumentos o cualquier tipo de audio. Tenemos el siguiente ejemplo el cual gráfica la cantidad de energía en un fragmento o (bit) de sonido en todo el panorama de una cantidad de muestras en 20 milisegundos:

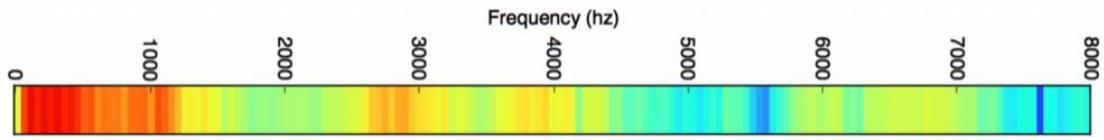


Figura 11. Fragmento de sonido de 20 milisegundos indicando la cantidad de energía en esa muestra.

Analizando las frecuencias que pertenecen al acorde de Do Mayor tenemos identificado nuestro acorde en la gráfica anterior.

Tabla 1
Frecuencias del acorde de Do Mayor

Nota	Do	Mi	Sol
Frecuencia (Hz)	261	329	391

Como resultado la información coincide con los puntos rojos pertenecientes a las frecuencias posteriores a las bajas, indicándonos la información que se requirió anteriormente. Si repetimos el procedimiento en función al tiempo con el ejemplo anterior del audio “hola” tendríamos una gráfica que se puede interpretar como tridimensional, obteniendo información de amplitud, frecuencia y periodo.

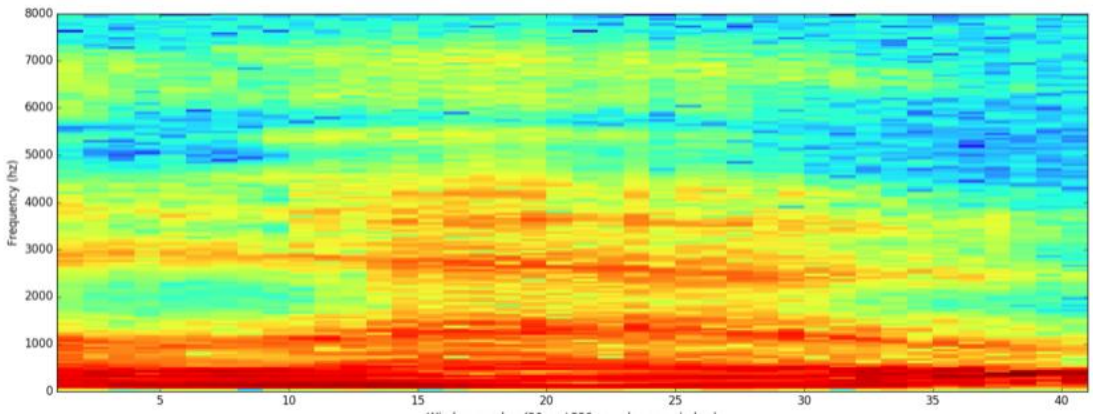


Figura 12. 320 Samples por ventana (frecuencia).

2.6 Clasificación

Como se había mencionado existen algunas aplicaciones ligadas con las posibilidades del ML, pero en este trabajo se verá únicamente la clasificación, es así que hablaremos acerca del programa que realizará el trabajo de clasificar nuestros datos, comenzando por su estructura para manejar algoritmos de clasificación de tal manera que sea simple para manejar en caso

de que tengan aplicaciones artísticas o aprender a visualizar el funcionamiento del aprendizaje automático (Aguirre, 2017).

La herramienta que veremos tiene el nombre de Wekinator, es un programa que es basado en una plataforma de aprendizaje automático llamado Weka. Para entender las bases de este sistema, se trata de un entorno utilizado para el análisis del conocimiento de la Universidad de Waikato, desarrollando sus datos en Java y convirtiéndose en un software libre distribuido bajo la licencia GNU-GPL.

El paquete Weka contiene una colección de herramientas de visualización y algoritmos para analizar datos y modelado predictivo, estas herramientas se unen con una interfaz gráfica de usuario para acceder fácilmente a sus funcionalidades. La versión original se diseñó inicialmente para analizar datos procedentes del dominio de la agricultura como objetivo de predicciones, actualmente se la utiliza en muchas áreas en particular con finalidades docentes y de investigación como es el caso actual.

2.7 Wekinator

El flujo del programa para realizar modelos de aprendizaje automático se visualiza en la figura 15, la cual indica que deben existir datos de entrada, varios modelos o algoritmos que se manejan y sincronizan entre sí para poder realizar predicciones en valores de salida que puedan ser interpretados por el usuario o desarrollador.

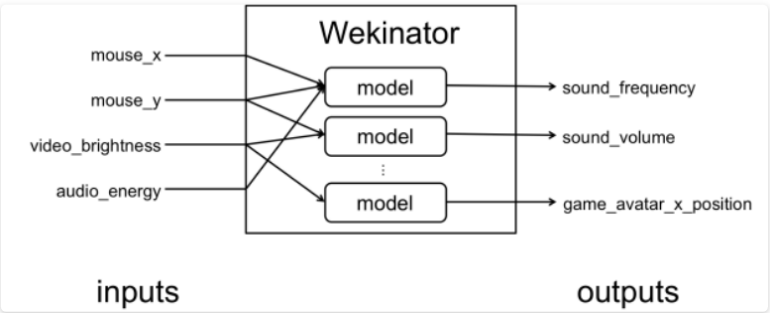


Figura 13. Diagrama de flujo de trabajo de Wekinator.
Tomada de Fiebrink, 2009.

Wekinator es una herramienta fácil de usar, creada para realizar aplicaciones artísticas, siendo una herramienta muy utilizada para compositores, artistas, músicos y otras personas creativas para utilizar ML en sus carreras profesionales. Permite utilizar una serie de algoritmos "estándar" de aprendizaje automático sin realizar ningún tipo de programación y permite conectar o comunicar con todo tipo de software que artistas o músicos ya están utilizando actualmente.

Una de las ventajas que facilita el Wekinator es que puede realizar operaciones en tiempo real, siendo fácil de grabar datos de entrenamiento en tiempo real y ejecutar modelos de prueba *in situ*.

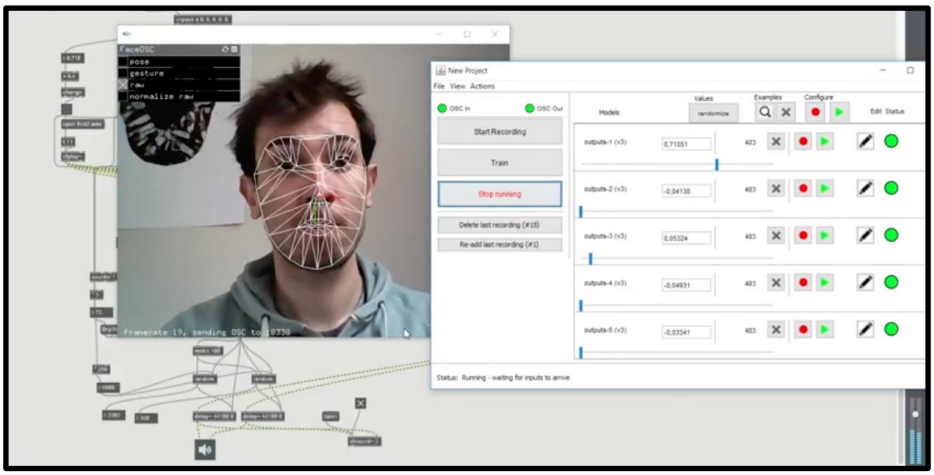


*Figura 14***.** Ejemplo de aplicación utilizando un programa de reconocimiento facial.
Tomada de Fiebrink, 2009.

En el siguiente apartado se explicará un ejemplo de clasificación en una plataforma denominada “*pista de baile*”, que fue desarrollado en un entorno de programación con lenguaje Java llamado Processing. Este programa se llama “*Classifier Explorer*”, teniendo un recuadro de coordenadas para ingresar datos de entrada. Este ejemplo permitirá introducir una primera experiencia con ejemplos gráficos de cómo realizar el proceso de ML en un programa diseñado para tener esta experiencia de forma gráfica y en tiempo real.

2.7.1 Clasificadores de Wekinator

Para empezar los entornos de programación utilizados para las aplicaciones de Wekinator son Processing, OpenFrameworks, o Quartz Composer las cuales se enlazan con Java utilizando sensores de entrada como la webcam, kinekt, arduino, el *mouse*, max, etc. Muchos de estos entornos han creado pequeños programas o plug-ins que permitirán entender una parte del funcionamiento de Wekinator, como es el caso del Clasification Explorer, siendo este el que permite desarrollar ejemplos de entrenamiento de clasificación en Wekinator, así como su configuración y teoría acerca de los clasificadores.

En el mundo artístico los clasificadores vienen siendo una herramienta para crear nuevos sonidos y poder experimentar música con extracción de datos, esto lo fue discutido en el apartado de MIR. Pero por el contrario se puede decir que los clasificadores también tienen aplicaciones más específicas y requieren de menos ruido para encontrar una precisión alta en vez de una precisión que se acople al mundo real.

2.7.1.1 Ejemplo de clasificación de Wekinator

En este apartado indicaremos el funcionamiento del programa para realizar una simple clasificación con ayuda de un ejemplo desarrollado desde Processing para enviar ejemplos de entrenamiento en una plataforma de dos dimensiones llamado “*Classifier Explorer*”. Después indicaremos los modelos matemáticos

de los clasificadores que utiliza Wekinator para sus clasificaciones, también se explicará cómo se comporta Wekinator para poder utilizar varios de ellos al mismo tiempo sin que ocurran errores en los resultados. A continuación, el ejemplo:

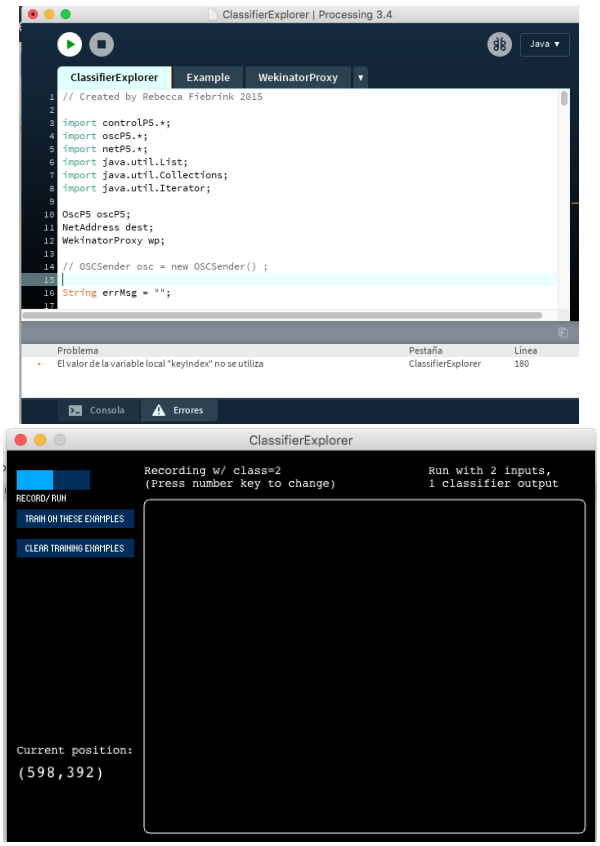


*Figura 15***.** Classification Explorer desde Processing.

Una vez ejecutado el programa CE, procedemos a ingresar la cantidad de datos de entrada que ingresan al Wekinator. En este caso tenemos dos elementos de entrada, los cuales son las coordenadas o dimensiones que tendría cada muestra, las cuales serán los datos que el Wekinator entrene para obtener los datos de salida del clasificador “CE”. La cantidad de salidas termina siendo una sola, la cual indica que clase de muestra se estaría registrando dentro del Wekinator, seleccionando esta cantidad igual a cinco. Como se ve en la siguiente figura se tiene como configuración del ejemplo, dos entradas, una salida y cinco clases.

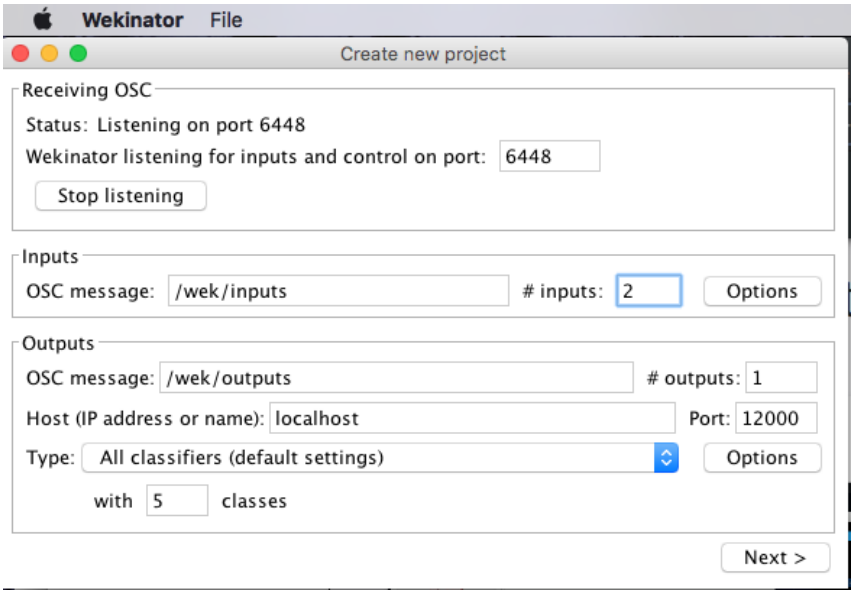


*Figura 16***.** Configuración de un ejemplo base de clasificación en Wekinator.

La siguiente figura muestra cómo se vería Wekinator al seleccionar la opción Next, para poder abrir el "*Classifier Explorer*" para ingresar los datos de entrada, entrenarlos con sus correspondientes clases y analizar la sección de trazo que dibujara el entrenamiento de Wekinator, plasmandolo en el escenario de CE.

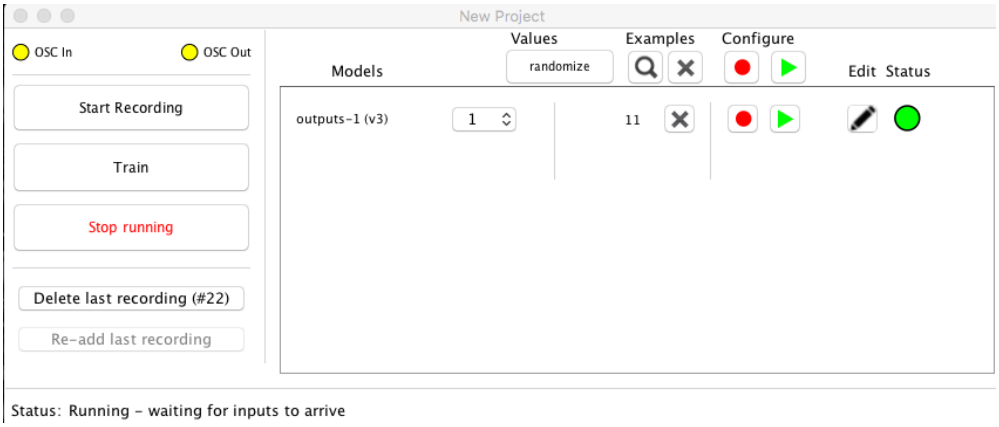


Figura 17. Visualización de Wekinator en el clasificador de ejemplo.

Al introducir los datos de entrada, Wekinator tendrá una base de datos para poder clasificar puntos nuevos de prueba, así lo indicamos en la figura 18.

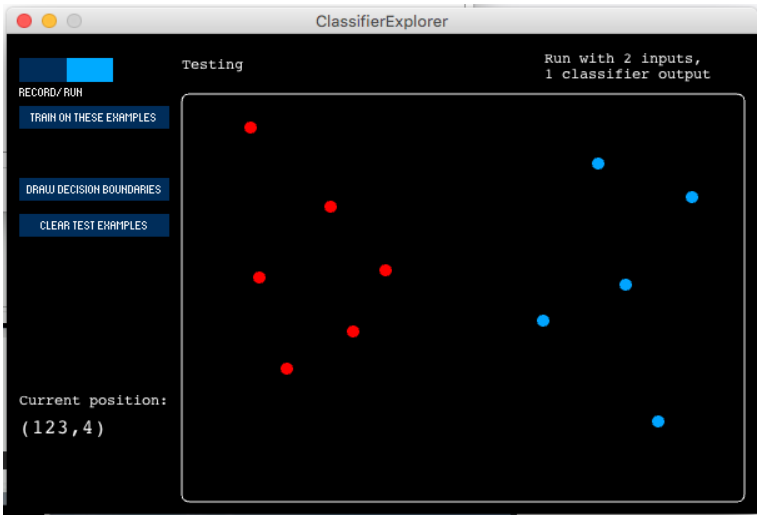


Figura 18. Base de datos con dos clases en Classifier Explorer. Tomada de Fiebrink, 2019.

Al correr el programa se empieza a colocar nuevos puntos los cuales ya están siendo clasificados. La figura 19 verifica cómo sería el resultado de una clasificación de dos clases.

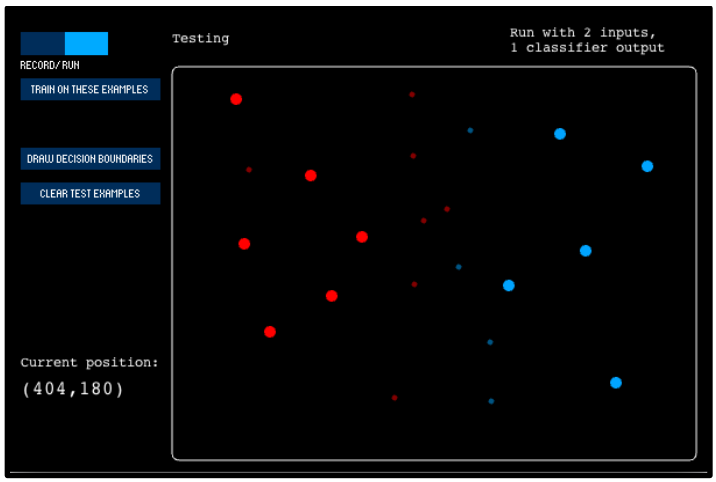


Figura 19. Clasificación del entrenamiento realizado por Wekinator. Tomada de Fiebrink, 2019.

Para poder visualizar de mejor manera el perímetro o dominio que ha clasificado el programa, "*Classifier Explorer*" tiene la opción de dibujar el dominio para analizar cómo se ha comportado el contorno del clasificador.

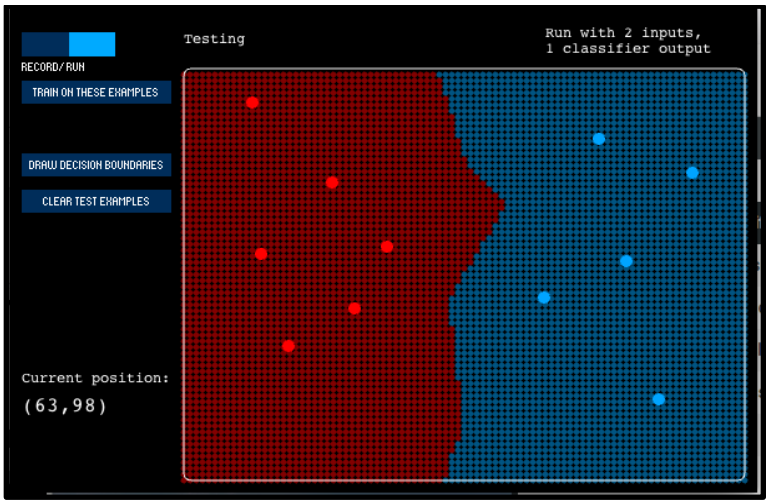


Figura 20. Contorno construido por el entrenamiento desde Wekinator. Tomada de Fiebrink, 2019.

Una vez entendido cómo se maneja la clasificación en Wekinator se debe preguntar, cómo realiza ese tipo de predicciones, que ecuaciones o qué algoritmos se manejaría para obtener estos resultados; es así que esta parte del documento se podría decir es la más importante para poder entender ese tipo de procedimientos y tener la intuición de que tipo de clasificador utilizará Wekinator para obtener distintos tipos de resultados y que se podría recomendar para mejorar.

Los modelos matemáticos presentados a continuación vienen siendo utilizados durante mucho tiempo para aplicaciones que manejan datos numéricos, como por ejemplo predecir posiciones, movimientos, comportamientos lineales y no lineales; es así que se debe tomar en cuenta que estos clasificadores serán utilizados para clasificar sonidos, por esta razón se ha explicado la manera en la cual se extraen los datos para poder entrenar al programa en el apartado de espectrograma frecuencial en la parte de análisis musical y combinar esa información con el tipo de procesamiento que se pueda dar a esos datos, con el objetivo de obtener buenos resultados y encontrar la manera de mejorar los mismos. A continuación, los clasificadores:

2.7.1.2 *K* - Nearest Neighbors

En esta sección no mostraremos las pautas para construir un buen clasificador, sino entender de manera práctica los algoritmos que resuelven una gran variedad de problemas en el mundo real, y es probable que se encuentre estos mismos algoritmos en numerosos proyectos creativos de música y arte, transformando las herramientas que estos algoritmos tienen para darle un buen uso general y convertirlas útiles para aplicaciones creativas.

A continuación, tenemos un ejemplo el cual contiene un material que es realmente específico, reproducido tal y como se desearía visualizarlo en un curso de ML real al uso de aprendizaje automático en contextos creativos y utilizando el aprendizaje automático para construir sistemas interactivos. Inicialmente todo algoritmo pasa por una pregunta fundamental, *¿Cómo se construye un modelo a partir de datos?* Cómo tomamos algunos ejemplos que ilustran algo sobre el mundo o promover problemas que ayuden a crear un modelo y utilizarlos para generalizar dichos problemas en el mundo real. O reproducir un modelo para hacer predicciones sobre nuevos datos a partir de algunos ejemplos.

Esta termina siendo una pregunta muy compleja en donde se incluye un concepto filosófico en el cual se visualiza en el aprendizaje humano y la percepción humana. Es así que se debe tener una idea de cómo los algoritmos pueden responder a este tipo de preguntas, teniendo una imagen mental de los pasos que debe atravesar un algoritmo para construir un modelo a partir de datos que sea útil para tomar decisiones inteligentes sobre qué algoritmo utilizar y darle un motivo del por qué seleccionar un modelo en específico para solucionar los problemas que se tendrían en los diferentes casos de clasificación. Como por ejemplo que se debe hacer si un clasificador sigue cometiendo errores, o si el entrenamiento toma mucho tiempo.

El ejemplo que hablaremos es un poco concreto acerca de algo que supondremos desearíamos tener en un algoritmo de aprendizaje automático. Así mismo el tener una buena idea sobre lo que realmente importa es importante si encontramos en un plano de razonamiento sobre cuál algoritmo de aprendizaje podría ser mejor para la aplicación que desearíamos aplicar.

Se empieza mostrando un conjunto de datos de entrenamiento utilizando una visualización que permita identificar cada uno de ellos para tener un razonamiento de que es lo que sucede con diferentes tipos de clasificadores. Se visualiza en la figura 21 que se tienen dos funciones de entrada *X* e *Y*, cada posición en este espacio podría ser la posición de un bailarín en un escenario (Analogía para identificar vectores). Se tiene una figura que tiene forma de *"x"* la cual está etiquetada como clase uno y con una *"o"* si es etiquetado como clase dos, para mostrar el comportamiento de tres clasificadores diferentes para este conjunto de entrenamiento.

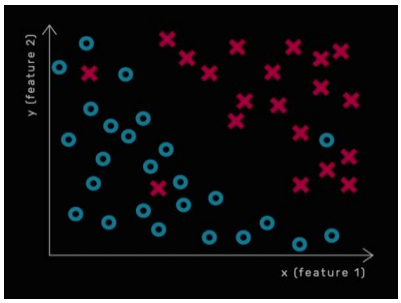


Figura 21. Función de datos.
Tomada de Fiebrink, 2019.

Como podemos ver en la siguiente imagen, se pueden visualizar los tres tipos de clasificadores, ahora es momento de escoger cual serviría más en tres diferentes casos de la vida real, más no encontrar cual sería el "mejor" clasificador o cual gustaría más. Es así que podría depender de la aplicación que le podríamos brindar a cada uno, por eso es que la respuesta termina siendo complicada a la hora de escoger cual es el mejor clasificador. Para esto empezamos a analizar cada uno de los casos.

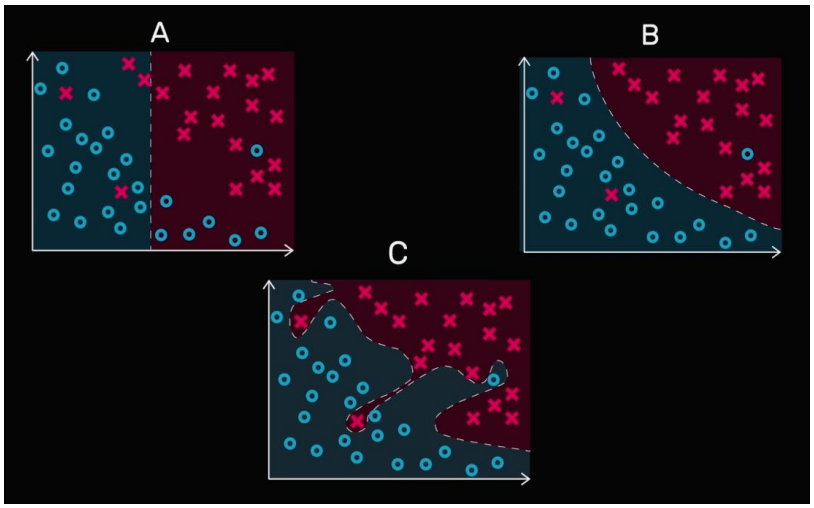


Figura 22. Tres tipos de clasificadores.
Tomada de Fiebrink, 2019.

En el clasificador A podemos concluir que parece ignorar descaradamente mucha información en el conjunto de datos de entrenamiento, se podría asumir que es algún tipo de clasificador que realiza algún tipo de predicción de que es lo que podría suceder en el futuro, reflejando que muchos de estos puntos futuros están mal. Este clasificador parece haber sido creado por un algoritmo inflexible que no tiene la capacidad expresiva o la complejidad requerida para reflejar el patrón que se observa pasa en los datos de entrada.

A continuación, se analiza el clasificador C, se observa que ninguno de los datos clasificados falla al caer en el lado contrario de la pista, esto quiere decir que al querer encontrar cada clase de dato el clasificador lo identificaría sin ningún error. Esto significa que tiene un 100% de precisión de entrenamiento logrando un complejo desarrollo de trabajo limitando la decisión de forma muy ondulada.

Comparándolos con el clasificador B cometería algunos errores en los ejemplos de entrenamiento si los volviera a ver en el futuro. Eso corresponde a menos

del 100% de precisión de entrenamiento, pero por lo general lo está haciendo bastante bien en la captura del patrón básico que vemos en los datos, logrando un trazado suave como en el ejemplo A, pero con la diferencia que hace un trabajo mucho mejor en caracterizar con precisión el patrón que vemos pasar donde las *x* tienden a estar dentro de la región curva.

Si queremos tomar uno de los clasificadores entre el ejemplo B y C y utilizarlo dentro de un sistema del mundo real y usarlo para clasificar nuevas entradas, ¿cuál es el ejemplo más probable en darnos el resultado más preciso en futuras clasificaciones?, es aquí donde se convierte en una pregunta difícil pero más definida entre lo que se planteaba al inicio. Es así que el clasificador más preciso dependerá simplemente de cómo interpretar su comportamiento con los ejemplos de entrenamiento ingresados.

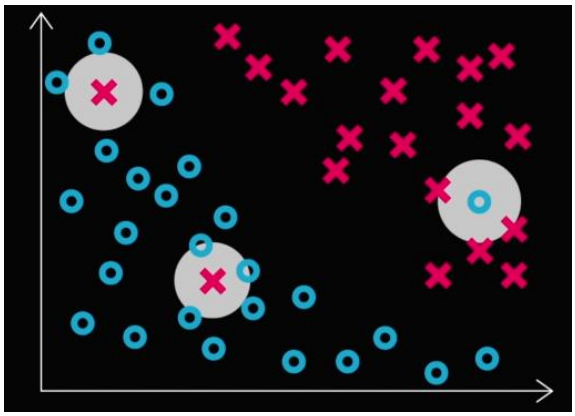


Figura 23. Errores en el entrenamiento.
Tomada de Fiebrink, 2019.

Complementando el caso de clasificación indicaremos las posibles razones por las cuales existen errores en el ingreso de datos para poder interpretar de mejor manera la precisión. En primer lugar, podemos interpretar el resultado de estos extraños entrenamientos como una existencia de ruido en el sistema, ahora imaginando que se está utilizando este conjunto de datos para construir un segmento principal de escenario para la clasificación de datos es donde cambia la idea de si en verdad el ejemplo C sería el que tenga mejores resultados pues estaríamos equivocados, la razón se encuentra en el rastro

que dejan los dos ejemplos. Como podemos ver en la figura 24 tenemos el rastro que deja el ejemplo B y podemos ver que tendríamos dos valores por cada sección con errores, pero si logramos visualizar la manera que clasifica el ejemplo C encontraríamos muchos más errores a la hora de clasificar las nuevas muestras debido al rastro que ha dejado en la pista.

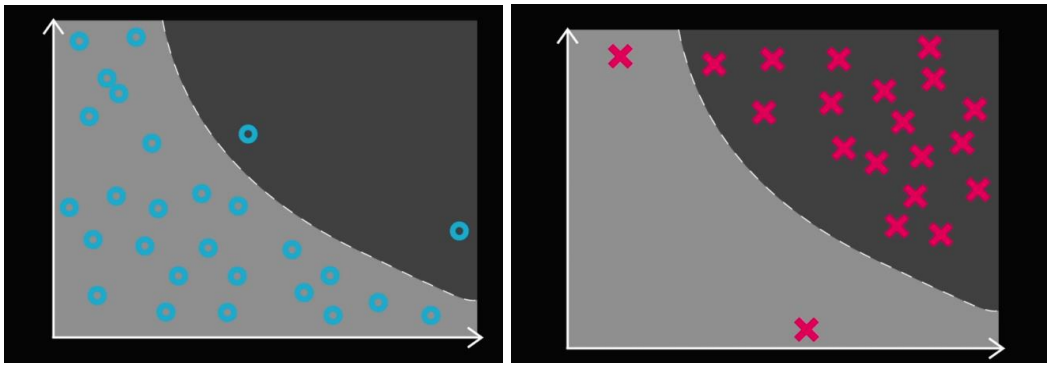


Figura 24. Nuevos datos de entrada para interpretación.
Tomada de Fiebrink, 2019.

La siguiente figura indican la diferencia de cómo clasificarían los dos ejemplos. Además de posiblemente tener ruido nuestros ejemplos de entrada se podrían interpretar como una representación de rasgos imperfectos de otros datos de entrada como podría funcionar en aplicaciones del mundo real.

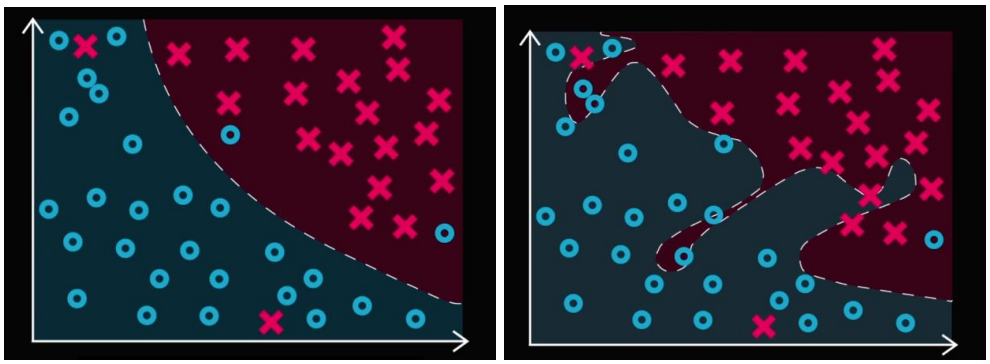


*Figura 25***.** Resultados de la clasificación del ejemplo B y C.
Tomada de Fiebrink, 2019.

Una vez visto cómo poder interpretar resultados entre simples ejemplos de clasificación, ahora aplicaremos matemática en el comportamiento de la clasificación en Wekinator y una de las herramientas que brinda el programa para realizar aproximaciones, la constante *k*, la cual indicarán varias maneras de predicción según el "*vecino más cercano*".

Para iniciar tomemos el ejemplo de una sola muestra la cual se encuentra mirando a través de todos los ejemplos de entrenamiento encontrando el más cercano y luego asumiendo que el nuevo punto de datos tendrá la misma clase que el ejemplo de entrenamiento más cercano. En este ejemplo bidimensional que encontramos en la figura 26 podemos calcular la distancia entre dos ejemplos de la misma manera que se haría en el mundo real, la cual la llamamos como distancia euclidiana y se puede escribir una ecuación para medir esa distancia entre los puntos X1, Y1 y X2, Y2.

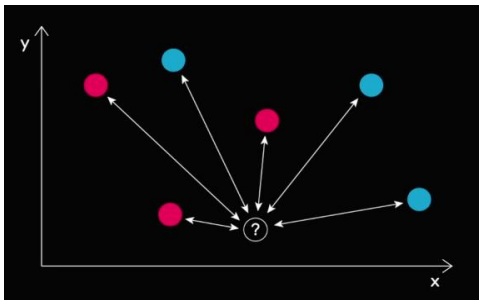


Figura 26. Interacción con todos los puntos.
Tomada de Fiebrink, 2019.

Este ejemplo se puede asumir como un simple caso del teorema de Pitágoras que se utiliza para calcular la longitud de una hipotenusa del triángulo rectángulo, utilizando la longitud de los dos lados.

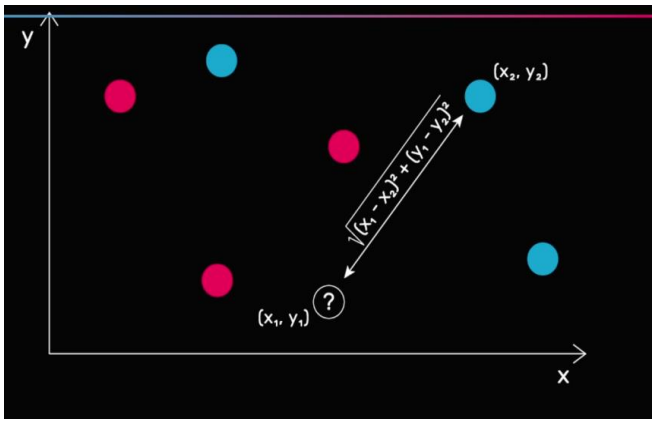


Figura 27. Distancia Euclidiana.
Tomada de Fiebrink, 2019.

Recordemos que podemos generalizar este cálculo de la distancia a un mayor número de funciones de entrada. Si tenemos tres características de entrada, podemos usar la fórmula para distancia euclidiana en tres dimensiones, la cual podría ser utilizada para medir la distancia entre dos objetos en un mundo tridimensional. Es así que podríamos generalizar el número de dimensiones como se visualiza en la siguiente fórmula.

$$dist = \sqrt{\sum_{f=1}^{d} (x_{fi} - x_{fj})^2}$$ Ecuación 01

Claro que no estamos tratando de comparar una distancia literal en el espacio físico entre un ejemplo y otro, sino que se trata de construir una pieza interactiva de performance, como por ejemplo se podría medir constantemente la frecuencia cardíaca de un artista intérprete o ejecutante, la velocidad de movimiento de su pie derecho, y el volumen al que está gritando. Si está actuando de una manera lenta, digamos que eso es una clase, si está realmente activo y emocionado, eso es clase dos, así que tengo tres actividades que hacen referirse como coordenadas en un espacio físico real dándole algo de sentido a la distancia euclidiana según la intuición que algunos seres humanos podrían tener.

Si *x* es su ritmo cardíaco, *y* es la velocidad de su pie derecho y *z* el volumen de su voz, entonces puedo usar esta fórmula para computar mi intérprete, y si encuentro un ejemplo que sea similar la distancia total será pequeña o muy grande. Ahora algunos de estos casos podrían salir mal y se recomienda tener cautela cuando se utiliza el vecino más cercano. En primer lugar, se considera el caso donde las características tienen magnitudes muy diferentes, donde el ritmo cardiaco se mide en minuto, la velocidad del pie se mide en segundos y el volumen se utiliza una métrica de audio llamada RMS.

En el siguiente ejemplo tenemos las magnitudes muy diferentes y un nuevo ejemplo para evaluar y predecir qué clase de comportamiento tiene el intérprete. Pudiendo evaluar que el nuevo ejemplo de entrenamiento está justo casi igual en el medio de ambos ejemplos iniciales, la velocidad del pié es exactamente igual a la del primer ejemplo y RMS tiene el mismo valor del primer ejemplo.

Tabla 2
Ejemplo de entrenamiento

Ejemplos de entrenamiento	Ritmo cardiaco (bpm)	Velocidad del pié (m/s)	Volumen (RMS)	Clase
1	60	0,01	0,05	1 (Calmado)
2	100	0.5	1,8	2 (Excitado)
Nuevo	80.5	0.01	0.05	??

Es así que como humanos podríamos decir razonablemente que este nuevo ejemplo es más similar al ejemplo de entrenamiento para la clase uno; como sea esto no es lo que indica la distancia euclidiana que da el siguiente resultado:

Distancia entre el nuevo ejemplo y el ejemplo 1:

$$\sqrt{(80{,}5-60)^2+(0{,}01-0{,}01)^2+(0{,}05-0{,}05)^2}=20{,}5$$

Distancia entre el nuevo ejemplo y el ejemplo 2:

$$\sqrt{(80{,}5-100)^2+(0{,}01-0{,}5)^2+(0{,}05-1{,}8)^2}=19{,}55\ (más\ cercano)$$

Mirando la ecuación para distancia euclidiana vemos que la diferencia entre cada valor de la característica es ponderada por igual y porque la ponderación que elegimos en cada caso es muy grande para lo que sería la frecuencia cardiaca y números muy pequeños para la velocidad del pie y RMS el ritmo cardiaco siempre va a terminar siendo mucho más importante a la distancia total calculada. Es así que cada magnitud que se implemente en algún caso la

diferencia en la característica magnitud variaría de diferentes maneras, para muchos de ellos no hará ninguna diferencia, pero el vecino más cercano será impactado muy fácilmente por eso. Es así que en la práctica a menudo se debe normalizar cada característica para que caigan dentro de un rango relativamente similar, como es el caso de los decibeles en el entrenamiento del programa en la presente investigación.

Hay varias maneras de encontrar el vecino más cercano de manera más precisa, utilizando una constante denominada *k-vecina* más cercana que tiene Wekinator dentro de su sistema. *k* es un parámetro, algo así como un mando virtual que se puede ajustar para ajustar el rendimiento del clasificador y *k* es solo un número entero que especifica cuántos de los vecinos más cercanos se estaría utilizando para tomar una decisión. Por ejemplo, si pongo *k* igual a tres el cual indicará que se utilizaran tres ejemplos para promediar cual sería el más cercano, en este caso cada vez que veo un nuevo punto busco entre los tres puntos un nuevo espacio principal, cualquier clase es compartida por la mayoría de esos vecinos sería la elegida por el clasificador como podemos visualizar en la figura 28.

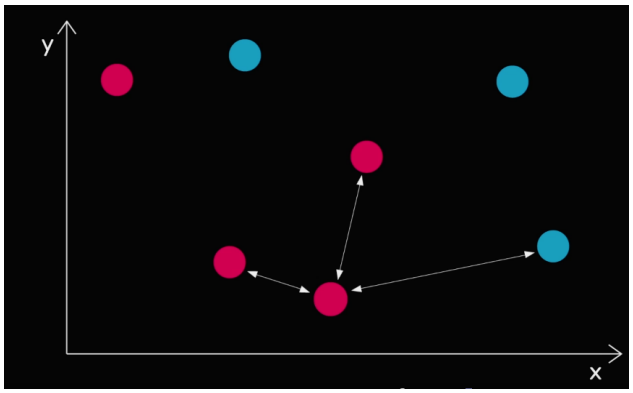


Figura 28. Ejemplo de promedio entre tres ejemplos.
Tomada de Fiebrink, 2019.

Utilizando Classifier Explorer se puede visualizar un ejemplo de conjunto de entrenamiento que se podría dar a *k*-clasificador vecino más cercano. Se asume que se va a utilizar *k* igual a uno, y se considera lo que podría pasar si

se le coloca un nuevo punto en una zona donde la mayoría de puntos son de otro color, como en la siguiente figura que se visualiza un punto rojo entre varios puntos azules y al tener *k* igual a uno deja rastro en una región de color rojo cerca de él para clasificar otro posible punto.

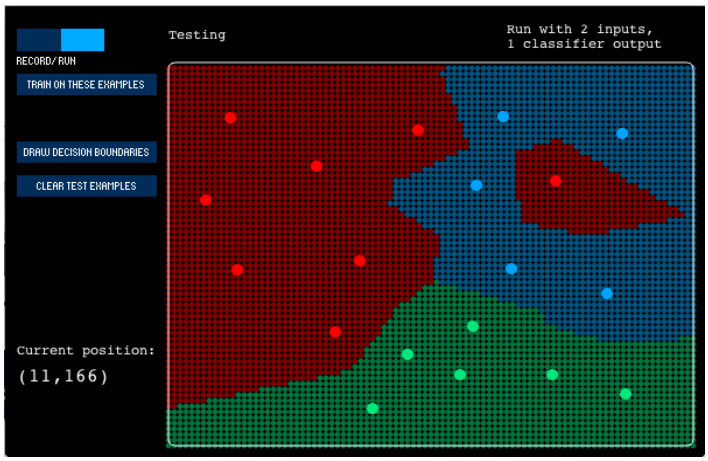


Figura 29. Clasificador con *k* igual a 1.
Tomada de Fiebrink, 2019.

Ahora se procede a cambiar *k* igual a tres y se visualiza en la siguiente figura que el clasificador ya no deja un rastro si se entrena colocando punto de diferente clase dentro de una región de otra clase, pero con ventaja de que considera el hecho de que las clases están separadas en secciones con una diferente interpretación o un sentido intuitivo dentro del clasificador. Estas dos diferencias pueden interpretarse de la manera en que un clasificador va a ser más influenciado que otro en los datos de entrenamiento, lo que sería desfavorable porque causaría mayor probabilidad de generar errores al introducir datos que puedan generar ruido o simples ejemplos atípicos. En conclusión, al aumentar *k* reducirá el sobreajuste y hacer los límites de decisión menos complejos, tomará más tiempo para el procesamiento, pero será insignificante.

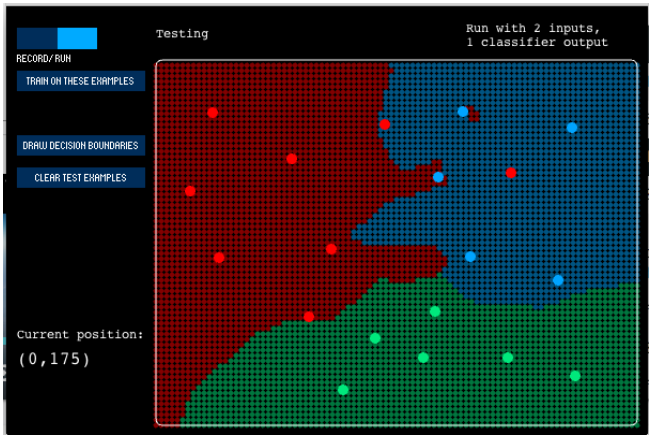


Figura 30. Classifier Explorer con *k* igual a 3.
Tomada de Fiebrin, 2019.

2.7.1.3 Clasificador bayesiano ingenuo (*Naive Bayes)*

Por otro lado, tenemos el algoritmo *Naive Bayes,* el cual es el que brinda un enfoque diferente en formular una decisión de clasificación en términos de probabilidades. En otras palabras, el modelo tren mira una entrada diferente, analiza las características de estas entradas y hace un conjunto de preguntas.

- Primero, dado en cuenta que tenía estos valores de ciertas características, ¿cuál es la probabilidad de que este punto de datos es de clase uno?

$$P(C_1|x_1, x_2, \dots) = ?$$

- Después cual es la probabilidad de que sea clase dos.

$$P(C_2|x_1, x_2, \dots) = ?$$

- Etc., para todas las clases.

Una vez que el clasificador tiene calculadas estas probabilidades elige emitir una etiqueta para la clase con el de mayor probabilidad, siendo esta la característica principal del algoritmo, la etiqueta. Para hacerlo más complicado se averigua cómo responder cada una de estas preguntas, para esto existe una estrategia básica para realizar esto llamada *Naive Bayes.* El primer paso para calcular la probabilidad de los valores de características indicadas es observar

cada clase, es decir, una por una. A continuación, indicamos con un ejemplo una ilustración:

Tenemos un caso de una persona que vive en un barrio en Londres con una cierta cantidad de gatos amigables de barrio, y también una cierta cantidad de zorros salvajes, que son muy llamativos, pero causan muchos problemas como producir mucho ruido y comer todo en los jardines de los vecinos. Ahora digamos que se desea construir un sistema de webcam que vigilará el jardín del usuario, el cual, si ve un gato el cual activará un brazo mecánico que le brindará un bocadillo de atún, pero si es un zorro el sistema emitirá una alarma y hace que se prendan muchas luces parpadeantes para asustarlo.

Ahora digamos que el usuario tiene la capacidad de medir la longitud de un animal cuando camina frente a su cámara, confiando en el hecho de que los zorros son más largos que los gatos de su vecindario, también tomando en cuenta que también existen lobos pequeños y que algunos gatos son grandes, convirtiendo el caso en un sistema imperfecto de clasificación, para finalmente intentar hacer el mejor esfuerzo con la clasificación. Al recopilar los resultados que se han obtenido en la base de datos durante una semana que recopiló la webcam en el patio trasero del usuario tenemos la siguiente figura.

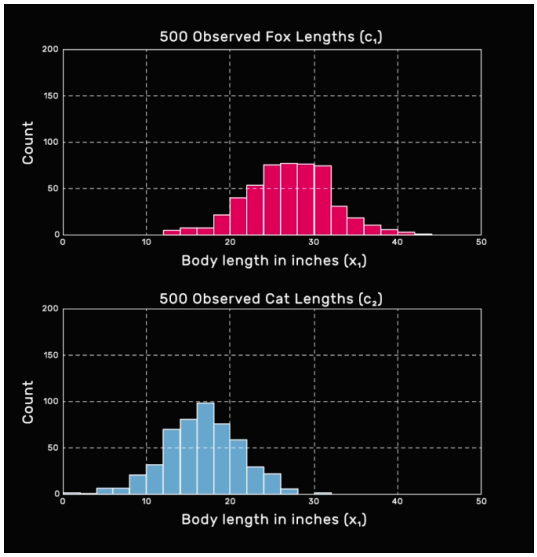


*Figura 31***.** Recopilación de datos de ejemplo *Naive Bayes.*
Tomada de Fiebrink, 2019.

Podemos observar algunas cosas, en primer lugar, la cantidad de muestras son de 500 gatos y 500 zorros, en segundo lugar, es que el promedio de longitud de los gatos es que tienden a ser más pequeños y que los zorros tienden a ser más largos, existiendo un poco de superposición en la mitad. Ahora si miramos del lado de los gatos podemos darnos cuenta que el largo promedio tiende a estar cerca de la misma altura y que la cantidad que tiende a ser dramáticamente pequeña es mínima, como se observa en la figura 32 y 33.

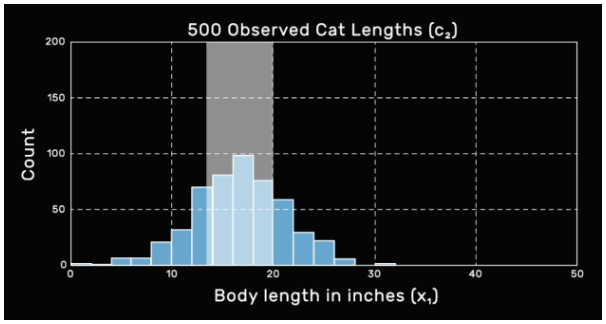


Figura 32. Largo promedio, mayoría de gatos.
Tomada de Fiebrink, 2019.

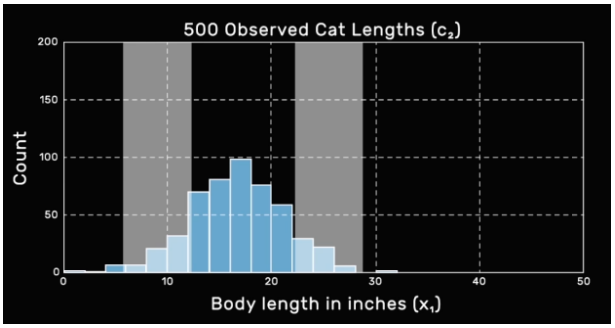


Figura 33. Largo mínimo, cantidad despreciable de gatos.
Tomada de Fiebrink, 2019.

Ahora para hablar en términos probabilísticos las longitudes de los gatos tienen una distribución normal, también denominada como una curva de campana, la cual se puede visualizar en la siguiente figura. Esta curva de campana indica que el valor más alto de la curva en algún valor de longitud se parecerá más a un gato si se acerca más a ese valor. Igualmente, si visualizamos en el lado de los zorros observaremos la misma cosa, una parte indica que el largo de los zorros es igual en promedio y que la muestra de zorros con largo pequeños es significativamente más pequeña.

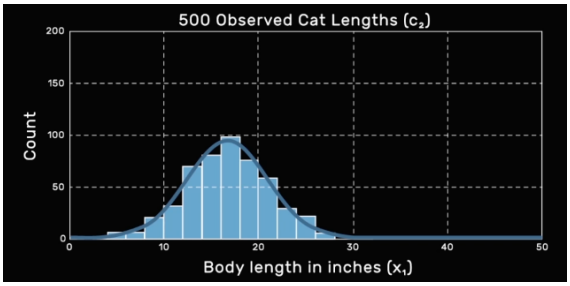


Figura 34. Curva de campana.
Tomada de Fiebrink, 2019.

Si visualizamos estos dos ejemplos y calculamos la mitad de la muestra como promedio de la probabilidad encontramos que al predecir al azar un animal en el vecindario se podría decir que al ser un zorro tendría una longitud más grande de lo que sería un gato. Esta sería una manera simple de entrenar los datos de entrada en función a las muestras dadas, es así que hemos encontrado gráficamente la probabilidad de cual animal sería más largo al diferenciar sus clases.

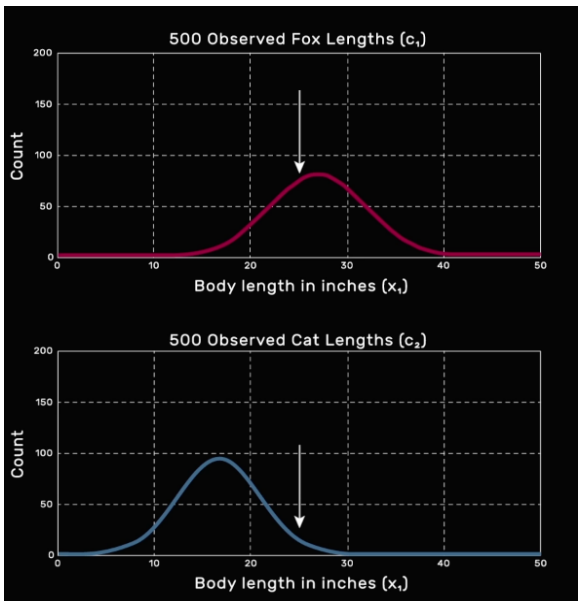


Figura 35. Media de Neive Bayes.
Tomada de Fiebrink, 2019.

Esto es lo que sería el primer paso de la estrategia de *Neive Bayes,* calculando la probabilidad de lo dado con algunos de los valores característicos que se han observado para cada clase.

$$P = x_1 = (x \mid class\ =\ c_n) = ?$$ Ecuación 02

Estos métodos pueden calcularse con ecuaciones bastante fáciles de calcular similares a esto:

$$= \frac{1}{\sqrt{2\pi\sigma_n^2}} e^{-\frac{(x-\mu_n)^2}{2\sigma\ \ \frac{2}{n}}}$$ Ecuación 03

Donde $\mu_n$significa la longitud de los ejemplos de las clases c_ny $\sigma_n$es la desviación estándar de la longitud de los ejemplos de la clase c_n.

Realizando este cálculo plasmará un solo número para cada clase, cuanto mayor sea el número más probable es que un animal de longitud *x* sea miembro de esa clase. Ahora si introducimos otro dato en el cual indicamos que existe solo un 10 por ciento de zorros en el vecindario y veo un animal del rango indicado en la anterior imagen, si la cámara ve un animal de dimensiones grandes, sería más probable que sea un gato grande que de un zorro pequeño, solo porque los zorros son relativamente raros. Este nuevo dato varía los resultados, es así que se procedería a combinar estas dos medidas probabilísticas con una regla de Bayes que dice que si queremos saber la probabilidad de que ocurra un evento A dado que sabemos algo del evento relacionado con un evento B esté sucediendo, se puede calcular esto en términos de algunas probabilidades simples lo que podría ser fácil para calcular específicamente, la probabilidad de un suceso en absoluto.

$$P(A|B) = \frac{P(A)\ P(B|A)}{P(B)}$$ Ecuación 04

La probabilidad del evento A sobre un evento B dado que A está sucediendo y la probabilidad de B, sobre todo.

$$P(class = cat \mid x_1 = x) = \frac{P(class=cat)\ P(x_1=x \mid class = cat)}{P(x_1=x)}$$ Ecuación 05

Aquí estamos usando la regla de Bayes para calcular la probabilidad de que estemos viendo un gato dado que hemos observado un animal de una longitud particular. Se computa usando la probabilidad de que cualquier animal que vemos es un gato, la probabilidad de observar esta longitud si el animal es de hecho un gato y la probabilidad de observar esta longitud para cualquier animal.

$$P(class = fox \mid x_1 = x) = \frac{P(class = fox)\, P(x_1 = x \mid class = fox)}{P(x_1 = x)}$$ Ecuación 06

Así mismo con la probabilidad de otra clase, se nota que en las dos fórmulas se dividen por el mismo número, la probabilidad que observamos es una longitud particular para cualquier animal. En la práctica, realmente no se debe preocuparse por este número porque estamos dividiendo ambas ecuaciones por el mismo número no va a impactar en nuestros resultados, así que en la práctica podemos simplemente ignorarlo. Así que ignorando ese denominador para computar la probabilidad de nuestro nuevo animal desconocido siendo un gato vamos a multiplicar la probabilidad de esta medida de longitud viniendo de un gato por la probabilidad que cualquier animal dado que camina en el patio sea un gato.

$$P(class = cat \mid x_1 = x) = P(class = cat)\, P(\, x_1 = x \mid class = cat)$$

$$= \frac{1}{\sqrt{2\pi\sigma_n^2}} e^{-\frac{(25-\mu_n)^2}{2\sigma_n^2}} = .013$$

Haremos lo mismo por la parte del zorro.

$$P(class = fox \mid x_1 = x) = P(class = fox)\, P(\, x_1 = x \mid class =$$

$$fox) = \frac{1}{\sqrt{2\pi\sigma_n^2}} e^{-\frac{(25-\mu_n)^2}{2\sigma_n^2}} = .007$$

Al calcular los dos casos encontramos que el nuevo animal tendría una mayor probabilidad de que sea un gato.

Cabe mencionar que *Naive Bayes* utiliza un mecanismo sencillo para introducir no solamente un dato de información para tener mayor probabilidad de clasificación, sino varios de ellos como por ejemplo la altura del animal o el color de piel. *Naive Bayes* utiliza probabilidades para medir estas cualidades y simplemente mejorará su predicción si multiplicamos todas. Finalizando esta explicación, este clasificador ayudará a tomar decisiones acerca de qué hacer con las probabilidades, es decir, si encontramos un 51% de probabilidad de que sea un zorro existiría dudas si quisiéramos atraparlo, caso contrario si tendríamos una probabilidad de un 99% tendríamos que ir por él.

2.7.1.4 Árboles de decisión

Los dos casos anteriores se han trabajado bastante con decisiones tomadas ya sean por probabilidades o por distancias, ahora indicaremos una técnica de probabilidad más complicada la cual es *Decision Trees* el cual puede realizar decisiones de tres maneras equivalentes*:*

- Primero con una declaración if-then-else el cual elige una clase si la condición es verdadera, y por lo demás elige la otra clase.
- Segundo, como un pequeño diagrama de flujo con un nodo de decisión.
- En tercer lugar, como un umbral, como una clase en uno y otra clase en otro.

El sistema de decisión sólo utiliza una característica para el umbral, lo que significa un tipo de clasificación con un problema de dos características que permitirá dibujar una línea horizontal o una línea vertical en alguna parte del espacio de características. Este es el ejemplo básico de una raíz de un árbol de decisiones, es así que para construir un buen clasificador con esta técnica de

if-then-else se deberían realizar los mismos pasos algunas veces, aquí se muestra un ejemplo:

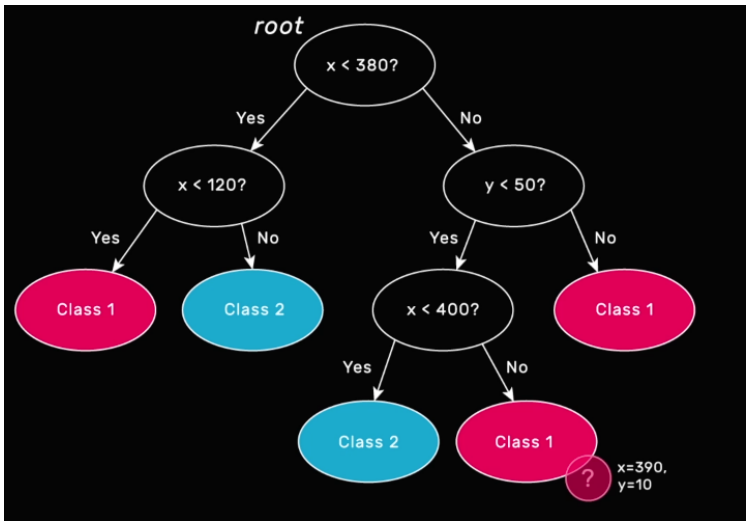


*Figura 36***.** Árbol de decisiones.
Tomada de Fiebrink, 2019.

Como podemos ver la clasificación realizada por el árbol de decisiones ha sido satisfactoria al cumplir varios parámetros o condiciones, para ser más claro tomaremos un ejemplo en donde una muestra que es entrenada con varias condiciones y finalmente clasificada obteniendo la gráfica en la parte derecha.

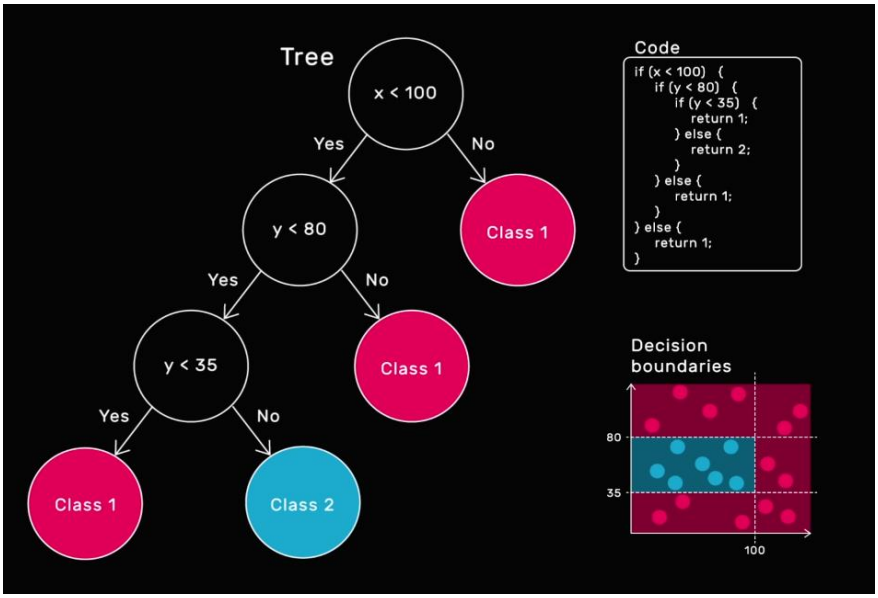


Figura 37. Entrenamiento de un árbol de decisión.
Tomada de Fiebrink, 2019.

2.7.1.5 AdaBoost

Los clasificadores que se han descrito anteriormente han sido catalogados muy poderosos para aplicaciones de clasificación con propósitos en general, ahora hablaremos de AdaBoost. Esta técnica tiene como función acercarse a la problemática de aprender de los datos de entrada usando otra estrategia, similar a la estrategia que se utiliza cuando se desea encontrar un nuevo libro para leer, pero existe un desconocimiento de qué leer a continuación, es decir, cuando se desea encontrar un buen libro se empieza a realizar una serie de preguntas (recolección de datos) hacia algunas personas o amigos para buscar "*recomendaciones*", lo que tendríamos algunos puntos por tomar en cuenta, como por ejemplo el hecho de que existirán personas que no me conozcan tan bien como para recomendar un libro de mi tipo de gustos, mi sentido de humor, colores preferidos, géneros, etc. Es así que el modelo que indicamos reemplazaría un conjunto de amigos por un estudio de decisión.

Es así que tenemos el más simple y menos flexible algoritmo que se ha discutido hasta ahora, siendo realmente una estrategia para crear un conjunto de muchos nodos o muñones de decisión o algoritmos de aprendizaje igualmente básicos y luego hábilmente combinando las clasificaciones para finalmente obtener una muy buena decisión. Llamado algunas veces como un algoritmo de meta-aprendizaje porque hace el trabajo agregando los resultados de los algoritmos de aprendizaje más débiles. Ahora veremos cómo realizar la técnica de aprendizaje AdaBoost paso por paso.

Asumiendo que Adaboost va a usar algunos muñones de decisión, siendo esa una elección bastante común y es el valor predeterminado utilizado por Wekinator. Adaboost comienza entrenando un estudio de decisión en los datos, este estudio encuentra la línea que mejor separa los ejemplos de entrenamiento en una clase de ejemplos de otra clase, entonces AdaBoost calcula la precisión de entrenamiento en este estudio de decisión, es decir, si

tuviéramos que tomar cada uno de los ejemplos de entrenamiento y pedirle al estudio de decisión que los clasifique esperando que lo realice de buena manera. Es así que si se tiene un buen comportamiento y el estudio es capaz de clasificar todos los ejemplos de entrenamiento al 100% de precisión el proceso terminaría, por supuesto que eso casi nunca pasa por la gran cantidad de datos que ingresan en un entrenamiento por más simple que sea.

Caso contrario, si el estudio de decisión obtiene algunos ejemplos correctos y obtiene algunos ejemplos equivocados, AdaBoost realiza un seguimiento de cada uno de estos y calcula lo que es llamado un valor de peso para cada ejemplo de entrenamiento. Se puede interpretar el peso como un valor de comunicación indicando si el ejemplo es más o menos importante para clasificar correctamente. Si el estudio de decisión tiene clasificado incorrectamente un ejemplo de entrenamiento simplemente aumentamos su peso, de lo contrario se disminuye su peso, en la siguiente figura se verá que se está utilizando el tamaño de punto de ejemplo para representar su peso.

Luego de esto pasamos el conjunto de entrenamiento ponderado a otro estudio de decisión, lo que se ha hecho es comunicar que existe un conjunto de datos fuera de la clasificación y buscar hacer algo con ellos, esta vez intentando de minimizar el error de entrenamiento ponderado a diferencia del error de entrenamiento no ponderado. El error de entrenamiento no ponderado es sólo un conteo de cómo se puede cometer errores de entrenamiento mientras que puedes ver que el error ponderado es una suma ponderada.

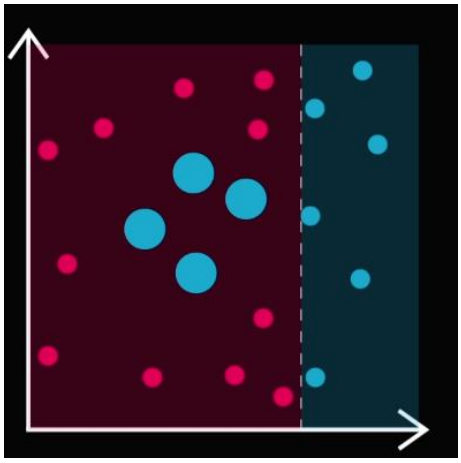


Figura 38. Ponderación de errores.
Tomada de Fiebrink, 2019.

Se puede visualizar en la siguiente figura que se tienen datos de error ponderados los cuales son calculados para identificarlos y etiquetarlos como cualquier otro clasificador.

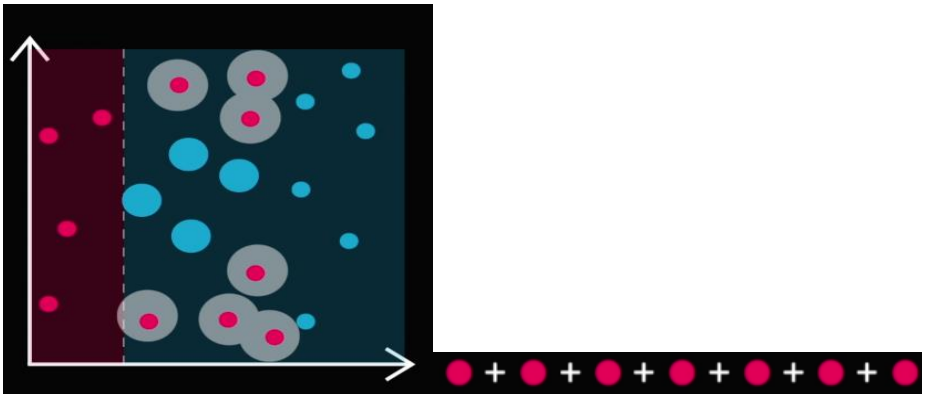


Figura 39. Suma de errores de entrenamiento AdaBoost.
Tomada de Fiebrink, 2019.

Existen 7 ejemplos de error con un peso de 0.62 cada uno, al multiplicarlos obtenemos un error total de 4.32 de 21 en la muestra total. Con esta información lo que se realizaría es disminuir los datos que ya están clasificados y aumentar los que están etiquetados como error, tal y como se realizó con los datos que faltaban clasificar se aumentó su peso, obteniendo algo como esto:

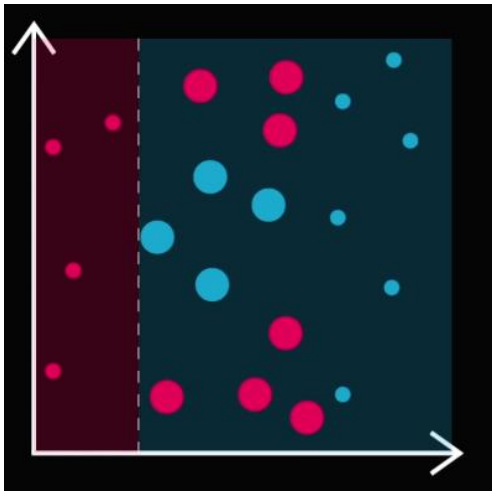


Figura 40. Ajuste inicial de peso de datos de entrenamiento. Tomada de Fiebrink, 2019.

A esto se lo denomina ajustar de nuevo de errores, realizando una repetición de este proceso de entrenamiento dentro del estudio de decisión y reajustando pesos, por un cierto número de iteraciones, obteniendo finalmente un estudio de decisión por iteración, cada uno de estos tocones tiene un aprendizaje de umbral diferente en el espacio de características, logrando averiguar cuánto realmente confiamos en cada uno de ellos. Para cada estudio de decisión se lleva al conjunto de entrenamiento el cálculo de precisión de cada muñón (*estudio de decisión*) en la clasificación de estos datos de entrenamiento, algunos muñones van a ser realmente exactos, caso contrario otros no lo serán pero esto es un proceso que estaría correctamente aceptado, aferrándose a la puntuación de cada uno y utilizarlos como medidas de confianza finalizando el proceso de entrenamiento.

Figura 41. Ponderación de errores.
Tomada de Fiebrink, 2019.

Cuando AdaBoost obtiene una nueva entrada para clasificar lo que pasa es que pregunta a cada uno de los tópicos de decisión utilizar su límite de decisión para clasificar este ejemplo, después combina todas las decisiones en una clasificación general teniendo en cuenta cuánto confía en cada tocón diferente, pudiendo interpretar este caso como un voto ponderado por parte de AdaBoost donde pesa más el resultado que consigue más influencia en la decisión final, finalizando la clasificación.

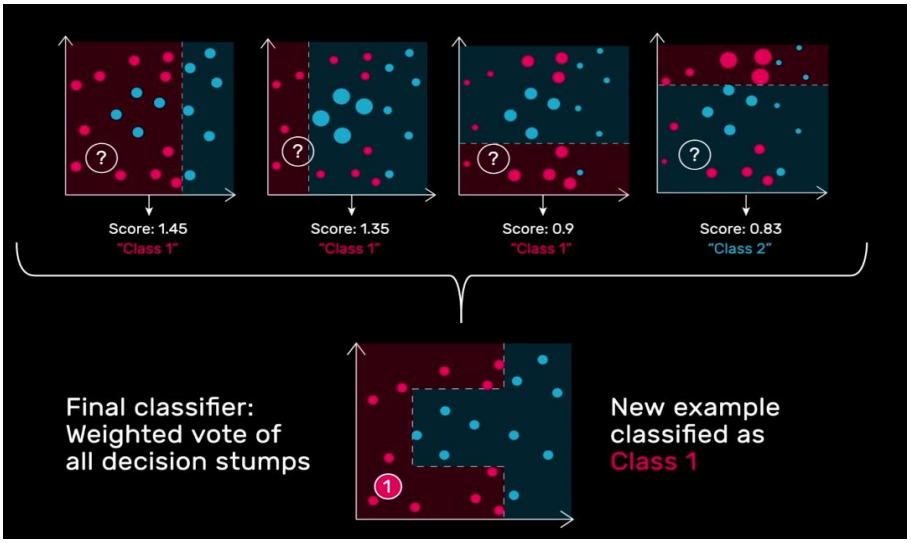


Figura 42. Clasificación final por parte de AdaBoost.
Tomada de Fiebrink, 2019.

2.7.1.6 Máquina de vectores de apoyo

Finalmente hablaremos del último algoritmo de clasificación que tiene Wekinator, el cual también ha sido ampliamente utilizado como AdaBoost. Máquinas de vectores de soporte o SVMs es uno de los algoritmos más difíciles de explicar, SVMs es un algoritmo que también utiliza el recurso de dibujar líneas de decisión o límites de decisión en el espacio de características y también prioriza fuertemente haciendo estos límites lisos y simples. El creador Vladimir Vapnik y su equipo en los laboratorios AT&T desarrollaron este modelo que representa a los puntos de muestra en el espacio, separando las clases a dos espacios lo más amplios posibles mediante un hiper-plano de separación definido como el vector entre los dos puntos, de las dos clases, más cercanos al que se le llama *vector soporte.* Para poder clasificar nuevos datos de entrada, se colocan en correspondencia con dicho modelo, en función de los espacios a los que pertenezcan, los cuales pueden ser clasificadas si bien a una u otra clase. Un ejemplo simple es obtener un conjunto de datos los cuales visiblemente sean muy difíciles en la búsqueda de su clasificación y que se estén visualizando en un plano de dos dimensiones. Para resolver esto lo que se realizan son desplazamientos dimensionales, en este caso aumentar una tercera dimensión con el objetivo de separar las clases y que esto permita una correcta clasificación; cabe recalcar que los hiper-planos son espacios de dimensionalidad muy alta e incluso infinita, estos hiper-planos ayudarán a realizar clasificaciones muy complejas de datos o vectores ingresados (Martínez, 2013).

2.8 Bases de datos

Dentro de los objetivos que se han desarrollado en la investigación es el diseño y producción de una base de datos que permita realizar un sistema de verificación de los clasificadores de *Wekinator* el programa que maneja el flujo de trabajo del aprendizaje automático utilizando los distintos algoritmos de clasificación, para esto es importante entender todo el conjunto de instrumentos

planteados para trabajar, siendo esta una clasificación basada en analogías de circuitos acústicos.

2.8.1 Instrumentos Andinos

En esta sección se explicarán las dos familias que se clasifican en el programa por evaluar, estas familias tienen ciertas características físicas las cuales fueron influenciadas por las funciones que les fueron otorgadas. Algunas de ellas se mantienen en la actualidad como herramientas de identidad para muchas culturas latinoamericanas, es así que se realiza esta descripción de los instrumentos que tienen mayor repercusión desde la época prehispánica hasta la actualidad en material musical andino registrado en algún registro fonográfico (Ibarra, 2016).

2.8.1.1 Flautas rectas (Clase 1)

Las flautas rectas son instrumentos alargados, muchos de estos han sido comunes alrededor del mundo por las utilidades semejantes que se han dado en las diferentes culturas del mundo, lo que se podría variar es la forma de tocar, las aplicaciones y los materiales. Aquí tenemos un grupo de ejemplos que aportarán características que indiquen sus pequeñas diferencias y matices que las hacen diferentes, con el objetivo de poder entender y plantear posibles clasificaciones posteriores; así mismo esta pequeña investigación brindará información para poder entender de manera más clara las ventajas y desventajas del programa de clasificación de sonidos que se ha elegido para el tema.

En la siguiente figura se indican algunas de las características acústicas que tienen las flautas rectas, esencialmente son instrumentos que controlan su parte acústica principalmente con la capacidad de almacenar el caudal de la presión efectuada por el ejecutante. Haciendo una analogía mecánica se trataría de una masa acústica la cual almacena caudal, lo que sería en una

analogía eléctrica de impedancia sería una bobina que almacena el flujo de los electrones.

La flecha representa el caudal o energía de la onda que resuena dentro del tubo, la línea roja es la onda avanzando hasta la desembocadura donde empieza la excitación. Es así que a la altura del primer agujero abierto la onda pierde parte de su energía hacia el exterior, quedando lo suficiente como para seguir avanzando.

Estos tubos son la resistencia del aire, así mismo el área del agujero toma un papel importante, así como su altura o espesor de la pared y el área de la columna de la flauta.

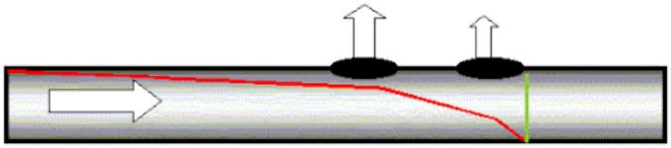
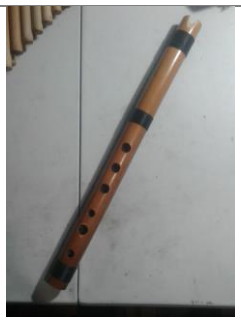


Figura 43. Flujo de caudal de la flauta larga.

Tabla 3

Flautas rectas utilizadas para la fase de entrenamiento y prueba

Nombre	**Quena**	**Pífano**	**Quenacho**	**Pingullo**
Figura				

2.8.1.2 Flauta de pan (Clase 2)

Igualmente, como las flautas rectas, se presentan a continuación la otra familia de flautas, denominadas flautas de pan. Estas flautas tienen características similares; es decir todas tienen el mismo principio físico, pero varían en su coloración, tonalidad, brillo, timbre y resonancia. Este conjunto de instrumentos puede verse diferentes, pero constituyen una misma analogía acústica de sistemas de resonadores de Helmholtz. Estos son ejecutados al ser excitados por presión del aire provocada por un individuo “músico” el cual provoca la resonancia de cada resonador que tiene la capacidad de resonar a distintas frecuencias, en este caso notas fundamentales que pueden complementarse al secuenciar un compás y realizar una armonía.

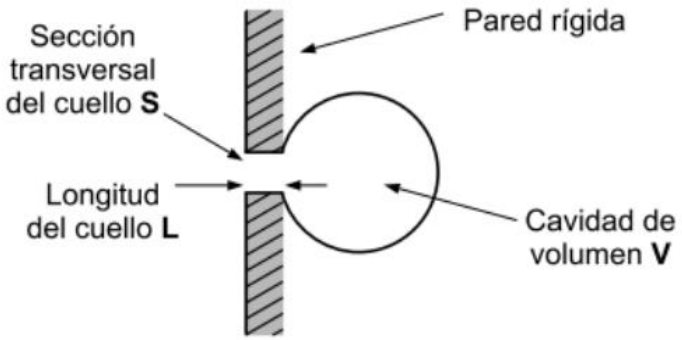


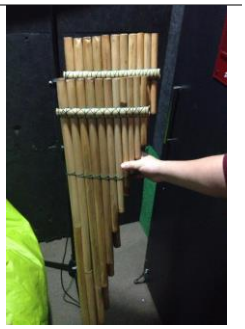
Figura 43. Esquema simple de un resonador de Helmholtz.

A continuación, se muestran ejemplos de flautas de pan:

Tabla 4

Flautas de pan utilizados para la fase de entrenamiento y prueba

Nombre	Paya	Zampoña Cromática	Flauta de Pan Andina	Toyo
Material	Caña	Guadua	Carrizo	Madera

Imagen			
			

3. CAPITULO III: METODOLOGÍA

En este apartado se explicará el desarrollo e implementación que el programa ejecuta mediante un proceso muy sencillo que debe realizarse en un orden en específico; es así que es importante detallar muchas de las características para lograr una ejecución óptima. Los módulos que se explicarán en la metodología corresponden a todas las partes que formaron parte para realizar un clasificador de audio; cabe recalcar que todas las herramientas utilizadas tienen diferentes funciones, es decir, por ejemplo una parte de ellos fueron desarrollados para realizar aplicaciones creativas de audio, siendo el caso Reason y Ableton Live, además las herramientas que fueron creadas como herramientas de comunicación con *Sound Classifier* de *Processing* que es un entorno virtual desarrollado en Java y que utiliza distintas librerías para comunicarse o extraer información de los sonidos.

Se describe en la siguiente figura el flujo de trabajo que se tomó en cuenta para poder desarrollar la clasificación juntando las herramientas creativas con las herramientas de programación y la aplicación de algoritmos ML. El trabajo desarrollado se lo indica en dos fases, la fase de base de datos y la fase de prueba del programa, seguido de la validación cruzada y el análisis de los resultados.

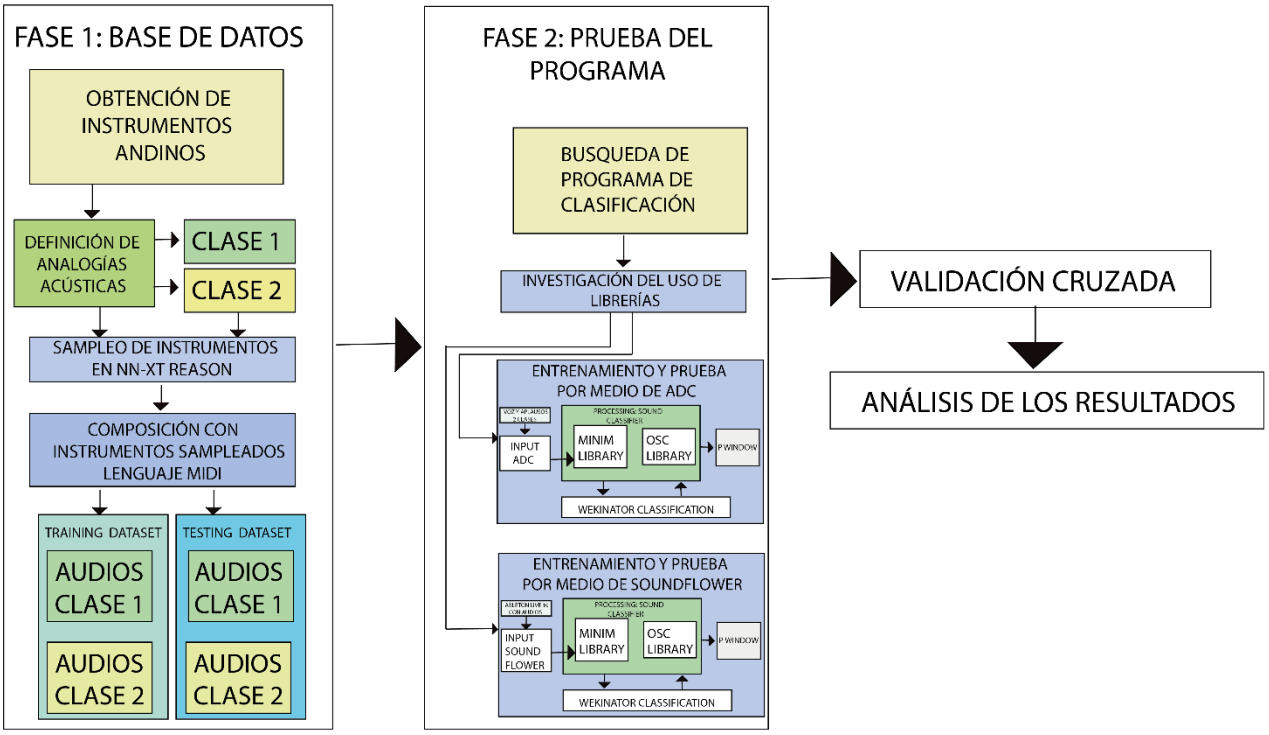


Figura 44. Flujo de trabajo de la clasificación.

3.1 Herramienta de Clasificación

Este programa llamado "*Sound Classifier*" es una herramienta diseñada por Tore Knudsen, un diseñador de interacción con sede en Copenhague y graduado de K3 en Malmö Högskola. Es una herramienta para analizar sonidos de entrada que son procesados dentro de un espectrograma.

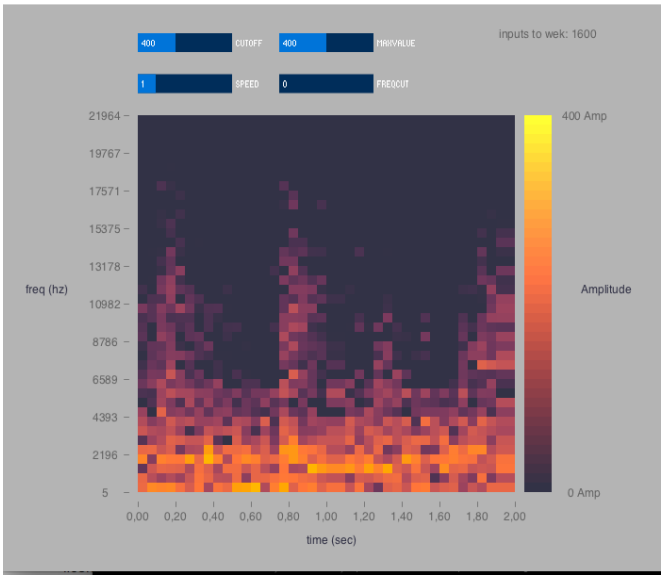


Figura 45. Interfaz gráfica del programa.

La figura 46 es un cuadro que cambia de color en función a la clase que el clasificador está reconociendo, esta es una de las herramientas más interactivas e importantes de la investigación, debido a que en la validación cruzada que se evalúa al momento de la fase de prueba se utiliza este cuadro para realizar las observaciones correspondientes y evaluar la precisión del programa con sus respectivos clasificadores.

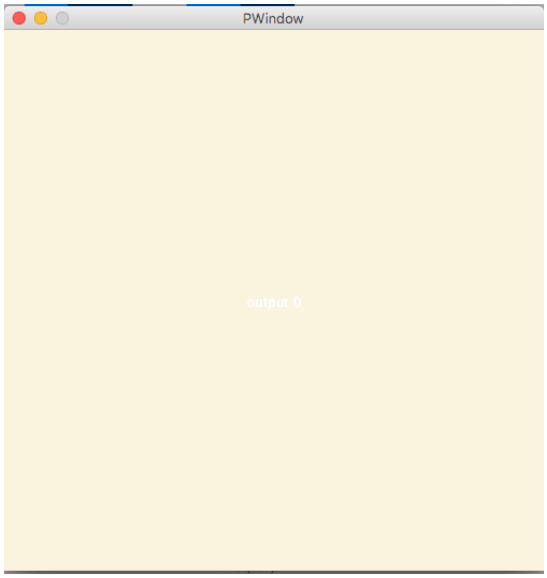


Figura 46. Señalador de clases del clasificador.

En esta sección evaluaremos el comportamiento del programa “Sound Classifier”, indicaremos las librerías utilizadas para desarrollar el espectrograma y que herramientas se utilizaron para modificar los parámetros del mismo. Por otra parte se evaluará la comunicación OSC la cual permitirá realizar la comunicación entre “SC” y “Wekinator”; para finalmente indicar cómo se realiza el procedimiento para medir la efectividad del programa en función a resultados de muestras utilizando un protocolo de audio interno llamado “SoundFlower” el cual permitirá clasificar audios en un bus de salida configurado desde Ableton Live y Reason, en nuestro caso composiciones de los instrumentos andinos para entrenar nuestro clasificador.

El programa es muy fácil de ejecutar en cortos pasos, la plataforma *Processing* permite crear entornos virtuales dentro del mismo y no dentro del sistema operativo, el mismo que permite importar librerías las cuales serán explicadas en el mismo apartado. Las ventajas de tener esta herramienta a la mano es el tipo de lenguaje en el cual está basado el programa, Java es un lenguaje de programación que ha permitido a los programadores evitar cambios de sintaxis ya que es un lenguaje estándar en el desarrollo de producción multimedia, así mismo *Processing* a través de Java ha implementado una sintaxis simple a la hora de realizar cálculos y la creación libre de librerías que pueden ser descargadas en la Web y que son evaluadas bajo una licencia GNU GLP (*General Public License),* evitando problemas que han sucedido con el lenguaje de programación de *Python* el cual tiene muchas versiones las cuales al abrir un programa se debe especificar en qué entorno virtual desarrollar y verificar si existe compatibilidades con las librerías, como por ejemplo la librería *librosa* que tiene muchos cambios y muchas versiones, muchas incompatibles con varias versiones de *Python.*

El programa se encuentra ubicado en una página de desarrollo de programación llamado “GitHub” el cual explica los pasos a seguir para abrir el programa, también permite encontrar el perfil de los creadores de los programas y poder verificar su CV y muchos datos que son necesarios para poder seguir los pasos de un programador y amante de la tecnología. El contenido del programa se encuentra ubicado en el siguiente enlace web: *https://github.com/torekndsn/SoundClassifier.*

El sistema tiene el siguiente contenido, el cual consta de la estructura que será analizada en relación a qué conexiones y funciones tienen cada uno de ellos, obteniendo una gran ventaja; ya que existirá un orden permisible e intuitivo para quien está interesado en experimentar con él, así mismo en comunicación con OSC de Wekinator.

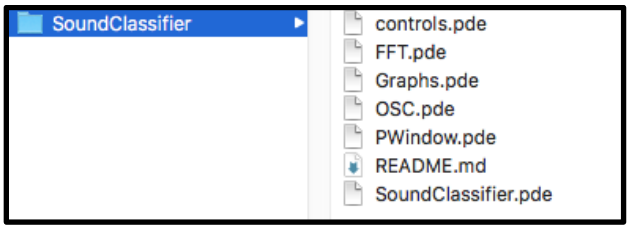


Figura 47. Contenido de la carpeta SoundClassifier.

La carpeta es un paquete de programas enlazados entre sí de formato. pde, siendo seis ventanas las cuales representan una parte y función del programa, el cual es desarrollado en Processing. En el siguiente apartado se mostrará la estructura del programa y cómo se procesa su funcionamiento.

3.2 Processing

Es un visualizador de código flexible y utiliza un lenguaje para aprender a codificar en el contexto de las artes visuales. Desde 2001, Processing ha promovido la alfabetización en software dentro de las artes visuales y la alfabetización visual en tecnología. Fue creado, en principio, por Casey Reas y Ben Fry, y se trata de un software de código abierto que se distribuye bajo una licencia GNU, General Public License (Buioli & Pérez, 2013).

Gracias a los medios de producción multimedia, nace este potente software instaurado para producir imágenes, animaciones e interactivos. Convirtiéndose a partir de terminologías realizadas en el MAT Lab un poderoso entorno de producción basado en Java.

En contexto su interfaz gráfica es muy cómoda la cual tiene un menú principal, una barra de herramientas, un conjunto de ventanillas que conllevan a un solo proyecto, un editor de texto, un área de mensajes y finalmente la consola. La consola es la que permite visualizar que es lo que sucede dentro del programa al correrlo. La teoría de programación en esta plataforma puede encontrarse en manuales en la red, donde se explicaría cómo programar la sintaxis de todos los casos de variables, funciones y ejemplos.

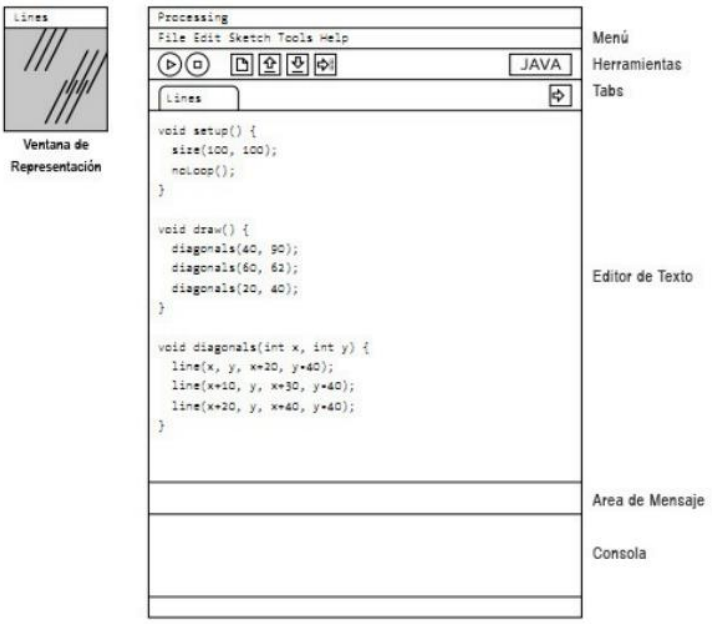


Figura 48. Módulos de la aplicación

3.3 Librerías

Para manejar el programa principal, se deben manejar ciertas librerías que proporcionarán ampliar las posibilidades de extraer datos de salida hacia la entrada de Wekinator, con ayuda de estas librerías y su respectiva explicación se comprenderá fácilmente la estructura que utiliza *Sound Classifier*. Es importante comprender el flujo de trabajo que realizaría el programa, en qué parte se utilizarían las librerías y la sintaxis que deben utilizar.

3.3.1 oscP5

Es una implementación del protocolo OSC para el procesamiento de programas que comunican datos de audio entre sí. “oscP5” es una biblioteca escrita por Andreas Schiegel, actualmente su última actualización se la realizó el 19/12/2011. Su acrónimo es Open Sound Control, el cual es un protocolo de red desarrollado en cnmat, UC Berkeley. Su función es la comunicación entre computadoras, sintetizadores de sonido y otros dispositivos multimedia,

optimizado para la tecnología de red moderna, siendo utilizado en muchas áreas de aplicación.

OSC ha logrado un amplio uso en campos que incluyen nuevas interfaces basadas en computadora para la expresión musical, sistemas de música distribuida en red de área local, comunicación entre procesos y dentro de aplicaciones individuales.

Básicamente se introduce la opción de oscP5 conjuntamente con una variable de entrada, así como una de salida. Las comunicaciones deben comportarse de esa manera para poder interactuar con el circuito del algoritmo de ML para poder entrenar a "*Wekinator".* Es así que se tiene una entrada por el bus 12000 y una salida por el canal interno 6448 que pertenece a la entrada de *"Wekinator".*

Por ejemplo, tenemos una descripción de qué es lo que se enviará a *"Wekinator",* en este caso tenemos cerrado la dirección ID *"127.0.0.1",* la cual indica que es el mensaje, siendo solamente una representación de la sintaxis. Es así que por el canal 12000 se enviará un arreglo del espectrograma por la carpeta "("/wek/inputs")".

3.3.2 Minim

Minim es una biblioteca de audio que utiliza API JavaSound, un poco de Tritonus, y el MP3SPI de Javazoom, con el objetivo de proporcionar una biblioteca de audio sencilla para el desarrollo de procesamiento. El objetivo y filosofía de la API es lograr que la integración de audio sea lo más sencilla posible, al mismo tiempo que proporciona una gran cantidad de flexibilidad para los usuarios que deseen realizar proyectos más avanzados.

Una de las ventajas de esta librería es que no se necesita manipular matrices de muestras como sucede en la librería de *“librosa”* en *Python* que utiliza las constantes MFCC, las cuales son un conjunto de constantes que resumen muestras de audios y tienen mayor relevancia solamente para identificar de mejor manera las características que tiene el habla humana en vez de evaluar sonidos de distinto índole acústico y evitar problemas al querer clasificarlos, debido a la utilización de la escala de Mel en vez de una escala normal logarítmica.

La ventaja de tener flexibilidad a la hora de crear bocetos de manera rápida es que se puede realizar una clasificación en tiempo real, que pueda indicar cómo se comporta el clasificador que queremos evaluar y validar, así mismo que permita tener un proceso estable en el momento de entrenar los datos de entrada de *“Wekinator”.*

A continuación, se presentan algunas de las características de Minim, las cuales se han utilizado algunas en nuestro programa y serán analizadas con algunos ejemplos como fue el caso de la anterior librería Osc. Estas características están ubicadas en la página oficial de minim.

- AudioPlayer: Reproducción mono y estéreo de archivos WAV, AIFF, AU, SND y MP3.
- AudioMetaData: un objeto lleno de metadatos sobre un archivo, como las etiquetas ID3.
- AudioRecorder: Grabación de audio mono y estéreo con búfer o directamente al disco.
- AudioInput: Monitoreo de entrada mono y estéreo.
- AudioOutput: Síntesis de sonido mono y estéreo.
- FFT: Transformada de Fourier en los datos de audio para generar un espectro de frecuencia.

- BeatDetect: Detección de tiempos.
- Un marco de síntesis en tiempo real basado en generadores de unidades, que llamamos UGens.

Se declara las funciones que entrarán en el código, es importante hacerlo ya que se podría confundir si se desearía introducir otro audio independiente, o si se desearía realizar otro proceso de transformada de fourier, así mismo estas funciones pertenecen a la librería Minim.

El monitoreo que activa la entrada del programa de la computadora utilizando la variable *in* depende de la configuración interna que está corriendo el computador. Utilizando el método getLineIn se obtienen el tipo de audio de entrada el cual puede ser mono o estéreo y el tamaño de buffer, el cual se ha seleccionado 2048. También se podría ampliar a obtener la velocidad de sampleo, así mismo la profundidad de bits.

3.3.3 ControlP5

ControlP5 es una librería creada por Andreas Schlegel diseñada para el entorno de programación Processing; esta librería permite definir con facilidad los elementos de los formularios, los cuales pueden ser cuadros de texto, cuadros de contraseñas, casillas de verificación, barras de desplazamientos, botones, etc.

Los controladores de ControlP5 son elementos que se encontrarán posiblemente en algún tipo de formulario, de los cuales se encuentran:

- Button y Toggle: equivalen a botones de selección.
- CheckBox y RadioButton: son casillas de verificación y botones opcionales.
- DropdownList y ListBox: son listas desplegables y listas fijas.

- Slider: barras de desplazamiento.
- TextField: cuadros de texto.

3.3.4 SoundFlower

Es una extensión de kernel de código abierto para MacOS, está diseñada para crear un bus interno que envía una salida de audio virtual, el cual también tiene la característica de actuar como entrada. El desarrollador principal es *"Cycling '74"*, una empresa que tiene como finalidad construir herramientas para el sonido, multimedia y la interactividad entre programas que manejen las dos. En 2014 al no solucionar los problemas que tenía esta interfaz interna pasa a la administración Rogue Amoeba, obteniendo malos resultados para mejorar el producto, pero es el autor original de Soundflower Matt Ingalls para terminar el prototipo y lanzarlo a la página web de GitHub.

Esta interfaz interna permitirá comunicarnos directamente con el programa en Processing, y con la librería Minim, ya que esta librería lee la entrada por defecto que se le seleccione al computador por medio de su configuración inicial. Es así que se podría utilizar esta salida desde una DAW, produciendo una comunicación directa sin necesidad de tener una librería que logre esta comunicación. Un punto positivo para la configuración de *"Sound Classifier".*

3.4 Módulos

Los siguientes módulos facilitarán el entendimiento del sistema que utiliza Wekinator para clasificar diferentes clases de datos.

3.4.1 Módulo de Clasificación

Wekinator consta de un conjunto de módulos, los cuales pueden variar dependiendo de la aplicación que se desea realizar. En este pequeño módulo de clasificación se puede seleccionar una sola variable mediante una barra

espaciadora y por otro lado se selecciona que tipo de clasificador vamos a utilizar para verificar cual tendría mejores resultados.

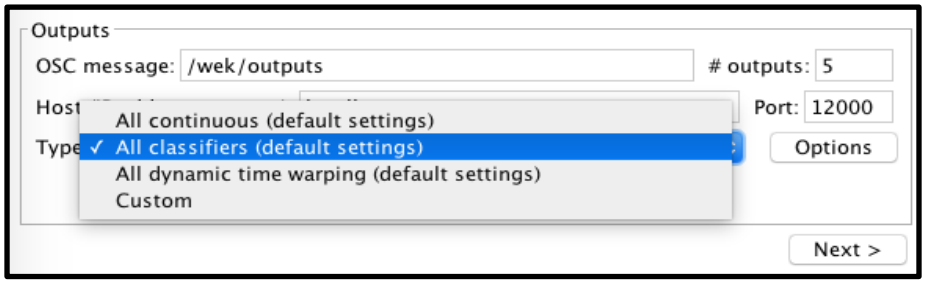


Figura 49. Primer Módulo de clasificación

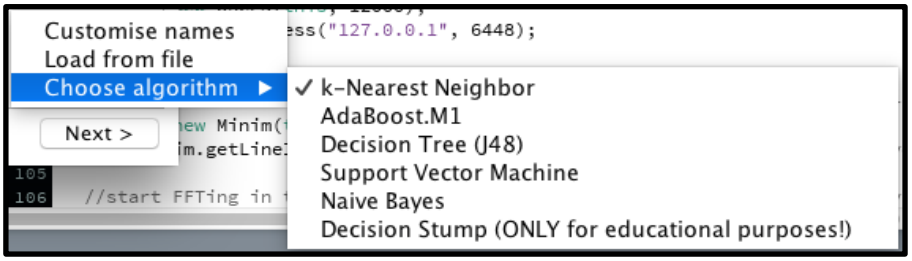


Figura 50. Segundo módulo de selección de clasificador.

Igualmente es lógico tener en cuenta que existirán otros módulos o una configuración de ellos para lograr el conjunto de datos de entrada para la correspondiente clasificación.

3.4.2 Módulo general: Configuración

Anteriormente se ha hablado del flujo que maneja Wekinator, o ejemplos que aclaran un funcionamiento más generalizado. Es importante comprender la estructura de seteo o configuración al iniciar el programa, este tiene un orden en específico pero para poder desarrollado se deben tener claro para qué sirve cada casilla que se va a utilizar; así mismo se toma en cuenta que esta explicación está explícitamente ejecutada para realizar clasificaciones, más no otras aplicaciones, es así que si existe algún módulo que no sea utilizado en el proyecto no vale la pena revisarlo, para ello podrían ingresar a la página web del programa e investigarlo por ustedes mismos.

Wekinator funciona a través de proyectos, los cuales pueden ser guardados y guardar el entrenamiento realizado anteriormente, es una de las ventajas que tiene el programa. Al crear un nuevo proyecto se plasma una ventana que se compone de tres módulos, recibimiento de datos OSC, entradas y salidas.

- Módulo Receiving Osc: Es el protocolo de comunicación de entrada, cabe mencionar que Wekinator es un programa que ha sido pensado para utilizar audio, es así que Osc funciona muy bien con este tipo de comunicación flotante ya que es continua, igualmente se podrían manejar otro tipo de datos como vectores, midi, etc. Se puede visualizar que tienen entradas de control en puertos internos del protocolo, el estándar es el puerto 6448.
- Inputs: En esta sección del módulo general se captan los mensajes en cantidad de entradas, las que el programa de salida permite, sino existirían errores de compatibilidad.
- Outputs: Los mensajes de salida de Wekinator se encuentran en una carpeta contenedora de estos datos; al igual que las entradas se selecciona la cantidad de salidas por medio del puerto estándar 12000. En esta sección también se indica el tipo de clasificadores que el programa utilizará.

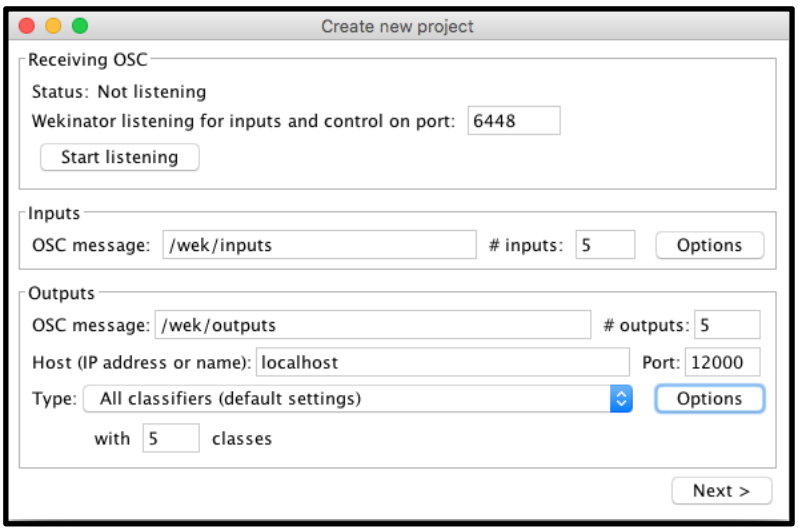


Figura 51. Módulo general de configuración Wekinator.

3.5 Base de datos

La base de datos del sistema consiste en tener varios ejemplos, con algunas condiciones similares, debido a que la interpretación que se pretende desarrollar es encontrar una clasificación con buenos resultados, ya que se ha diseñado un sistema de audios y un sistema de comunicación que evite los ruidos externos que el micrófono haya captado y eléctricos que puedan afectar al entrenamiento y a la prueba. Para esto ha sido necesario utilizar un software que tenga las características óptimas para diseñar una base de datos en base al sampleo y a la creación de instrumentos que puedan generar audios para entrenamientos de Machine Learning con una buena calidad, ligado a la clasificación de instrumentos.

NN-XT es el sampleador favorito de Reason, el cual tiene módulos muy amigaBles los cuales se explicarán superficialmente en el siguiente apartado, para dar énfasis al trabajo desarrollado de sampleo de instrumentos andinos.

3.5.1 Sesión de instrumentos en Reason

Para realizar el entrenamiento del programa se debe contar con una base de datos que contenga dos o más clases de sonidos para obtener resultados y un posterior análisis de los mismos, es así que se presenta el diseño de la base de datos para este modelo, la base de datos consiste en un sampleo de varios instrumentos, donde se utiliza un software que permitirá grabar instrumentos musicales y sus respectivos sonidos característicos. Este sampleador pertenece a la DAW de Reason conocido como el NN-XT, como podemos ver en la siguiente imagen es una herramienta muy útil para crear instrumentos virtuales a través de sampleos que un desarrollador sonoro puede crear, estos instrumentos serían únicos debido a que las condiciones siempre serán distintas.

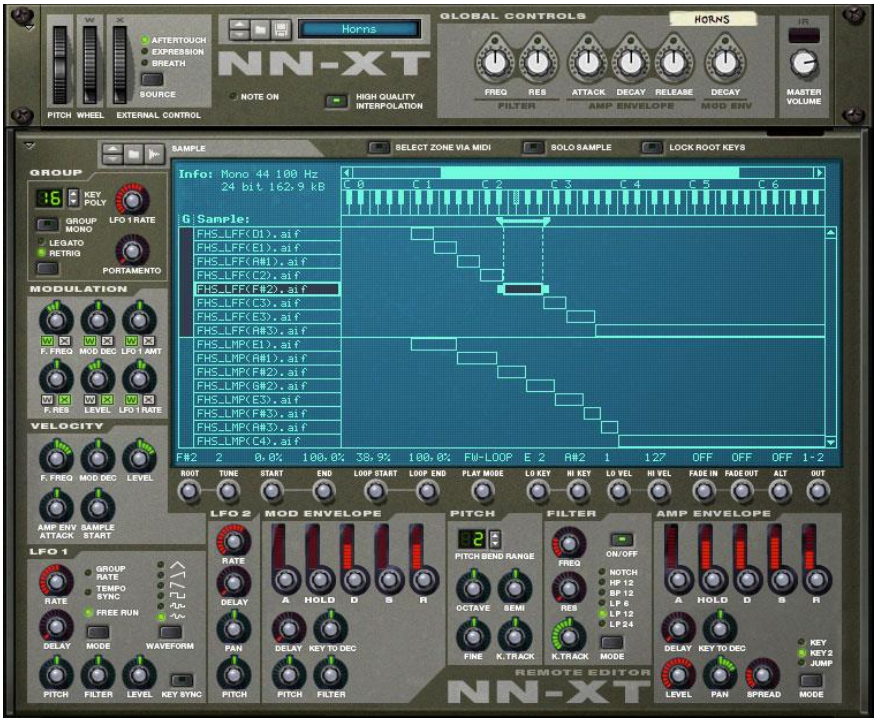


Figura 52. NNXT

Las funciones básicas del NN-XT son muy similares que las otras herramientas del Rack de Reason. Por ejemplo, como en el NN-19, el NN-XT te permite cargar samples y crear configuraciones de multi-samples que se distribuyen alrededor de un teclado de notas. Los sonidos grabados pueden ser modificados por un set comprensivo de síntesis de parámetros.

Para proceder a samplear los sonidos de los instrumentos existe una opción llamada *“Sampling”,* la cual te dirige a una ventana de edición de audio donde puedes proceder a visualizar el audio de entrada y controlar varias funciones como los fundidos de entrada, salida, normalizar la onda, realizar cortes o realizar modos de repetición o “Loop”.

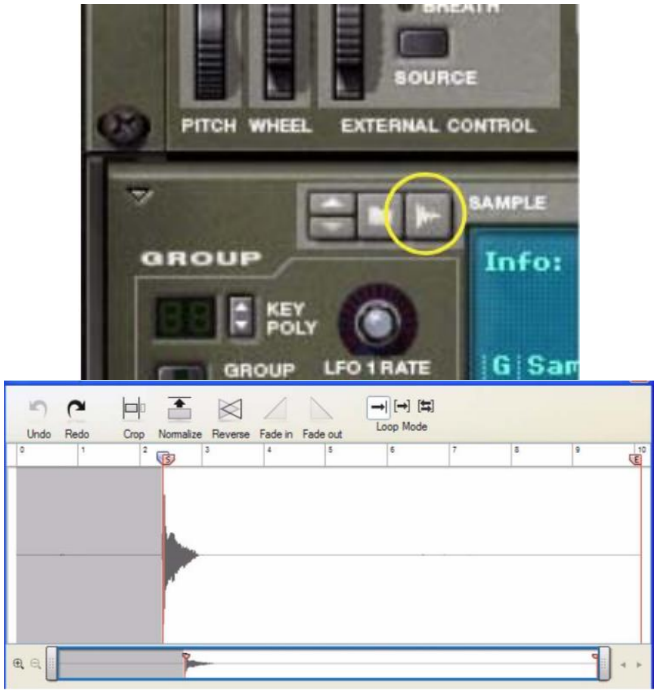


Figura 53. Sampleador.

Para alimentar nuestro clasificador necesitamos una amplia base de datos para encontrar una clasificación que permita validar los resultados en base al comportamiento del programa. Existe una clasificación de instrumentos, las cuales se mostraron en el marco teórico, así mismo como una serie de ejemplos para tener claro cuál sería la clasificación base para la validación de resultados. Es así que a continuación se describen los instrumentos que se samplearon y la configuración de cada instrumento en NN-XT, como los equipos utilizados para su grabación y su cadena electroacústica.

3.5.1.1 Clase 1: Flautas rectas

Se procede a realizar la grabación de algunas de las notas de los instrumentos propuestos para la clasificación, con el objetivo de completar las notas dentro de una escala dependiendo de la capacidad tonal del instrumento. La realización de un sampleo dentro del sampleador NN-XT tiene la ventaja de tener un pitch bend que completaría el resto de la escala que el instrumento en

sí no puede modificar e igualmente completar la ejecución de armonías que se podrían ejecutar en una secuencia midi.

A continuación, se presentan los rangos tonales que tiene cada instrumento sampleado.

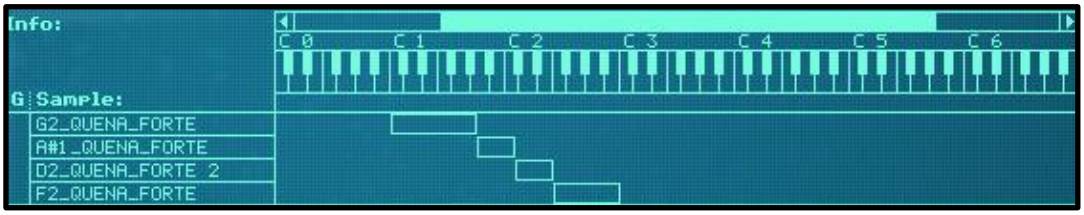


Figura 54. Sampleo de la Quena en NN-XT de Reason

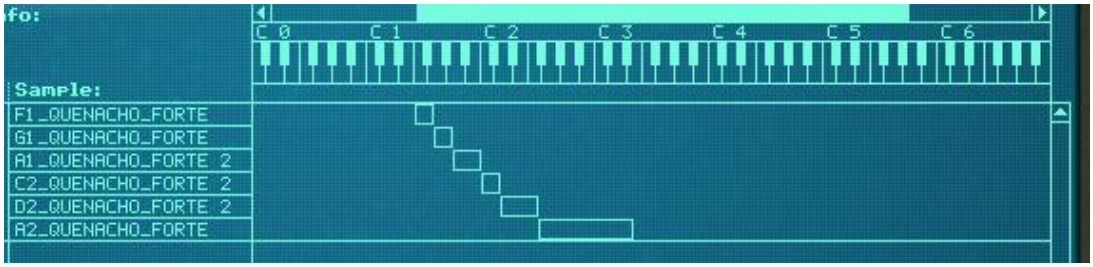


Figura 55. Distribución módulo de teclado quenacho

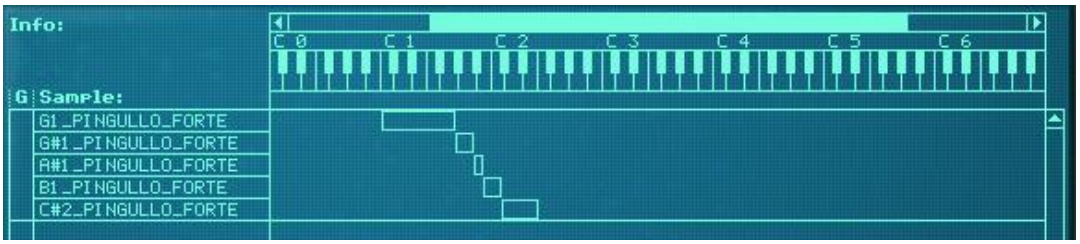


Figura 56. Distribución teclado midi Pingullo

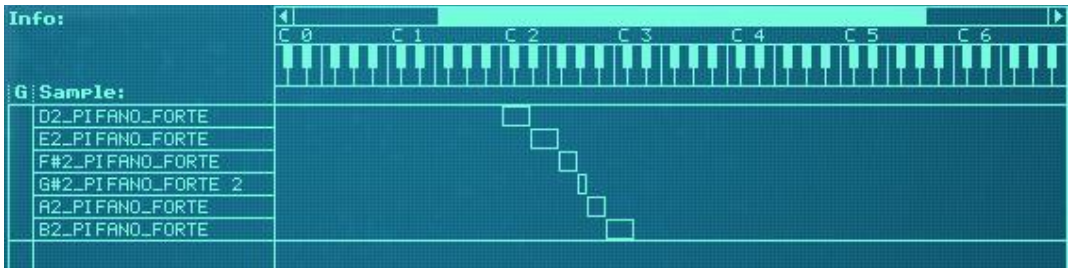


Figura 57. Sampleo del pífano en NN-XT de Reason

3.5.1.2 Clase 2: Flautas de pan

Cómo habríamos visto en el apartado de marco teórico, el principio de una flauta de pan consiste en resonar un volumen que tiene como características principales una masa acústica, resistencia acústica y compliancia acústica, estas tres juntas forman lo que se denominaría como un resonador en analogía acústica en función a una analogía mecánica.

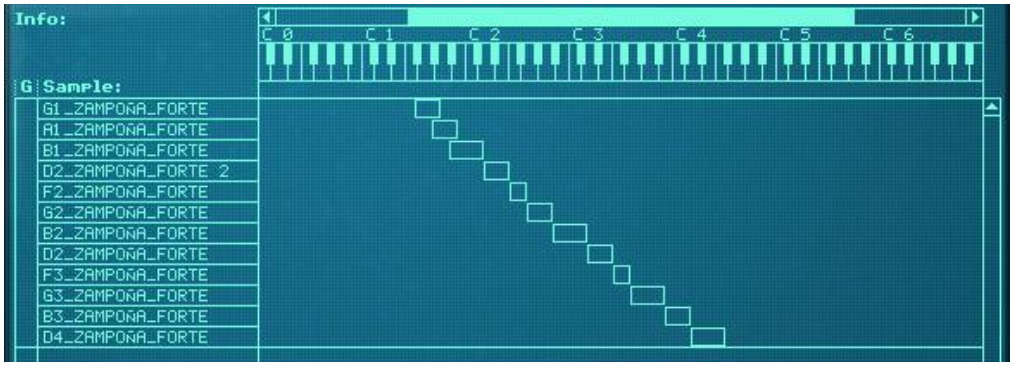


Figura 58. Sampleo de Zampoña cromática en NN-XT de Reason.

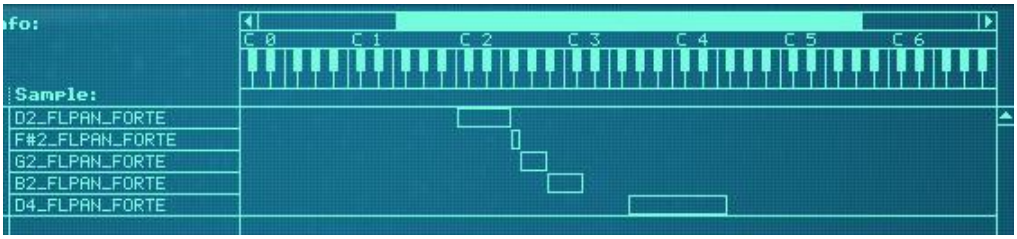


Figura 59. Sampleo de la flauta dulce en el NN-XT de Reason.

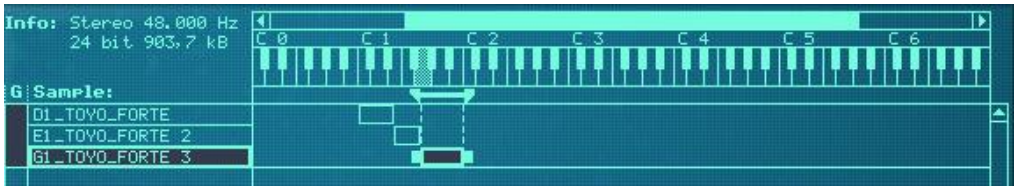


Figura 60. Sampleo del Toyo en NN-XT de Reason.

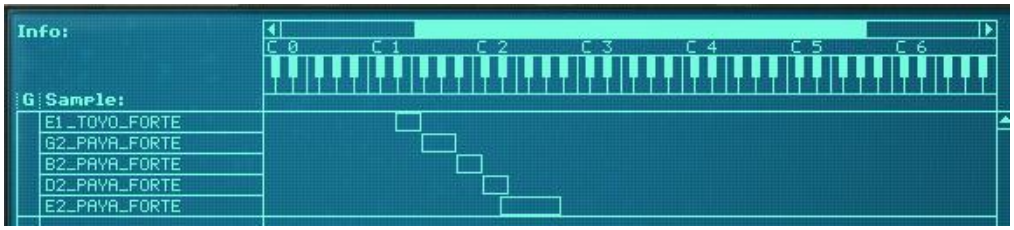


Figura 61. Sampleo de una paya en NN-XT de Reason.

A continuación, se muestran evidencias de las grabaciones que se realizaron respectivamente de los samplers. Las grabaciones fueron realizadas en un espacio libre de reflexiones importantes para obtener sonidos lo más limpios posibles y evitar la interacción con posibles reflexiones, esto se ha realizado con el objetivo de evitar ruido en la alimentación del programa de entrenamiento automático.

Para la cadena electroacústica se utilizaron los siguientes dispositivos:

- Interfaz Apollo-Twin
- Pedestal de micrófono
- Micrófono Shure Beta 57
- Audífonos Sennheiser HD 280 Pro

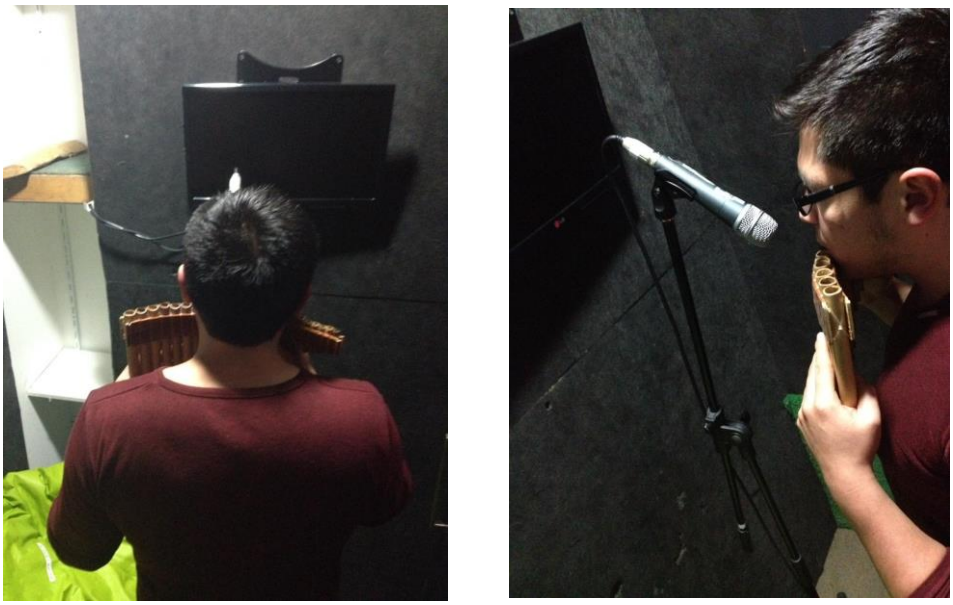


Figura 62. Grabaciones de los instrumentos andinos.

3.6 Training y Test Dataset

Una vez extraída los audios de la base de datos, el *dataset* será una matriz o una tabla que indique a qué clase pertenece cada uno de los ejemplos (*aprendizaje supervisado*), y que contenga las *features* con cada uno de los

ejemplos, que en este caso será únicamente las informaciones de las amplitudes por frecuencia. Un ejemplo de formato de *training dataset* se encuentra en la tabla 5.

Ahora presento una tabla donde están los audios de entrenamiento con sus clases y sus respectivas duraciones:

Tabla 5
Training DataSet

Training DataSet		
Audio	**Clase**	**Duración**
FLAUTADEPAN_01.wav	1	00:23
FLAUTADEPAN_02.wav	1	00:36
FLAUTADEPAN_02.wav	1	00:33
PAYA_01.wav	1	00:22
PAYA_02.wav	1	00:24
PAYA_03.wav	1	00:36
TOYO_01.wav	1	00:16
TOYO_02.wav	1	00:24
TOYO_03.wav	1	00:36
ZAMPOÑA_01.wav	1	00:15
ZAMPOÑA_02.wav	1	00:21
ZAMPOÑA_03.wav	1	00:16
PIFANO_01.wav	2	00:14
PIFANO_02.wav	2	00:12
PIFANO_03.wav	2	00:20
PINGULLO_01.wav	2	00:19
PINGULLO_02.wav	2	00:25
PINGULLO_03.wav	2	00:17

QUENA_01.wav	2	00:15
QUENA_02.wav	2	00:18
QUENA_03.wav	2	00:29
QUENACHO_01.wav	2	00:16
QUENACHO_02.wav	2	00:26
QUENACHO_03.wav	2	00:15

Para evaluar los resultados de la clasificación existen algunas técnicas, en este caso será la de validación cruzada, la cual consistirá en utilizar un conjunto de datos de entrenamiento *training dataset* (12 audios por cada clase), junto con una muestra más pequeña de evaluación (4 audios por cada clase) y la cantidad de veces que se han equivocado los clasificadores en cada una de las pruebas, obteniendo porcentajes de precisión que serán promediados para evaluar el comportamiento de cada uno de los clasificadores.

Los 12 audios del *training dataset* se dividen en 4 instrumentos, 3 audios por cada instrumento, igualmente tenemos un audio por cada instrumento en las muestras de *test* en cada clase, a continuación, indicamos una tabla que muestra las características de cada audio.

Tabla 6

Test DataSet

Test DataSet		
Audio	**Clase**	**Duración**
FLAUTADEPAN_04.wav	1	00:17 minutos
PAYA_04.wav	1	00:38 minutos
TOYO_04.wav	1	00:14 minutos
ZAMPOÑA_04.wav	1	00:15 minutos

PIFANO_04.wav	2	00:21 minutos
PINGULLO_04.wav	2	00:15 minutos
QUENA_04.wav	2	00:15 minutos
QUENACHO_04.wav	2	00:22 minutos

Se procede a entrenar el programa realizando una sesión en Ableton Live 10, la cual tendrá los 12 audios de cada clase. Ableton Live 10 se comunicará con el programa *Sound Classifier* por medio de la interfaz interna *Soundflower* para a través de OSC comunicarse con Wekinator para realizar el entrenamiento. Los audios están distribuidos continuamente uno de otro en Loop para evitar silencios que puedan producirse por transiciones no deseadas, esto se realizará para las dos clases de instrumentos de viento andinos.

Figura 63. Configuración comunicación SoundFlower Ableton Live 10

Al finalizar el entrenamiento por parte de las dos clases se procede a realizar el *test* utilizando los audios de test en canales independientes de Ableton Live 10, con la misma configuración de Loop y comunicación entre AL 10, *Sound Classifier* y Wekinator. Dependiendo la duración de cada audio se realizarán 4 pruebas por cada audio de *test* y promediando la precisión entre todos los audios y evaluando los 5 clasificadores en los resultados para finalmente sacar nuestras conclusiones. Si es pertinente y necesario se evaluarán los parámetros de los clasificadores para mostrar su precisión.

Figura 64. Loop de Ableton Live para la clase 1

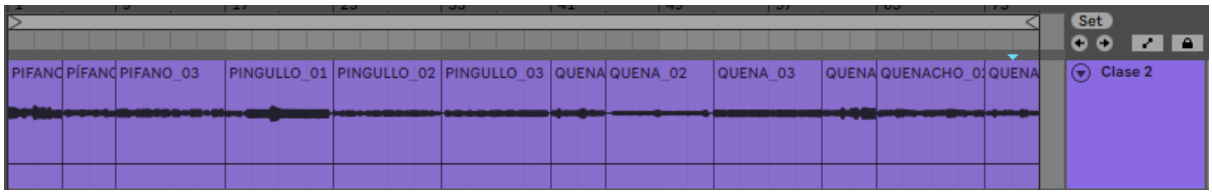


Figura 65. Loop de Ableton Live para la clase 2

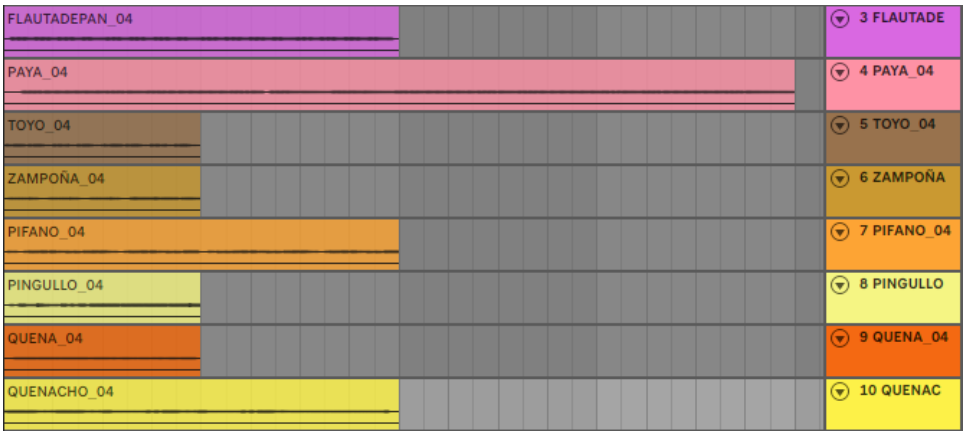


Figura 66. Audios para test set clase 1 y 2.

4. CAPITULO IV: ANÁLISIS DE RESULTADOS

Una de las ventajas de haber realizado el diseño para la base de datos de los instrumentos Andinos es que los sonidos de las dos clases tienen un bajo ruido de fondo, lo que ayuda de gran manera la clasificación. En el actual apartado se indicarán los resultados de los distintos algoritmos de clasificación, así como un análisis de las variables que pudieron haber llevado a que existan mejores resultados unos de otros.

4.1 Recopilación K-Nearest Neighbor

A continuación, se muestran las tablas de recopilación de datos, en los cuales se identifica la duración encontrada por cada clase en cada ejemplo por parte del módulo de salida OSC PWindow, en donde se visualiza los cambios que realiza la clasificación, cada muestra de cada instrumento representa a una pasada del loop en el canal correspondiente de Ableton Live 10. Con estos datos se encontrarán errores en los resultados, los cuales serán promediados por cada clase para encontrar resultados posteriormente. Finalmente se plasmarán los datos correspondientes a los porcentajes de precisión de cada clasificador y concluir cuál sería el más adecuado para realizar clasificaciones con recopilación de datos en un espectrograma de amplitudes por frecuencia.

Tabla 7

Porcentajes de error y precisión del clasificador k-Nearest Neighbor

Muestra	Duración Clase 1	Duración Clase 2	Duración Total	% Error	% Precisión
FP1	8,376%	92%	17	15,576	1,424
FP2	0,000%	100%	17	17	0
FP3	12,565%	87%	17	14,864	2,136
PAYA 1	88,755%	11%	38	4,273	33,727
PAYA 2	85,008%	15%	38	5,697	32,303
PAYA 3	93,463%	7%	38	2,484	35,516
TOYO 1	12,714%	87%	14	12,22	1,78
TOYO 2	28,943%	71%	14	9,948	4,052
TOYO 3	28,457%	72%	14	10,016	3,984
ZAMPOÑA 1	66,460%	34%	15	5,031	9,969
ZAMPOÑA 2	66,460%	34%	15	5,031	9,969
ZAMPOÑA 3	68,833%	31%	15	4,675	10,325
MEDIA	47%	53%	21	8,901	12,099
D.E.	35%	35%	10,31	5,04	13,60

PIFANO 1	0%	100%	21	0	21
PIFANO 2	0%	100%	21	0	21
PIFANO 3	0%	100%	21	0	21
PINGULLO 1	0%	100%	15	0	15
PINGULLO 2	9%	91%	15	1,424	13,576
PINGULLO 3	7%	93%	15	1,068	13,932
QUENA 1	0%	100%	15	0	15
QUENA 2	0%	100%	15	0	15
QUENA 3	0%	100%	15	0	15
QUENACHO 1	0%	100%	22	0	22
QUENACHO 2	0%	100%	22	0	22
QUENACHO 3	0%	100%	22	0	22
MEDIA	1%	99%	18,25	0,20	18,0423
D.E.	3%	3%	3,41	0,49	3,65

Para no hacer este procedimiento repetitivo y extender sin sentido el documento se realiza el mismo procedimiento para todos los clasificadores, indicando en el próximo apartado los resultados pertinentes.

4.2 Duraciones de las muestras

En las siguientes gráficas podemos visualizar las duraciones de las dos bases de datos utilizadas para modelar la evaluación del entrenamiento para la clasificación de sonidos. Se encuentran en un rango de 14 segundos a 38 segundos, los cuales son audios que contienen distintas aplicaciones como sería un caso en la vida real, como por ejemplo al tocar una armonía larga o una armonía corta. Esto ayudará a comprender si existe algún tipo de variabilidad en los resultados si los audios de mayor duración tienen mayor o menor precisión o similar a los de menor duración.

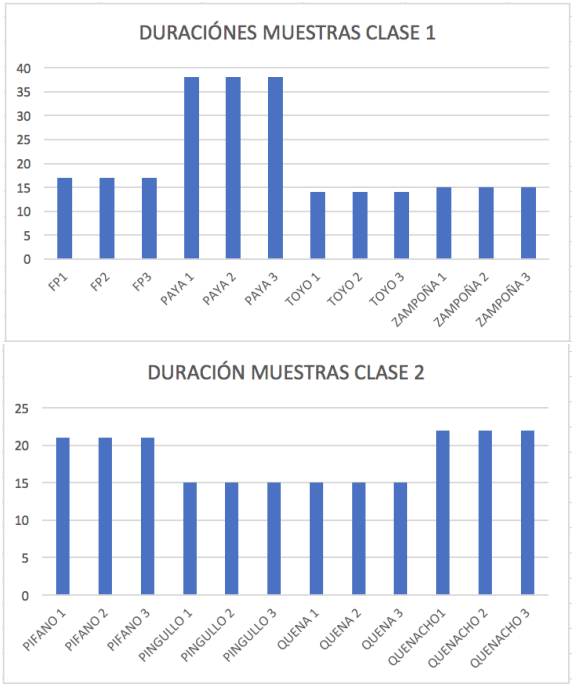


Figura 67. Duraciones de las muestras de las dos clases.

4.3 K-Nearest Neighbor

El clasificador del vecino más cercano tuvo resultados desfavorables en la validación de clasificación en la clase 1 con una media del 53%, caso contrario lo que ocurrió en la clase dos en donde existió una media de 99%. Se realizó la clasificación con k = 1, lo que pudo haber existido una mayor precisión si se hubiese aumentado *k.*

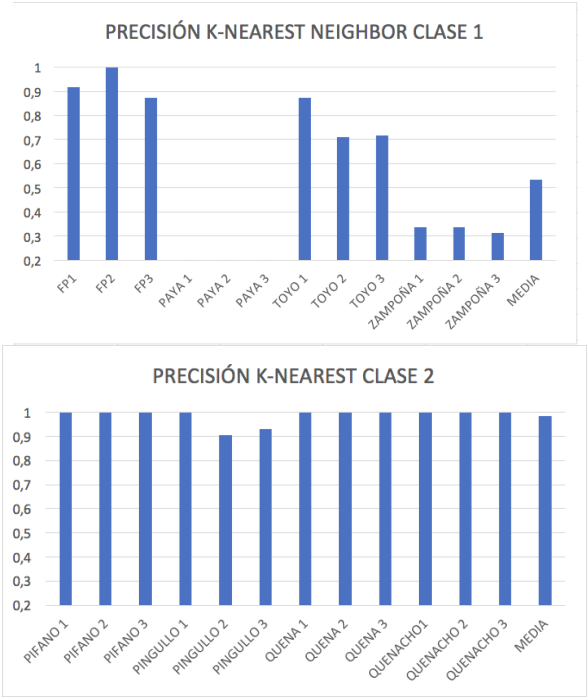


Figura 68. Precisión K-Nearest Neighbour Clase 1 y 2.

4.4 Naive Bayes

Al igual que *k-Nearest Neighbour* existieron malos resultados a la hora de clasificar las tres revisiones correspondientes de cada audio de evaluación en la clase 1, con una media de 53% a diferencia de la precisión en la clase 2 de igualmente el 90%. Hasta el momento no ha existido alguna diferencia notable en los dos clasificadores analizados, lo que indica que no podría ser un factor favorable a tener mayor o menor precisión, más bien se podría asumir que los clasificadores no tienen la misma potencia que deberían para obtener resultados más estables.

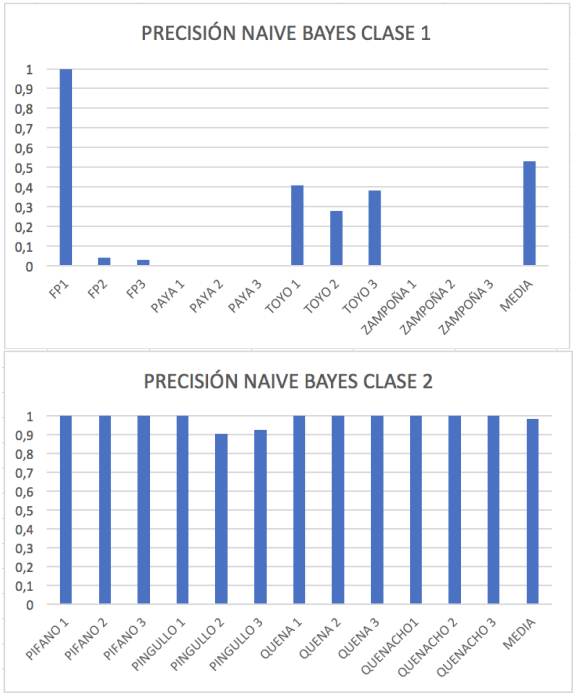


Figura 69. Precisión Naive Bayes clase 1 y 2.

4.5 Árboles de decisión

Como se había mencionado en apartados anteriores a este, uno de los algoritmos que más peso tiene por sus cualidades condicionales y de ajuste, es el de toma de decisiones a través de un árbol de decisiones que realice a través de su algoritmo una clasificación muy aceptable. Los resultados indican un buen comportamiento frente a los datos de entrada que se extrajeron del espectrograma, algo que se habló anteriormente al indicar que sería la manera más óptima cuando se tratase de audios complejos y de aplicaciones más creativas, caso contrario en audios que representen otras cualidades como personas hablando, música, ruidos, etc. la extracción de información musical o acústica sería diferente y más optimizable.

Tenemos dos gráficas que indican mediante una figura las precisiones por cada audio y una media que representa el promedio de todos los *test* para evaluar únicamente el comportamiento en general del clasificador, cabe recalcar que si hubiesen existido una mayor cantidad de audios a evaluar la media hubiese variado, es así que se indica de igual manera una desviación estándar del 9%

en la clase 1 y un 2% en la clase 2, siendo el clasificador más estable de la investigación.

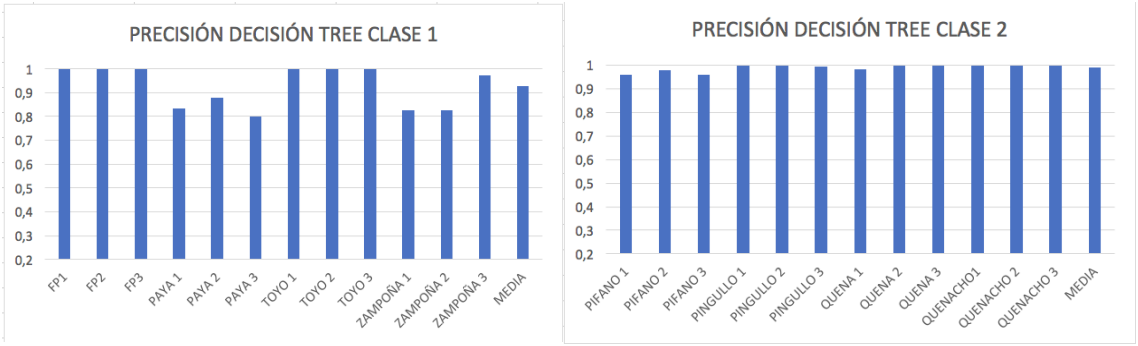


Figura 70. Precisión de árbol de decisiones, clase 1 y clase 2.

4.6 AdaBoost

Precisando otros algoritmos mucho más elaborados y complejos, no solamente en sus modelos matemáticos sino en su entendimiento en general, se brindan resultados más favorables a la estabilidad de clasificación no solamente en evaluaciones de medición sino también en tiempo real, esta es una de las cosas que se han verificado durante toda la evaluación del programa en sí. Podemos ver que la media ponderada de precisión en la clase 1 es de 87% con una desviación estándar del 14%, igualmente obteniendo buenos resultados en la clase dos con un 89% de precisión y una desviación estándar del 15%. En este caso es la primera vez que se observa, que la clase 1 termina logrando una mayor precisión a comparación de la clase 2.

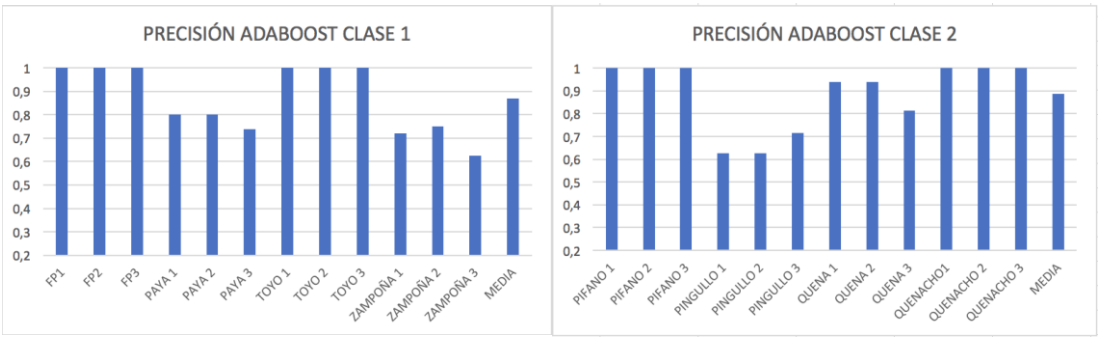


Figura 71. Precisión AdaBoost clase 1 y clase 2.

4.7 Máquina de Vectores de Apoyo

Este es uno de los algoritmos más complicados de entender en la parte de su funcionamiento, pero en la parte gráfica termina siendo uno de los clasificadores más precisos de esta investigación. Como se ve en la siguiente figura se ha escogido una configuración en la dimensión de Kernel, utilizando una ecuación polinomial con un exponente de segundo orden y una constante de complejidad igual a 1. Esta configuración fue suficiente para logra R los resultados indicados en las siguientes figuras.

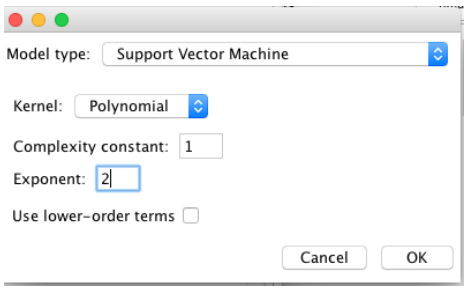


Figura 72. Configuración SVM.

La precisión media ponderada obtenida en la clase 1 es del 84% con una desviación estándar del 18%, por otro lado se tiene la clase dos que tiene una precisión del 97% con una desviación estándar del 5%. Logrando el segundo resultado más preciso y más estable de la investigación.

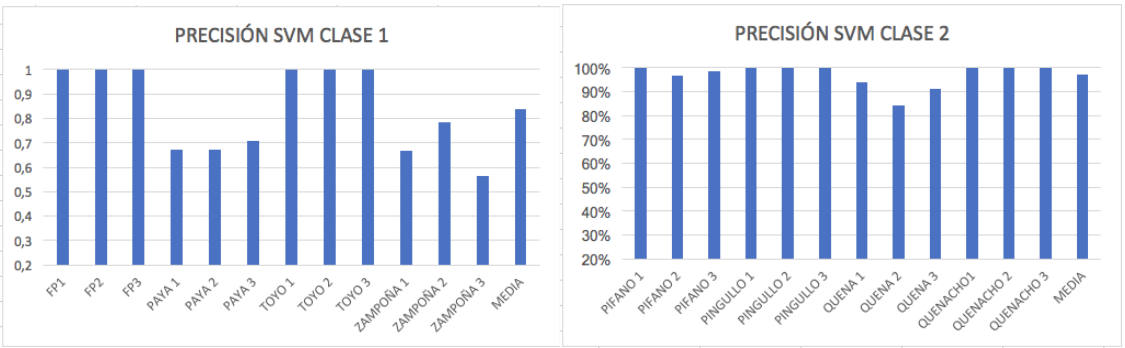


Figura 73. Precisión SVM clase 1 y 2.

4.8 Comparación entre Clasificadores

Esta es una de las partes más importantes de la investigación, ya que por primera vez se observará la comparación de una clasificación para instrumentos andinos de viento en Wekinator, utilizando *Sound Classifier* y verificar cual tiene un mejor comportamiento utilizando una base de datos única y de alta calidad.

En la siguiente figura indicamos los porcentajes de precisión que tiene la clase 1, obteniendo como resultado el árbol de decisiones como el algoritmo que mejores resultados obtuvo, teniendo un 40% de más precisión a comparación del K-N y Naive Boyes. También observamos los algoritmos de AdaBoost y SVM que tienen resultados muy buenos con un 87% y un 84% de precisión respectivamente.

Figura 74. Precisiones de los clasificadores en la clase 1.

Por parte de la clase 2 tenemos tres algoritmos que tuvieron resultados casi perfectos y una desviación estándar entre todos los resultados visiblemente bajos, es decir que la mayoría de algoritmos se comportaron favorablemente hacia los instrumentos de viento largos a comparación de los instrumentos de flautas de pan o resonadores de Helmholtz.

Figura 75. Precisiones de los clasificadores en la clase 2.

Finalmente, los resultados globales indican que al promediar los resultados de las dos clases, es decir de todas las muestras, existe una diferencia notable por parte del algoritmo de árbol de decisiones, obteniendo una diferencia de 20% de precisión más que los algoritmos de precisión más bajos. Una de las cualidades que han permitido decidir que este es el algoritmo con mejor estabilidad es que realiza un buen trabajo al clasificar audios en tiempo real así como en mediciones realizadas en un medidor de tiempo de alguna DAW como fue el caso de Ableton Live 10.

Figura 76. Precisiones globales.

5. CONCLUSIONES Y RECOMENDACIONES

5.1 Conclusiones

En este trabajo se ha logrado la experimentación con varias herramientas no solo de programación, sino con herramientas creativas como es el caso de Ableton Live 10 y Reason 10 con su sampleador NN-XT, el cual ha facilitado de gran manera para la obtención de una base de datos dinámica, por el hecho de poder realizar todo tipo de audios que puedan alimentar o seguir validando a los diferentes algoritmos que utiliza la herramienta de Machine Learning Wekinator.

Estas bases de datos creadas han ocupado un papel muy importante en la clasificación, debido a que poseen la información o los datos que permitirán entrenar al sistema de aprendizaje automático, siendo crucial a la hora de analizar los resultados en algoritmos que necesariamente pueden manejar una gran cantidad de entradas, las cuales se ha permitido en el programa *Sound Classifier.*

Es así que, al hablar de cantidad de entradas, cabe recalcar que no se ha realizado una selección o combinación de extracción de audio de diferente índole como es el ejemplo de los MFCC que pueden aportar cierta información muy útil para tipos de clasificación, dando mérito a que este tipo de información se puede resumir de manera simple en un espectrograma a tiempo real que permita evaluar una clasificación más real y más cercana al trabajo que el cerebro realiza con las redes neuronales. En la actualidad existe la tecnología suficiente como para realizar este tipo de aplicaciones sin limitación alguna, es decir, no existe la necesidad de optimizar ciertos parámetros que se han investigado durante décadas con el objetivo de ejecutarse en todo tipo de computador que pudiese tener limitaciones en su procesamiento, concluyendo

que el espectrograma donde se indican las amplitudes por cada frecuencia en diferentes lapsos de tiempo y que estos parámetros puedan ser ajustados como es el caso de *Sound Classifier* sería la opción más dinámica que pudiese existir en el mercado, aumentando que el programa permite visualizar estas amplitudes con colores que permitan plasmar la información de entrada de tal manera que la interpretación del usuario que lo esté ejecutando sea más clara y más natural.

Por otro lado, los algoritmos de *Machine Learning* utilizados en la investigación, termina destacando los resultados del árbol de decisiones, los cuales terminaron siendo los más precisos y más estables.

Finalmente, cabe decir que los objetivos se han realizado exitosamente, logrando la utilización de diferentes herramientas para validar un clasificador ya ensamblado, obteniendo información que se pueda alimentar en una red de *Machine Learning,* entendiendo lo que se necesita para poder obtener buenos resultados y evaluar cada uno de los algoritmos que Wekinator tiene para ofrecer, utilizando un sistema de verificación por validación cruzada.

5.2 Recomendaciones

El presente trabajo presenta ciertas dificultades por parte de las librerías que se han indicado en la metodología, estas librerías brindan la facilidad de materializar el programa que ingresa los datos a Wekinator, es así que se recomienda tener instalados las respectivas librerías y entender para que sirven cada una de ellas, para de esta manera poder investigar acerca de los problemas de errores que puedan aparecer y solucionarlos.

Así mismo para obtener mejores resultados en algún otro tipo de clasificación con otro tipo de instrumentos se recomienda planificar las clases respectivas, es decir poder identificar los diferentes tipos de instrumentos dependiendo de

sus características físicas y acústicas, como fue el caso de la presente investigación que se propuso dos clases con dos tipos de analogías, así mismo se deben identificar las analogías que corresponde cada instrumento para la creación de una nueva base de datos y realizar el procedimiento de validación cruzada con el objetivo de verificar la precisión de cada clasificador en cada caso.

Finalmente se recomienda tomar el tiempo necesario para poder ingresar la base de datos con la herramienta Ableton Live, el procedimiento se podría definir como un poco tedioso, pero depende del tiempo y la paciencia que se tome para poder entrenar los algoritmos que correspondan influirán en los resultados y los porcentajes finales para el respectivo análisis de resultados.

REFERENCIAS

Aguirre. F. (2017). Desarrollo y análisis de clasificadores de señales de audio. *Master.* Recuperado el 01 de Enero del 2019 de: https://riunet.upv.es/bitstream/handle/10251/90005/Aguirre%20-%20Desarrollo%20y%20an%C3%A1lisis%20de%20clasificadores%20de%20se%C3%B1ales%20de%20audio..pdf?sequence=1

Buioli, I., Pérez, J. (2013) Processing un lenguaje al alcance de todos. (2.ª ed.) Buenos Aires, Argentina: Universidad Nacional de las Artes. Recuperada el 24 de septiembre del 2018, de http://laurence.com.ar/artes/comun/Processing_un_lenguaje_al%20alcance_de_todos.pdf

Chathuranga, D., Jayaratne, L. (2013). *Automatic Music Genre Classification of Audio Signals with Machine Learning Approaches.* República Democrática Socialista de Sri Lanka: University of Colombo School of Computing. Recuperada el 11 de julio del 2018, de https://www.globalsciencejournals.com/content/pdf/10.7603%2Fs40601-013-0014-0.pdf

Colonel, J. (2017). *Improving Neural Net Autoencoders for Music Synthesis.* NY, United States: The Cooper Union for the Advancement of Science and Art. Recuperada el 15 de Septiembre del 2018, de http://www.aes.org/tmpFiles/elib/20190227/19243.pdf

Courtney, F., Lauren, S. (2016). *Meaning-based machine learning for information assurance.* United States: Purdue University. West Lafayette. Recuperada el 22 de octubre del 2018 de https://www.aaai.org/ocs/index.php/FLAIRS/FLAIRS16/paper/download/12969/12577

Falk, C., Stuart, L. (2016). *Meaning-based Machine Leaning for Information Assurance.* West Lafayette, Indiana, United States: Purdue University, Computer and Information Technology. Recuperada de https://www.aaai.org/ocs/index.php/FLAIRS/FLAIRS16/paper/download/12969/12577

Fiebrink, R. (2017). *Machine Learning for Musicians and Artist*. Recuperada 24 de noviembre del 2018, de https://www.kadenze.com/courses/machine-learning-for-musicians-and-artists/info

Gabrielli, L., Cella, C. (2018). *Deep Learning for Timbre Modification and Transfer: An Evaluation Study.* Ancona, Italia: Università Politecnica delle Marche. Recuperada de http://www.aes.org/e-lib/browse.cfm?elib=19513

Ibarra, M. (2016). Zampoña, Lakita y Sikuri en Santiago de Chile: Trenzados y contrapuntos en la construcción de sonoridades andinas en y desde el espacio urbano-metropolitano*.* Universidad de Chile, Facultad de Artes. Recuperada de http://repositorio.uchile.cl/handle/2250/142551

Kuri, F. (2005). Inteligencia artificial*,* Capítulo 11. Recuperada el 12 de octubre del 2018, de http://smia.mx/komputersapiens/download.php?file=ks11_3.4MB_extensa.pdf

Liang, S. (2000). *Recurrent Neural-Network-Based Physical Model for the Chin and Other Plucked-String Instrument.* Hain-Chu, Taiwan: National Chiao-Tung University, Department of Electrical and Control Engineering. Recuperada de http://www.aes.org/e-lib/browse.cfm?elib=12037

Maghsoudi, J. (2017). *A Behavioral Biometrics User Authentication Study Using Motion Data from Android Smartphones.* NY, United States: Pace University, School of Computer Science and Information Systems. Recuperada de https://pdfs.semanticscholar.org/5264/f0b5934c569855d9ba34f86841e59d692cf4.pdf

Martínez, G., Aguilar, G. (2013). Reconocimiento de voz basado en MFCC, SBC Y ESPECTROGRAMAS. Ciudad de México, México: Ingenius "*Revista de Ciencia y Tecnología".* Instituto Politécnico Nacional. Recuperado de https://ingenius.ups.edu.ec/index.php/ingenius/article/view/10.2013.02

Mitchel, P. (1997). *Machine Learning.* Porlant, Oregon E.U. Recuperado el 27 de Septiembre del 2018, de http://profsite.um.ac.ir/~monsefi/machine-learning/pdf/Machine-Learning-Tom-Mitchell.pdf

Mishra, M. (2015). *Task Relationship Modeling in Lifelong Multitask Learning.* Kansas, United States: Submitted to the Departament OF Electrical Engineering & Computer Science and the Graduate Faculty of The University of Kansas. Recuperada de https://kuscholarworks.ku.edu/bitstream/handle/1808/21688/Mishra_ku_0099D_14257_DATA_1.pdf?sequence=1

Mohri, M., Rostamizadeh, A., Talwalkar, A. (2012). *Foundations of Machine Leaning.* Londres, Inglaterra: Cambridge, Massachusetts Institute of Technology. Thomas Dietterich, Editor. Recuperado el 13 de Agosto del 2018, de https://d1rkab7tlqy5f1.cloudfront.net/EWI/Over%20de%20faculteit/Afdelingen/Intelligent%20Systems/Pattern%20Recognition%20Laboratory/PR/Reading%20Group/Foundations_of_Machine_Learning.pdf

Smith, K., Gupta, J. (2011). *Neural networks in business: techniques and applications for the operations researcher.* Clayton, Australia: Monash University, School of Business of Management

Wang, A. (2006). *An Industrial-Strength Audio Search Algorithm.* Palo Alto, United States: Shazam Entertaiment. Recuperada el 15 de Noviembre del 2018 de http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.466.8377&rep=rep1&type=pdf

Zwan, P., Bozena, K., Kupryjanow, (2011). *Automatic Classification of Musical Audio Signals Employing Machine Learning Approach.* A. Gdansk, Polonia: University of Technology, Narutowicza 11/12, (pp. 80-233). Recuperada el 8 de Julio del 2018, de https://www.globalsciencejournals.com/content/pdf/10.7603%2Fs40601-013-0014-0.pdf

ANEXOS

ANEXO I: Código General Sound Classifier

```
////////////////////////////////////////////////////////////////////////////////

import ddf.minim.analysis.*;
import ddf.minim.*;
import oscP5.*;
import netP5.*;
import controlP5.*;

PWindow win;
ControlP5 cp5;

Minim minim;
AudioInput in;
FFT fft;

OscP5 oscP5;
NetAddress dest;

////////////////////////////////////////////////////////////////////////////////
//                  I N I T   V A R I A B L E S                   //

//------------------------------------------------------------------------------------
//FFT variables
int value;
int newValue = 1;
int maxValue = 1;
int vLoc = 0;
int binSize;

. .
//------------------------------------------------------------------------------
//Set the number of inputs to wekinator here.
//This also controls to resolution of the plotter by keeping a 1x1 ratio
// So, 900 inputs returns a resolutin of sqrt(900) = 30 * 30 px. 2500 inputs = resolution of 50 * 50 px
int inputsToWek = 1600;

//------------------------------------------------------------------------------
//layout properties
float r, g, b, w;
int step;
int resolution = round(sqrt(inputsToWek));
PFont f;
int margin, graphStartX, graphStartY;

//------------------------------------------------------------------------------
//Pixel array
color[] Pixels = new int[resolution*resolution];

//------------------------------------------------------------------------------
// FrameRate also controls the speed, and the variable is used to calculate the time graph.
// sp change frameRate here
int frameRate_ = 20;

//------------------------------------------------------------------------------
//Smoothing variables for reducing wrong classification peaks.
int output;
int sampleSize = 4;
int[] classSample = new int [sampleSize];
int index = 0;
int realOutput, oldOutput = 0;

//               E N D   O F   V A R I A B L E S                 //
////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////
//                  S T A R T   O F   S E T U P                           //

public void settings() {
  size(700, 600);
}

void setup() {
  // size(700, 600);
  win = new PWindow();
  background (180);
  frameRate(frameRate_);

  //Setup OSC connection to wekinator
  //-----------------------------------------------------------------------------------
  oscP5 = new OscP5(this, 12000);
  dest = new NetAddress("127.0.0.1", 6448);

  //Minim setup for audio
  //-----------------------------------------------------------------------------------
  minim = new Minim(this);
  in = minim.getLineIn(Minim.STEREO, 2048);  //uncomment to use Line In, live audio

  //start FFTing in three modes: full spectrum, linear avrage and logical avrage
  // But only one is activitly used at a time. See the FFT tab to select which one.
  fft = new FFT(in.bufferSize(), in.sampleRate());
  fft.window(FFT.HAMMING);

  //Layout properties
  //-----------------------------------------------------------------------------------
  margin = width / 24;
  step = round((width-margin*10) / resolution);
```

```
  graphStartX = margin * 5;
  graphStartY = margin * 4;

  f = createFont("helvetica", 12);
  textFont(f);
  textAlign(CENTER, CENTER);

  //CP5 Controls
  //---------------------------------------------------------------------------------
  cp5 = new ControlP5(this);
  sliderSetup();
}

//                      E N D   O F   S E T U P                       //
////////////////////////////////////////////////////////////////////////////



////////////////////////////////////////////////////////////////////////////
//                     S T A R T   O F   D R A W                      //
void draw()
{
  background (180);

  //---------------------------------------------------------------------------------
  //Start the sound analyse
  soundAnalyse();

  //---------------------------------------------------------------------------------
  //Move the pixel array one step, to evolve the "movie".

  for (int r = 0; r < resolution; r++) {
    for (int p = 0; p < speed; p++) {  //change p to make it go faster

  for (int r = 0; r < resolution; r++) {
    for (int p = 0; p < speed; p++) {  //change p to make it go faster
      arrayCopy(Pixels, resolution * r + 1, Pixels, resolution * r, resolution-1);
    }
  }

  //---------------------------------------------------------------------------------
  //Draw the graphics and graphs
  fill(180);
  noStroke();
  drawTimeStamp();

  //---------------------------------------------------------------------------------
  //Draw the spectrogram onto the screen, scaled by the calculated step size (scaling factor)

  for (int x = 0; x < resolution; x++) {
    for (int y = 0; y < resolution; y++) {
      int loc = x + (y * resolution);
      color col = Pixels[loc];
      fill(col);
      noStroke();
      rect(graphStartX+(step*x), graphStartY+(step*y), step, step);
    }
  }

  //---------------------------------------------------------------------------------
  //Send pixels to wekinator
  if (frameCount % 2 == 0) {
    sendOsc(Pixels);
  }
}
```

```
        //                      E N D   O F   D R A W                          //
        ////////////////////////////////////////////////////////////////////////////


        //--------------------------------------------------------------------------------
        //Just a way to stop minim properly on sketch end
        void stop()
        {
          in.close();
          minim.stop();
          super.stop();
        }
```

ANEXO II: Código FFT

```
void soundAnalyse() {

  //--------------------------------------------------------------------------------
  //Perform a forward FFT on the samples in the input buffer.
  fft.forward(in.mix);

  //--------------------------------------------------------------------------------
  //freqCut makes it possible to select a hz range to analyze in the sketch
  binSize = fft.specSize() - freqCut; // raw signal / full spectrum

  //--------------------------------------------------------------------------------
  //Go through all the bands in the fft analyse.
  for (int i = 0; i < binSize; i++)
 {

   //--------------------------------------------------------------------------------
   //VLoc is controlling where on the Y-axe the reading sould be placed.
   vLoc = i*resolution/(binSize);

   //--------------------------------------------------------------------------------
   //Find the amplitude of the index band
   value = (int)Math.round(Math.max(0, 4*20* Math.log10(1000*fft.getBand(int(i)))));

   //--------------------------------------------------------------------------------
   //Uncomment to enable auto tuning/calibrating of the amplitude
   //store self tuning value
   /*   if (maxValue < value) {
    maxValue = value;
    //  println("max amplitude: " + maxValue);
    }
    */

   //--------------------------------------------------------------------------------
   //create a newvalue for the amplite, which lets the cutOff value decide
   //how much of the background noise we want
   newValue = (int)Math.max(0, (value - map(value, 0, maxValue, cutOff, 0)));

   //--------------------------------------------------------------------------------
   //lots of confusing math that acts as a gradient and then set the color
   r = constrain(50+norm(newValue, 0, maxValue)*(370), 0, 255);
   g = constrain(50+norm(newValue, 0, maxValue)*norm(newValue, 0, maxValue)*220, 0, 255);
   b = constrain((70+norm(newValue, 0, maxValue)*(maxValue/2-newValue)), 0, 255);

   //--------------------------------------------------------------------------------
   //set the last row of pixels in the pixel array to the new reading.
   color col = color(r, g, b);
   Pixels[resolution + (((resolution-1)-vLoc)*resolution-1)] = col;
 }
}
```

ANEXO III: Código Gráficas

```
void drawTimeStamp() {

  // A lot of boring coe to create the interfaces graphs and layout
  //-------------------------------------------------------------------------------------
  // Calculating the number of steps and marks we want on the graph
  int bins = 10;
  int stepsXY = resolution * step / bins;
  int y;

  //-------------------------------------------------------------------------------------
  // Draw graph rulers for time (x-axes)
  stroke(100);
  strokeWeight(0.7);
  fill(100);

  y = graphStartY + resolution * step;
  for (int i = 0; i <= bins; i ++) {
    int x = round(((graphStartX) + i * stepsXY));
    line(x, y+10, x, y+15);

    //calculating the time it takes for one row of pixels to travel across the canvas
    float time_ = (((float)resolution/((float)frameRate_*speed)) / bins * i);
    String timer = nf(time_, 1, 2);
    text(timer, x, y+25);
  }

  fill(50, 50, 70);
  text("time (sec)", width/2, y + 50);

  //-------------------------------------------------------------------------------------
  // Draw graph rulers for frequencies (y-axes)
  stroke(100);
  fill(100);
  for (int i = 0; i <= bins; i++) {

    y = round(( height - (height-(graphStartY+resolution*step)) - (i * stepsXY)));
    line(graphStartX-10, y, graphStartX-15, y);

    String freq = str(round(fft.indexToFreq(int((binSize/bins)*i))));
    text(freq, graphStartX-35, y);
  }

  fill(50, 50, 70);
  text("freq (hz)", margin*2-10, height/2);


 //-------------------------------------------------------------------------------------
 // Draw graph for amplitude

 noStroke();
 float r_, g_, b_, w_;
 int ampX = (graphStartX+resolution*step) + 10;
 ;
 int ampY;
 for (int vLoc = 0; vLoc < resolution; vLoc++) {

   r_ = constrain(50+norm(vLoc, 0, resolution)*(370), 0, 255);
   g_ = constrain(50+norm(vLoc, 0, resolution)*norm(vLoc, 0, resolution)*220, 0, 255);
   b_= constrain((70+norm(vLoc, 0, resolution)*(resolution/2-vLoc)), 0, 255);

   fill(color(r_, g_, b_));
   ampY = round(( height - (height-(graphStartY+resolution*step-step)))- (vLoc * step));
   rect(ampX, ampY, margin, step);
 }
 fill(50, 50, 70);
 text("Amplitude", width-margin*2, height/2);
```

```
  stroke(100);
  fill(100);
  //line(ampX + margin*2.1, graphStartY, ampX + margin*2.2, graphStartY);
  text(maxValue+" Amp", ampX + margin*2.2, graphStartY);
  text("0 Amp", ampX + margin*2.2, height - (height-(graphStartY+resolution*step)));
  text("inputs to wek: " + Pixels.length, graphStartX + margin*15, margin*1, 5);
}
```

ANEXO IV: Código OSC

```
//--------------------------------------------------------------------------------
//Send the spectogram array as a msg to wekinator
void sendOsc(color[] px) {
  OscMessage msg = new OscMessage("/wek/inputs");
  // msg.add(px);
  for (int i = 0; i < px.length; i++) {
    msg.add(float(px[i]));
  }
  oscP5.send(msg, dest);
}

//--------------------------------------------------------------------------------
//Receive inputs from wekinator
void oscEvent(OscMessage theOscMessage) {
  println("print something");
  if (theOscMessage.checkAddrPattern("/wek/outputs")==true) {
    output = int(theOscMessage.get(0).floatValue());

    //smooth incoming inputs from wekinator. if response time is to slow, increase frameRate,
    //or lower the sampleSize variable
    output = avrOutput(output);
  }
}


//--------------------------------------------------------------------------------
//Smoothing algortigm to smooth incoming wek values.
int avrOutput(int output_) {
  boolean match = true;
  classSample = splice(classSample, output_, 0);
  classSample = subset(classSample, 0, sampleSize);
  println(classSample);

  int first = classSample[0];
  for (int i = 1; i < classSample.length; i++) {
    if (classSample[i] != first) match = false;
  }

  if (match) {
    realOutput = output_;
    oldOutput = output_;
  } else realOutput = oldOutput;

  return realOutput;
}


//--------------------------------------------------------------------------------
//SEND OUTPUT NAMES BACK TO WEKINATOR
void sendOscNames() {
  OscMessage msg = new OscMessage("/wekinator/control/setOutPutNames");
  msg.add("gesture");
  oscP5.send(msg, dest);
}
```

ANEXO V: Código PWindow

```
//This Pwindow is dedicated to create sketches using the wekinator classification
//seperated from the spectrogram tool.

class PWindow extends PApplet {
  PWindow() {
    super();
    PApplet.runSketch(new String[] {this.getClass().getSimpleName()}, this);
  }

  //---------------------------------------------------------------------------------
  // Set sketch variables
  color[] colors = {#FAF3DD, #FFF05A, #FF6663, #4EA699, #9DD1F1, #30332E};

  //---------------------------------------------------------------------------------
  // Set the window size here
  void settings() {
    size(500, 500);
  }

  //---------------------------------------------------------------------------------
  // Set seutp properties
  void setup() {
    background(colors[0]);
    textAlign(CENTER, CENTER);
  }

  //---------------------------------------------------------------------------------
  // Start of draw
  void draw() {

    switch(output) {
    case 1:
      background(colors[0]);
      break;

    case 2:
      background(colors[1]); // - "yellow"
      break;

    case 3:
      background(colors[2]); // - "red"
      break;

    case 4:
      background(colors[3]); // - "green"
      break;

    case 5:
      background(colors[4]); // - "blue"
      break;
    }

    text("output " + output, width/2, height/2);
  }
}
```

ANEXO VI: Código Controles

```
void sliderSetup() {

  cp5.addSlider("cutOff")
    .setPosition(graphStartX, margin*1)
    .setSize(100, 20)
    .setValue(400)
    .setRange(0, 1000)
    ;

  cp5.addSlider("maxValue")
    .setPosition(graphStartX +150, margin*1)
    .setSize(100, 20)
    .setValue(400)
    .setRange(200, 600)
    ;

  cp5.addSlider("speed")
    .setPosition(graphStartX, margin*2.5)
    .setSize(100, 20)
    .setValue(5)
    .setRange(0, 10)
    ;


  cp5.addSlider("freqCut")
    .setPosition(graphStartX +150, margin*2.5)
    .setSize(100, 20)
    .setValue(0)
    .setRange(0, 800)
    ;

}
```