UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Programa de Maestría y Doctorado en Música

Facultad de Música
Instituto de Ciencias Aplicadas y Tecnología
Instituto de Investigaciones Antropológicas

Inteligencia artificial y minería de datos aplicadas al análisis musical

TESIS
QUE, PARA OPTAR POR EL GRADO DE
DOCTOR EN MÚSICA, TECNOLOGÍA MUSICAL

PRESENTA
CRISTIAN MANUEL BAÑUELOS HINOJOSA

TUTOR O TUTORES PRINCIPALES
Dr. Felipe Orduña Bustamante – ICAT

MIEMBROS DEL COMITÉ TUTOR
Dr. Nicolás Kemper Valverde – ICAT
Dr. Roberto Morales Manzanares – Universidad de Guanajuato

CIUDAD DE MÉXICO, OCTUBRE 2019

Para Elisa Borrero

Este trabajo es el resultado de muchas aventuras y exploraciones, así como apoyo de tantas personas con las que estoy muy agradecido. Les dedico este trabajo y años de esfuerzo a cada uno de ustedes. Agradezco a la *UNAM* por recibirme y permitir seguir mis interéses de investigación y crecimiento. Gracias a Elisa, la razón y motivación para salir adelante, te amo. Muchas gracias a mi familia, por el apoyo incondicional, a mi mamá María, a mi papá Manuel, mi hermanita Stephanie y mi cuñado Alejandro. Gracias a mis suegros, a la Sra. Wanda†, al Sr. Borrero, Ivette, a mis cuñados Necha, José. Muchas gracias a mi tutor, Felipe Orduña por guiarme con mucha paciencia, soy muy afortunado de trabajar con usted. Gracias a Enrique Díaz, por ser un gran amigo, estar siempre atento y darnos su apoyo y consejo. Gracias a Josué, Carolina, Guillermo, Carla, Pedro, Otto, Adela, Anabel, Liliana, Anya, Gudrun, Sol, Esteban, Itze, Dalila, Emanuel, Andrea, Maby, Pablo, María Clara, Itai, Yossi, Oriam y Cleo. Muchas gracias a mis maestros, que generosamente han compartido sus conocimientos y me pasan la estafeta para yo hacer lo mismo, gracias a Nicolás Kemper, Roberto Morales, Consuelo Carredano, Eunice Padilla, Hugo Solís, Fabiola Miroslaba, Jorge David, Evgeny Korolkov, Irina Shishkina, Pablo Silva, Álvaro Díaz. Muchas gracias al posgrado en música al Dr. Fernando Nava y al Dr. Roberto Kolb. Muchas gracias a Jasmin por su apoyo y amistad invaluables. Muchas gracias a ti, que estás leyendo esto, que aunque no esté tu nombre, estás en mi corazón. Muchas gracias a mi país, por darme la oportunidad de tener una educación de alta calidad, este trabajo fue hecho posible por el programa de apoyo del Consejo Nacional de Ciencia y Tecnología CONACYT, Número de beca: CVU 662627 no: 583329. Se agradece el apoyo recibido para la realización de esta tesis por parte del proyecto PAPIME PE405018 “Desarrollo e implementación didáctica de recursos informáticos y de cómputo para apoyar la enseñanza de la Música”.

Es curioso que, con el riesgo de sonar cursi, siento que al terminar este ciclo, más que un final, se siente como un nuevo inicio, al que voy con muchas fuerzas para continuar el trabajo y la investigación para contribuir al conocimiento libre.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE CUADROS

ÍNDICE DE CÓDIGOS

INTRODUCCIÓN

Artificial Intelligence is
the New Electricity

Andrew Ng

La inteligencia artificial y minería de datos se han convertido silenciosamente en parte importante de nuestras vidas. Poco a poco, en menos de veinte años se han incorporado a nuestro flujo de trabajo y vida cotidiana herramientas que nos ahorran mucho tiempo. En nuestras computadoras personales o teléfonos celulares, habitualmente realizamos operaciones computacionales complejas. Por ejemplo, desde un buscador de texto como Google, al introducir algunas palabras claves podemos encontrar una gran cantidad de imágenes y videos de nuestro interés. En las fotos que subimos a las redes sociales, automáticamente se nos sugiere etiquetar a nuestros conocidos o agregar información de dónde se tomó una foto. En aplicaciones musicales como Spotify, se nos muestran canciones y música que el sistema cree que nos podrían gustar. El punto común en todos, es que analizan nuestra información por medio de bases de datos con algoritmos de minería de información e inteligencia artificial para llegar a resultados que sean útiles.

Por medio de los avances teóricos y tecnológicos de estas dos áreas se ha abierto la puerta para trabajar con grandes

cantidades de datos y realizar análisis que antes serían simplemente imposibles. Por otra parte, estas herramientas tan poderosas están disponibles para cualquier persona con acceso a una computadora e internet y el poder de cómputo ha llegado al punto, que es fácil hacer cálculos muy intensos con un teléfono celular o una tableta.

Además de las aplicaciones sociales, esta tecnología también es utilizada en ambientes académicos y de investigación para encontrar patrones y relaciones ocultas entre datos, principalmente en el área de economía y finanzas. En cuanto a la aplicación de estas tecnologías al estudio de la música, los avances han sido bastante modestos. Con lo anterior, surge la pregunta ¿podría la musicología hacer uso de estas tecnologías tan poderosas para abordar problemas existentes y trabajar sobre proyectos que son inmanejables a mano y sobre papel? Creemos que actualmente las condiciones tecnológicas se prestan para ayudar a resolver estos problemas de manera eficiente y contribuir al trabajo de los musicólogos.

Actualmente, las herramientas computacionales más utilizadas por músicos son los editores de partitura y editores de audio. En cuanto a programas que realicen análisis profundo de obras, existen diversos ejemplos de los cuales hablaremos más adelante. En muchos de ellos, sus autores hacen un gran esfuerzo por traducir los conceptos computacionales requeridos para la música a una interfaz gráfica amigable para el usuario, de modo que se pueda evitar que el músico tenga que programar para obtener resultados. A pesar de ello, es muy común que estos desarrollos sean aislados y poco utilizados. Una de las principales razones es que, sin importar lo amigable que sea la interfaz gráfica, existe una fuerte

curva de aprendizaje, la cual aleja a los músicos de usar herramientas tecnológicas complejas. Proponemos que un analista computacional sería un aliado poderoso para ciertas partes del trabajo musicológico, como el análisis de corpus grandes de partituras para la búsqueda de similitudes y clasificaciones complejas, sin embargo, es necesario un trabajo profundo en conjunto para encontrar las mejores formas en que esta sinergia puede ser efectiva.

El objetivo de la tesis es desarrollar e implementar un toolkit y una interfaz gráfica para que el músico pueda utilizar minería de datos y la inteligencia artificial en su trabajo cotidiano. Como se sostiene a lo largo de este trabajo, existen pocos intentos recientes de acercar al intérprete y al musicólogo a estas herramientas que podrían simplificar muchos de los problemas en los que se requiere trabajar directamente sobre una partitura o grabaciones de audio. En este trabajo se muestran algunas de las necesidades y beneficios de incluir técnicas computacionales especializadas para el estudio y análisis de la música en partituras digitales, en particular, con el uso de técnicas de inteligencia artificial y minería de datos. Por medio de una serie de problemas y sus respectivas propuestas de algoritmos y soluciones, se propone que la musicología podría beneficiarse con la implementación de herramientas diseñadas a la medida para abordar problemas de extracción y análisis de información musical. Estas se montan en un sistema en línea al cual el usuario puede acceder desde una computadora personal, teléfono inteligente o tableta con un explorador de internet actualizado. Los cálculos y algoritmos se realizan en un servidor remoto, de modo que el usuario puede usar el sistema sin tener que programar

ni preocuparse por cuestiones técnicas de las características, configuración y desempeño de los equipos de cómputo. Esto es un gran beneficio, permite que un musicólogo tenga acceso a herramientas computacionales especializadas y avanzadas para el uso del análisis musical de forma sencilla.

Para acercar al músico a las herramientas de inteligencia artificial y minería de datos utilizamos un sistema de dos partes. Por un lado, una serie de herramientas optimizadas que implementen estas técnicas de cómputo, y una interfaz de usuario fácil de navegar. En este trabajo logramos esto con un motor de cálculo desarrollado en *Python*, junto un sistema de tipo cliente servidor en *Node.js* y *React.js*. Por medio de esta arquitectura orientada a servicio, el sistema puede instalarse localmente en una computadora personal o en un servicio de cómputo en la nube. De esta forma, el usuario es capaz de acceder e interactuar con el sistema de forma remota por medio de un navegador de internet moderno, sin necesidad de instalar ningún programa extra, lo cual reduce drásticamente la curva de aprendizaje.

A lo largo del texto, se muestran algunos de los beneficios de que el musicólogo agregue las herramientas computacionales especializadas a su flujo de trabajo. Se debate entre dos ideas para lograr este acercamiento: por un lado, podría incluirse en la formación de un musicólogo algunos antecedentes tecnológicos adecuados que le permitan navegar por el mundo tecnológico; por otro lado, se podría considerar que en el trabajo musicológico se incluyera a un especialista o consultor del tipo “analista musicológico computacional”, quién servirá de puente entre los problemas a enfrentar y las herramientas mencionadas. La segunda opción parece más

viable, ya que se podría tomar a algún experto en análisis de datos computacionales y darle una capacitación musical adecuada para que entienda los problemas propuestos por la musicología. En general, se necesita de un trabajo interdisciplinario para lograr resultados interesantes. Se requiere la interacción de distintas disciplinas: cómputo, musicología, inteligencia artificial, minería de datos y diseño de software, de modo que se tenga un equilibrio entre las necesidades, expectativas y realidades. A lo largo del desarrollo de este trabajo se buscó la colaboración en conjunto de expertos en música, así como expertos en cómputo, para asegurarnos que el sistema aborde problemas relevantes, de interés y factibles.

Es muy importante especificar que en este trabajo analizamos únicamente archivos de partituras digitales, es decir, archivos digitales de música que contienen información equivalente a una partitura en cuanto a instrucciones de las notas que deben ser interpretadas en tiempos especificados. Estos archivos no contienen información de audio. A estas representaciones de la música se les conoce como Representaciones Simbólicas de la Música. A lo largo de este trabajo llamaremos Archivos simbólicos a este tipo de archivos musicales y a sus contenidos los llamaremos Información Simbólica. En específico trabajamos con los formatos *MIDI* y *MusicXML*, los cuales se describen más adelante.

La razón por la que preferimos los archivos simbólicos a los de audio como *mp3* o *wav*, es debido a que nos interesa analizar la estructura de notas y duraciones. En los archivos simbólicos esta información aparece naturalmente, mientras que en los archivos de audio, es necesario utilizar algoritmos que traten de inferir la información simbólica, lo

cual representa otro campo de investigación que difiere del nuestro.

El sistema que se propone en este trabajo es módular y expandible. Cada módulo se especializa en resolver una tarea de análisis computacional de la música. El usuario interactúa con ellos por medio de una interfaz gráfica remota sin tener que instalar nada en su computadora más que un navegador de internet actualizado. La importancia de la modularidad, es que el sistema puede extenderse gradualmente con nuevos módulos de análisis desarrollados por otros investigadores. Esto permite que el sistema pueda crecer de forma colaborativa. Se plantea que el servidor y motor de cálculo esté montado en una plataforma de cómputo en línea, sin embargo, se incluyen instrucciones en caso de que algún investigador deseara implementarlo localmente en su propia computadora o red local. A en las siguientes secciones se muestran cuatro ejemplos de las aplicaciones de la inteligencia artificial y minería de datos que se implementaron en el sistema en línea mencionado anteriormente. Los algoritmos desarrollados son exclusivamente para el estudio de partituras digitales, es decir música simbólica.

En el sistema, se implementaron los siguientes módulos.

- Módulo de representación digital de la música y base de datos. Aquí se introducen los conceptos de bases y minería de datos.

- Módulo de análisis armónico y etiquetado semi-automático[1] de acordes. Se utiliza clasificación por lógica difusa y redes neuronales.

- Módulo de análisis de estructura y similitudes de fragmentos y obras musicales. Aquí se utiliza clasificación y clustering por medio de clasificadores con redes neuronales y funciones de similitud.

- Módulo de clasificación de archivos de acuerdo a reglas y ejemplos. Se utiliza como referencia el tratado de guitarra barroca de Gaspar Sanz: *Instrucción de música sobre la guitarra española y métodos de sus primeros rudimentos hasta tañerla con destreza* . En este módulo se implementa un clasificador por medio de un sistema experto.

- Módulo de análisis de textos y su relación con el contenido musical. Aquí se utiliza un sistema de búsqueda y clasificador de contenido musical.

- Módulo de análisis de partes y densidad de voces. En este se crea un análisis de partes solistas, duetos y otras combinaciones de voces para encontrar patrones musicales.

- Sistema administrador de módulos e interfaz de usuario.

La tesis se divide en siete capítulos. En el capítulo 1, mostramos los antecedentes teóricos y tecnológicos sobre los

1 Semi automático se refiere a que el sistema provee una sugerencia de acordes pero el usuario lo puede calibrar y corregir

cuales se basa este trabajo. Se presentan algunos conceptos musicológicos y de análisis musical que son necesarios como contexto para desarrollar los sistemas propuestos. Así mismo, se muestran las bases y terminología necesaria para tener un lenguaje común en el uso de la inteligencia artificial y la minería de datos.

El capítulo 2, presenta algunos de los principales problemas que surgen al representar el contenido de partituras en una forma digital, así como las metodologías usadas para trabajar y analizar los archivos. Se presentan las técnicas de pre procesamiento de información, así como los métodos para traducir el contenido de las partituras digitales a formatos de bases de datos, sobre la que aplicaremos los algoritmos en el resto de los módulos.

Al pasar al capítulo 3, se desarrollan los algoritmos necesarios para el módulo encargado de la extracción y etiquetado automático de acordes. Se comparan los beneficios y desventajas del uso de lógica difusa, redes neuronales, cadenas de Markov y otras técnicas, proponemos un algoritmo que funciona adecuadamente para analizar acordes.

En el capítulo 4, se presentan los algoritmos necesarios para el módulo de comparación y medidas de similitud para el análisis de la estructura musical. Se muestra una implementación de similitud con una variante del algoritmo de ajuste de tiempo dinámico (*Dynamic Time Warping, DTW*) y se utiliza para extraer información estructural de partituras digitales.

En el capítulo 5 se presentan ejemplos de módulos que implementan algoritmos de búsqueda y comparación en un corpus de partituras. Se utiliza como ejemplo una misa de

Tomás Ochando, en colaboración con la musicóloga Andrea Zamora (FAM-UNAM).

El capítulo 6, se enfoca en la creación del módulo de clasificación de piezas por medio de una serie de reglas implementadas en un sistema experto. Como ejemplo se toma el tratado de guitarra barroca de Gaspar Sanz, del cual se obtienen resultados satisfactorios en cuanto a la clasificación automática.

En el capítulo 7, se describen varias cuestiones importantes acerca de la instalación del sistema modular. Se enlistan los requerimientos y pasos necesarios para instalarlo localmente, así como los necesarios para montarlo en un servicio en la nube. De esta manera existe flexibilidad acerca del uso posible que se le pueda dar tanto por usuarios personales, como por instituciones académicas y de investigación.

Por último se presentan las conclusiones y las reflexiones del desarrollo del sistema, así como una muestra de trabajo futuro. Este texto está dirigido a dos tipos de lectores, principalmente: el musicólogo sin experiencia en programación, así como investigadores de tecnología musical. Dado el nicho tan estrecho que esto abarca, puede que durante la lectura haya partes más técnicas y otras más descriptivas. Los fragmentos de código que se anexan sirven como una referencia breve para el lector más tecnológico, sin embargo, para los lectores no familiarizados con el código, se pueden omitir sin perder el flujo de las ideas. A continuación pasamos a la revisión del estado del arte de las técnicas de inteligencia artificial y minería de datos aplicadas al análisis musical.

1

ANTECEDENTES

En los desarrollos científicos y tecnológicos que nos han llevado a crear máquinas y sistemas que simplifiquen nuestra vida, hemos tenido grandes avances y creado herramientas que extienden nuestras capacidades, tanto físicas como intelectuales. Las máquinas nos han superado en algunos aspectos, lo cual nos ha llevado a cuestionar si algún día nos reemplazarán completamente. En este sentido comenzamos a ver en ellas reflejos de nosotros, pero aun hay una gran diferencia entre un humano y una máquina.

Para que una máquina sea considerada con características similares a un humano, debería mostrar atributos como adaptabilidad, flexibilidad y autonomía, sin embargo, surge la pregunta acerca de si en verdad nos beneficiaría que las máquinas se comportaran como humanos. No es claro el contexto en que sería aceptable que una computadora tenga la capacidad de desobedecer. De momento si algún programa de cómputo no hace lo que deseamos, la razón principal es que hubo un error en la programación, sin embargo si una computadora tuviera la capacidad de decisión autónoma y libre, podría simplemente negarse a obedecer.

Existen casos donde la programación de un sistema inteligente queda corta en comparación con el comportamiento humano. Por ejemplo, en los sistemas de conducción automático de automóviles surgen problemas al introducir agentes programados a seguir reglas específicas de tránsito. Surge el caso en que algunas veces el auto permanecía en un cruce esperando su turno para cruzar, pero quedaba inmóvil ya que los demás autos tomaban la iniciativa y nunca existía el momento ideal de acuerdo a las leyes de tránsito estrictas que le fueron programadas. En este sentido, para que los sistemas inteligentes simulen al humano, deben tomar en cuenta tanto al usuario que sigue todas las reglas, así como los casos donde las reglas son flexibles para obtener los resultados correctos. Para que un sistema computacional sea más humano, también tiene que ser social en cierta manera, no con nosotros, sino con otros sistemas, que tenga la capacidad de compartir información, y pueda aprender nuevas cosas sin que le tengamos que programar.

Mas allá de las distopías tecnológicas de la ciencia ficción, donde las máquinas reemplazan al humano y acaban con él, es probable que se llegue a un punto que, de programarlas a nuestra semejanza, habrá máquinas "buenas" y "malas". Sería interesante cuando llegara el momento, que aprendamos cosas nuevas de un sistema inteligente de la misma manera que un padre aprende de su hijo. Lo más importante, si queremos que sean mas humanas, debemos también aprender a verlas no como algo que podemos usar y obtener un beneficio, sino como "alguien" con quien podemos relacionarnos, compartir y aprender.

En este contexto, introducimos y mostramos las necesidades de incluir la inteligencia artificial y minería de datos en el estudio de la música, como una serie de herramientas que complementan y no que reemplazan el trabajo humano. Estas técnicas sirven para ayudar a que el musicólogo pueda agregar argumentos a sus decisiones, con nuevos datos que surgen del análisis de la partitura. El objetivo de este trabajo doctoral es por un lado simplificar algunas áreas del trabajo del musicólogo y analista, y por otro, abrir las puertas a ciertos estudios que serían imposibles de realizar en el papel, debido a la revisión exhaustiva que requieren.

Uno de los trabajos con el que más coincidimos en ideas y métodos es el de McKay 2010, donde se desarrolla el sistema *jMIR*, con el cual compartimos mucha de la motivación para realizar este tipo de análisis digital de la música. Por medio del desarrollo de una biblioteca de código McKay, se encarga de extraer representaciones numéricas de la música que ayudan a realizar análisis interesantes. Su investigación nos servirá de referencia a lo largo de este trabajo.

McKay 2010, sostiene que no se puede esperar que un sólo individuo sea experto en tantas áreas del conocimiento, hay necesidad de integrar las metodologías. Sugiere también que es muy importante trabajar en archivos de representación simbólicas por varias razones: tienen información de alto nivel, son fáciles de extraer.

Como documento histórico, uno de los primeros intentos de aplicar métodos computacionales al análisis musical, tenemos el trabajo de Erickson 1968, donde se hace énfasis en la necesidad de que los musicólogos se familiaricen con los avances tecnológicos y utilizan una representacion de

la música llamada *Ford-Columbia language*. Utilizaban un lenguaje llamado *MIR*, pero tenían el problema de que sólo podía usarse en dispositivos especializados, por lo que era difícil reproducirse. Es muy interesante la idea de que desde los años sesentas cuando la era computacional estaba en sus inicios, ya había investigadores que tomaron conciencia de la importancia que esto podría traer a las artes. De acuerdo a ellos, la ventaja de utilizar una computadora no era hacer los procesos más rápido, sino extender el rango de problemas que se pueden abordar.

> *It should be stressed, therefore, that a computer may not produce a faster solution to a given problem; rather, its unique value is in broadening the range of problems that can be posed and solved.*[1] Erickson 1968.

En 1993 ya se tenía una concepción del concepto de musicología asistida por computadora, como podemos ver en el trabajo de Bel y Vecchione 1993, donde se menciona la creación de un área transdisciplinaria nueva a la que llaman *Musicología computacional*. Esto es muy interesante, ya que en aquellos años, las herramientas tecnológicas eran bastante limitadas, pero aún así se intentaban aplicar. Estos autores sostienen que para que dichas herramientas tengan éxito en la musicología, se deben trabajar en conjunto con los músicos, ya que hay una brecha muy grande entre la tecnología y los investigadores musicales que impide su exploración y aplicación.

1 Se debe enfatizar que quizá una computadora no produce una solución más rápida a un problema, sino que su valór único es extender el rango de los problemas que se pueden abordar y resolver. Traducción propia.

> *Since scholars in the humanities are not typically trained in the development or even the use of computing technology, there is a gap to bridge in order to make systems accessible to music researchers and to enable scholars to develop questions and seek answers that can be approached with the growing digital datasets and emerging computational tools.*[2] Bel y Vecchione 1993.

Los sistemas de inteligencia artificial y minería de datos cuentan con herramientas que pueden ayudar con algunos procesos que se utilizan para el análisis musical. Uno de los principales problemas de aplicar técnicas computacionales a procesos manuales es decidir hasta qué punto debe o no automatizarse una tarea. En el trabajo de Anagnostopoulou y Buteau 2010 se presenta una reflexión acerca de este punto, con ejemplos donde muestran que las herramientas computacionales deben ser siempre un apoyo para tomar decisiones musicológicas y no un sistema cerrado que entregue resultados categóricos.

ANTECEDENTES TECNOLÓGICOS

Ahora pasemos a la descripción de los conceptos tecnológicos que necesitaremos a lo largo de este trabajo, de modo que

2 Dado que los académicos en las humanidades típicamente no tienen un entrenamiento en el desarrollo o incluso el uso de tecnología computacional, existe una brecha que superar para hacer los sistemas accesibles a la investigación musical y habilitar a los académicos a desarrollar preguntas y buscar respuestas que puedan ser abordadas con los conjuntos de datos y herramientas conputacionales que se encuentran creciendo y emergiendo. Traducción propia.

tengamos un lenguaje común sin importar si venimos del lado de la musicología o la computación. Por un lado, la minería de datos consiste en la extracción de conocimiento de una colección organizada de información. Es decir, si tenemos una base de datos, queremos extraer de ella conocimiento o información que no es obvia, por medio de algoritmos especializados. Por otro lado, la inteligencia artificial se refiere a los procesos por los cuales se intenta emular la inteligencia humana a través de una máquina, en este caso, una computadora. Existen muchas ramas de la inteligencia artificial entre ellas, los sistemas expertos, redes neuronales, lógica difusa y procesamiento de lenguaje natural, entre otras.

Desde el punto de vista musical, el desarrollo que las herramientas computacionales han tenido en años recientes es sumamente importante y oportuno. Tenemos a nuestra disposición miles de archivos con información musical muy variada, desde archivos de audio (*.wav, .mp3*), partituras en distintos formatos (*.pdf, .jpg, MusicXML*), así como archivos con información de eventos musicales en una pieza (*MIDI*), de los cuales se hablará más adelante. Todos estos archivos se encuentran al alcance de cualquier persona con acceso a internet. Actualmente se encuentran los elementos necesarios para desarrollar herramientas de cómputo que permitan estudiar, clasificar y trabajar con todos estos datos e información musical, de modo que tengamos un mejor control y entendimiento de las relaciones que surgen al estudiar integralmente todo este contenido musical digital.

Para manejar y dar sentido a la complejidad que surge de trabajar con los registros digitales gigantes de las bases de datos modernas, ha sido necesario crear algoritmos de

clasificación, búsqueda y reconocimiento de información. Tanto la inteligencia artificial como la minería de datos se especializan en estos procesos, por lo cual su aplicación es una consecuencia natural.

La inteligencia artificial (IA) y la minería de datos (MD) comúnmente se aplican para el análisis de datos de negocios. Entre ellos, un uso muy frecuente es el estudio del comportamiento de clientes, como el saber qué productos suelen comprarse en conjunto. También se aplican para realizar búsquedas especializadas en grandes bases de datos, así como sistemas de recomendación, donde se sugieren al usuario elementos que sean similares a lo que consume frecuentemente. Este tipo de aplicación tiene implicaciones éticas con respecto a la privacidad de los usuarios que deben ser consideradas con cuidado. Sin embargo, como herramienta, abre las puertas a análisis de tendencias que pueden dar resultados muy interesantes.

Otra técnica utilizada en minería de datos para encontrar asociaciones entre datos se llama reglas de asociación, la cual establece relaciones entre aparición de elementos en una lista. Por ejemplo, se suelen usar en análisis de listas de compras, donde se asocia la frecuencia en qué elementos aparecen juntos en las compras.

Existe diversos trabajos de investigación que abordan la aplicación de la minería de datos e inteligencia artificial a la música. En cuanto a la primera, el trabajo de Li, Ogihara y Tzanetakis 2011 es una referencia fundamental, ya que hace una revisión profunda de las principales técnicas utilizadas actualmente. Lo más importante de su trabajo es que sirve de conexión entre la minería de datos en general y las cuestiones

necesarias a considerar para su aplicación musical. Los pasos requeridos para un análisis de este tipo son tres: administración, pre procesamiento y minería. El primero consiste en conseguir la información de una fuente confiable y organizarlos para su lectura. En el segundo se deben preparar y limpiar los datos para que los algoritmos de búsqueda puedan lograr su objetivo de forma eficiente[3]. Por último para la minería, se analizan los datos preparados por medio de búsqueda de secuencias, clasificación, *clustering*, similitud y visualización de los mismos.

Para clasificar la información musical de archivos digitales, es necesario tener representaciones matemáticas que sirvan de base para la clasificación y búsqueda. Se han hecho estudios muy detallados de este tipo de métodos, como en Tymoczko 2009; Tymoczko 2011, Mazzola y col. 2002, Szeto y Wong 2006, Toussaint 2010, Chew 2014, entre otros. Estos nos ayudan a poder manejar los conceptos abstractos de la música de forma numérica de modo que con ellos podemos organizar nuestra información en las bases de datos de acuerdo a funciones de similitud y medición de cercanía abstracta.

Tenemos también los trabajos que, en vez de clasificar, intentan obtener información musical que no se encuentra directamente en el archivo. Por ejemplo, en un archivo de audio se tiene la intensidad de la onda en cada punto, pero no hay información acerca del tempo o tonalidad de una pieza. Si se desea obtener esa información, se debe hacer uso de algoritmos especializados. A esta área de investigación se

3 Por ejemplo, se aplican técnicas de reducción de dimensiones, eliminación ruido, entre otros.

le conoce como *Music Information Retrieval (MIR)*[4]. Estas técnicas pueden aplicarse tanto a archivos de audio digital, como los archivos de representación simbólica. Los segundos son los que nos interesan para este trabajo doctoral, en parte por que los primeros requieren un análisis de cómo digitalizar la percepción humana y esto sale del enfoque de nuestro trabajo. Para una muestra de lo anterior sugerimos el trabajo de Xiao y col. 2008.

La extracción de información armónica también ha sido un punto de interés entre varios investigadores. La idea consiste en tratar a una partitura como una base de datos donde cada elemento de la misma es una nota. El objetivo sería agruparlas de acuerdo a ciertas reglas para encontrar los acordes y armonías presentes. Dado que éste es un problema complejo, existen distintas formas de abordarlo, con diversos grados de éxito. Como ejemplo de esto, en el trabajo de Berman 2007, donde analizan el comportamiento armónico de distintas obras W.A. Mozart en formato *MIDI*. En nuestro trabajo, más adelante mostramos una propuesta para clasificar las notas de archivos musicales *MIDI* y *XML* por medio de lógica difusa y redes neuronales.

Además de los sistemas de recomendación, las búsquedas o *queries* son otra aplicación muy importante de las bases y minería de datos. Estas búsquedas representan un filtrado de la información o instancias de la base de datos en las que deseamos obtener una lista de los que cumplen los criterios de búsquedas. El sistema *MUSEMBLE*, desarrollado por Rho y col. 2008, es un ejemplo de este sistema de *queries* donde se

4 Extracción de información musical.

pueden buscar piezas musicales en una colección por medio de una melodía silbada o tarareada por el usuario.

Otra de las tendencias principales de la minería de datos aplicada a la música son los trabajos de clasificación de archivos de acuerdo a géneros, estilos y emociones. Como ejemplo tenemos el caso del trabajo de François Pachet, Westermann y Laigre 2001, donde utilizan la agrupación de archivos musicales con las técnicas de agrupación por *clusters*[5] para organizar listas de reproducción necesarias para la radio así como en los álbumes de CD.

La clasificación de emociones es un tema muy ambiguo y difícil de medir, sin embargo, es común encontrar trabajos que intentan abordarlo. Se han desarrollado algoritmos que buscan asociar los archivos musicales digitales, con las emociones que surgen en los usuarios. El concepto de clasificación emocional es bastante ingenuo y consiste en sencillamente registrar la emoción que el usuario reporta en ciertos fragmentos musicales para luego agruparlas. En el trabajo de Kuo y col. 2005, crean un sistema que pretende asociar ciertas armonías con las emociones que evoca en un grupo de individuos, para después extenderlo a un sistema de clasificación de archivos musicales basado en supuestas emociones. A pesar de la ambigüedad y subjetividad de sus hipótesis, reportan un 85 % de eficacia en sus pruebas de resultados.

En una línea similar a los anteriores, el trabajo de Kaminskas y Ricci 2012a, busca implementar un sistema de recomendaciones musicales basado en el contexto del usuario, desde sus emociones, sus tendencias y hábitos de uso digitales. Ellos buscan clasificar los datos de archivos musi-

5 Agrupación de elementos de acuerdo a características similares.

cales de forma que el usuario pueda encontrar los resultados fácilmente. Todos estos trabajos, nos sirven de contexto y referencia para conocer el tipo de aplicaciones que se le ha dado a la inteligencia artificial y a la minería de datos en la música.

INTELIGENCIA ARTIFICIAL

En esta investigación se utilizan algoritmos de Inteligencia Artificial para analizar y clasificar el contenido musical en partituras digitales. A continuación presentamos un breve recuento de los algoritmos más comunes utilizados en esta área y las implicaciones que tienen en el procesamiento de contenido complejo con la intención de establecer un lenguaje común, así como referencias adecuadas para el lector interesado en la parte más técnica.

La inteligencia artificial, es un concepto que involucra varias disciplinas y a grandes rasgos intenta emular la inteligencia y razonamiento humano por medio de máquinas. Una computadora de inicios del siglo XXI es capaz de realizar miles de millones de operaciones por segundo, éstas pueden ser: operaciones aritméticas, como sumas, multiplicaciones, operaciones lógicas y ejecución condicional de instrucciones. El hecho de que una computadora pueda realizar operaciones numéricas más rápido que un humano no implica que sea más inteligente. Para entender el problema que la IA aborda, es necesario que definamos lo que entenderemos por inteligencia.

Para que un sistema sea considerado inteligente es necesario que cumpla con algunas de las siguientes condiciones:

- El sistema debe ser capaz de tomar decisiones no triviales de acuerdo a los datos que tome como entrada.
- Debe ser capaz de aprender nuevas reglas y modificar sus algoritmos de toma de decisiones sin tener que programarlo explícitamente.
- Debe ser capaz de adaptarse a su entorno de modo que sus decisiones sean óptimas de acuerdo a una medición de costo elegida.

Actualmente los sistemas inteligentes que existen se diseñan para abordar un área y problema en específico. Más allá de la noción idealista de hacer que la computadora sea hecha a nuestra imagen y semejanza, son pocos los problemas inteligentes que las computadoras pueden abordar y resolver.

Los más comunes de enfrentar son: clasificación, regresión y toma de decisiones. Debido a que cada uno requiere técnicas diferentes. En nuestro trabajo utilizamos los sistemas expertos, lógica difusa y redes neuronales.

Los sistemas expertos consisten en un conjunto de reglas y condiciones lógicas booleanas que se codifican en una base de conocimientos. Éstas sirven como el fundamento lógico sobre el cual se construye una máquina de inferencia, la cual es responsable de tomar una decisión de acuerdo a los datos que se introduzcan. Este tipo de reglas las utilizamos en el capítulo 6 para identificar archivos musicales de acuerdo a las reglas del tratado de Gaspar Sanz.

En estos sistemas, para construir la base de conocimientos es necesario trabajar con un experto humano, especialista en el área del problema. Estos sistemas sirven para que la máquina sea capaz de tomar una decisión equivalente a la

que tomaría un profesional. Por ejemplo, se han programado sistemas expertos que sirven para hacer diagnósticos de enfermedades, el usuario introduce sus síntomas y el sistema hace una inferencia del diagnóstico. La idea básica detrás de estos algoritmos es seguir el flujo de un árbol de decisiones que te lleve a la inferencia más adecuada, podemos ver un ejemplo en la figura 1.

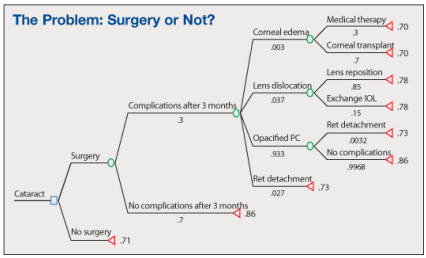

Figura 1: Árbol de decisión de condición médica, Management 2019.

En el trabajo de Hirata y Matsuda 2003 tenemos un ejemplo de análisis por medio de un sistema de reglas. Ellos analizan archivos MIDI, utiliza conceptos de la *Generative Theory of Tonal Music*[6] y los usa para buscar las frases musicales y encontrar similitudes entre fragmentos dentro del archivo digital.

Una rama de la inteligencia artificial que ha tenido mucho éxito recientemente es el aprendizaje automático, conocido en inglés como *Machine Learning*, la cual consiste en entrenar a una máquina a reconocer y clasificar objetos o instancias

6 Teoría generativa de la música tonal. Clasificación de las armonías y tonalidades desarrollada por Fred Lerdahl y Ray Jackendoff en 1983.

sin tener que programar explícitamente las reglas; es decir, a diferencia de técnicas como los sistemas expertos, donde necesitamos implementar exactamente cómo debe comportarse el sistema de cierta forma para cada caso, en el aprendizaje automático el sistema se entrena por medio de ejemplos, de modo que al estar expuesto a casos verdaderos y falsos, el sistema aprende a identificarlos de forma independiente. Para una referencia introductoria a ese tema, sugerimos consultar los trabajos de Tan, Steinbach, Kumar y col. 2006; Géron 2017.

Uno de los ejemplos canónicos del aprendizaje automático es el reconocimiento de letras en una escritura manual cursiva. Este problema requiere una programación que simule inteligencia, ya que a pesar de que cada carácter tiene una estructura particular, existen miles de variaciones en las inflexiones de las curvas y distribución del grosor del trazo en la escritura humana.

En los trabajos de Araokar 2005 y M. A. Nielsen 2011, se muestra cómo por medio de redes neuronales se puede entrenar a una computadora para que identifique las letras en una escritura cursiva de un humano. Es muy importante saber que el sistema necesita ser entrenado para poder distinguir cada carácter, sin importar las variantes que puede tener el trazo que cambia entre cada persona y cada vez que se escribe, ver figura 2. Una variante de este tipo de algoritmo es el que utilizaremos en el capítulo 3, para entrenar a la computadora a distinguir acordes, tonalidades y armonías, sin importar que se encuentren escondidas en el contenido melódico o rítmico.

En cuanto a las aplicaciones de aprendizaje automático a la música, el trabajo de Hartmann y col. 2007, crea un sistema

Figura 2: Clasificación MINST. M. A. Nielsen 2011.

que mezcla la minería de datos con el aprendizaje automático para analizar la obra completa de G.P. Teleman. Ésta utiliza una biblioteca digital de partituras de dicho compositor en formato *MusicXML* y aplican inteligencia artificial en el conjunto de datos para extraer información que sea útil de las piezas como tonalidad, modulaciones, entre otros. Este tipo de técnica lo utilizamos también en el capítulo 6 para analizar un corpus de piezas musicales.

Una muy buena referencia de aprendizaje automatizado se puede encontrar en Géron 2017, donde se encuentra una introducción a los conceptos de IA que aquí mencionamos.

Aprendizaje supervisado

Ahora vamos a introducir un concepto muy importante para el aprendizaje automatizado el cual nos permite abordar

distintos tipos de problemas de acuerdo a los datos con los que son entrenados. Los métodos aprendizaje automatizado se dividen principalmente en dos: supervisado y no supervisado. El primero consiste en mostrar al sistema una serie de ejemplos de datos con las etiquetas adecuadas. Por ejemplo, se entrena al sistema con una base de datos de los cuales se tiene su etiqueta de clasificación. Con ellos, se calibra el sistema adecuadamente para predecir la etiqueta de ejemplos nuevos. El objetivo del sistema es que aprenda a detectar correctamente la etiqueta de una nueva instancia, con base en los ejemplos con los que fue entrenado. En la parte izquierda de la figura 3, se muestra un ejemplo de puntos a clasificar donde se tiene información acerca de la etiqueta que corresponde a cada punto, en este caso círculos o cruces.

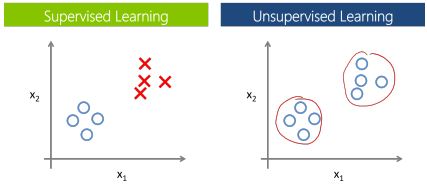

Figura 3: Aprendizaje supervisado y no supervisado. (Andrew NG, Coursera, Revisado 2018)

Para un ejemplo de aprendizaje supervisado, podemos ver en el trabajo de Dannenberg, Thom y Watson 1997, una aplicación para la identificación de lo que ellos llaman estilo musical. Buscan entrenar a un sistema que pueda detectar aspectos como la intencionalidad del músico o la emoción que expresa. De acuerdo a su terminología, buscan encontrar en una serie de notas *MIDI*, si el interprete tenía un estilo líri-

co, frenético, emocionado, asincopado o puntillístico, entre otros. Esto lo realizan entrenando un sistema con fragmentos musicales que ellos etiquetaron de antemano de acuerdo a los criterios de los investigadores. El sistema entonces, puede predecir la intencionalidad o emoción de ejemplos nuevos. Es muy importante notar que los resultados de este tipo de sistemas dependen en gran manera de los ejemplos con los que fue entrenado.

Aprendizaje no supervisado

Por otra parte, el aprendizaje no supervisado consiste en entrenar un sistema para aprender a distinguir agrupaciones de interés en una base de datos donde no se tiene información estructural de etiquetas.

Los problemas que aborda el aprendizaje no supervisado trabajan con datos que no tienen una etiqueta en sí mismos y es el algoritmo el que se encarga de agruparlos y asignarles una clase. Uno de los algoritmos más populares de esta área es el llamado *k-medias*, el cual toma como entrada una serie de datos y se elige el número de k grupos o clases en los que se desea agrupar. Como primer paso, se eligen k puntos aleatorios que representan una de las clases en las que se van a agrupar, a cada uno de los datos se les asigna el que tengan más cercano. Ya que se asociaron todos los puntos a una clase, se obtiene el promedio de cada una y se repite el proceso. Al final del proceso, se obtiene una clasificación de los puntos en el número de clases requeridas de acuerdo a la distribución de las instancias en el espacio sin tener que entrenar explícitamente al sistema. En la figura 4, se muestra una iteración del algoritmo para puntos en dos dimensiones.

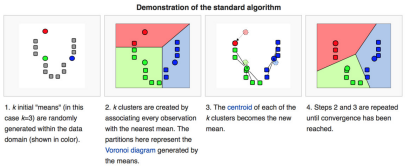

Figura 4: Ejemplo de algoritmo k-medias, Brilliant 2019.

El aprendizaje no supervisado se presta naturalmente para datos que no tienen etiquetas explícitas. La identificación de acordes en una pieza musical es un buen ejemplo, ya que a pesar de que las notas están ahí, no suele escribirse siempre explícitamente el acorde o armonía en un tiempo dado. En el trabajo de Pesek, Leonardis y Marolt 2014 se muestra un ejemplo de lo anterior, donde se utiliza un sistema de capas de información jerárquica para extraer información de acordes, entre otras.

La información que se clasifica con aprendizaje no supervisado puede ser de cualquier tipo. Por ejemplo, en el trabajo de Knopke 2004, se presenta una propuesta de clasificar archivos de audio de acuerdo a sus títulos o nombres y cómo se relacionan con su contenido musical. Esto es muy interesante por que abre las puertas a que podemos utilizar material extra musical para obtener mejores resultados. En nuestro trabajo, se utiliza una técnica similar para complementar el sistema de clasificación de archivos en el capítulo 6.

El aprendizaje automático, supervisado o no, involucra una serie de técnicas que se mencionan a lo largo de este trabajo. A continuación describimos las más importantes de forma conceptual y se incluyen referencias por si se desea ahondar

más en el tema. Comenzaremos por la *regresión*. Esta consiste en tomar una serie de datos y analizarlos para predecir cómo se comportarán ejemplos de datos nuevos que no se usaron en el entrenamiento. En el caso de la regresión. Esta predicción se hace con un modelo matemático que toma en cuenta cuanto contribuye cada uno de los datos al comportamiento general. De acuerdo al modelo matemático que se utilice, podemos hablar de regresión lineal y polinomial que sirven para predecir valores numéricos, o regresión logística, la cual nos sirve para clasificar en grupos similares.

Un ejemplo habitual de regresión logística es el de crear un sistema que decidiera si se aprueba o no un crédito a una persona. Para tomar esta decisión se deben tomar en cuenta diversos parámetros o atributos, por ejemplo: historial de crédito, ingresos anuales de la persona, edad, entre otros. El banco que implemente este sistema debe definir las condiciones que debe cubrir la persona para tomar la decisión crediticia. Para entrenar este sistema el banco puede tomar su base de datos históricos y revisar qué características tuvieron los sujetos que no cumplieron con sus obligaciones crediticias, en este caso, es un sistema con dos clases: aprobado o rechazado. De este modo, cuando el usuario introduzca sus atributos, el banco puede hacer un estimado de qué tan confiable es que esa persona cumpla con sus obligaciones. Podemos ver un ejemplo de regresión logística en dos dimensiones en la figura 5.

Las *redes neuronales* son otro método con el que podemos clasificar y predecir información. Actualmente es una técnica muy popular debido a la versatilidad de problemas que pueden abordar y resolver. A pesar de que teóricamente se

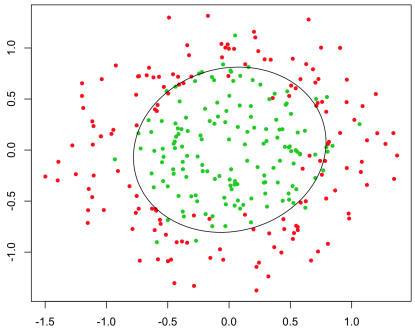

Figura 5: Regresión logística en dos dimensiones, Medium 2019.

desarrollaron hace varias décadas, su uso se popularizó en los últimos años debido a que los equipos de computo actuales comenzaron a ser lo suficientemente poderosos para que los resultados sean útiles.

El nombre de redes implica que hay una serie de objetos conectados. En este caso, una red neuronal se compone de capas donde cada una de ellas contienen un nodo. Cada nodo realiza una operación matemática sencilla, que cuando se encadenan entre sí, son capaces de realizar predicciones muy acertadas y eficientes si se entrenan correctamente.

Cada nodo de la red tiene coeficientes numéricos que se ajustan de acuerdo al problema a resolver. Entrenar a una red neuronal, es el proceso en el que introducimos los datos ejemplos y de acuerdo a técnicas como la *propagación hacia atrás*, los coeficientes se calculan automáticamente.

Una red neuronal puede pensarse como una serie de regresiones logísticas encadenadas entre sí, a las cuales se les introduce una función no linéal entre ellas. En la figura 6,

podemos ver un diagrama de cómo se organizan las capas de nodos en una red neuronal. Su nombre se presta a ciertas interpretaciones que suelen ser incorrectas. El término redes neuronales parece implicar que los nodos en los cálculos se comportan como las neuronas cerebrales y que estos sistemas enfrentan los problemas de la misma manera que nuestro cerebro.

En realidad, no existe un consenso acerca de cómo funciona la mente humana, por lo que el nombre es una mera analogía de cómo se cree que el cerebro funciona, pero nada más. Las redes neuronales son muy útiles en problemas de identificación y clasificación.

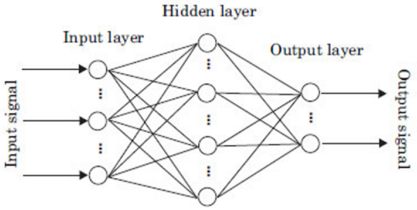

Figure 1. Structured chart of neural network.

Figura 6: Diagrama de las capas en una red neuronal, M. A. Nielsen 2015.

Una variante de las redes neuronales es el área conocida como aprendizaje profundo o *deep learning*. Esta área consiste en un caso particular de las redes neuronales cuando éstas tienen una gran cantidad de nodos y capas. En sí, el aprendizaje profundo no tiene definiciones claras acerca de cuántos nodos o capas se necesitan. Es un término que indica que el problema y los datos a analizar requieren una capacidad de cómputo mayor, por ejemplo, se puede usar un

sistema profundo para clasificar qué objetos están presentes en una imagen, como personas, carros, gatos, etc. En el caso de problemas musicales, se puede utilizar para identificación de contenido armónico.

Podemos ver un ejemplo de aplicación de las redes neuronales para la clasificación armónica en el trabajo de Hörnel 2004. Utilizan una red junto con programación dinámica para generar conducciones de voces optimizadas de acuerdo a las restricciones de las reglas del contrapunto. Lo que ellos proponen es crear un sistema flexible que más allá de seguir las reglas, pueda adaptarse a casos complejos donde no existe una solución única. En nuestro trabajo utilizamos las redes neuronales para ayudar con la clasificación de acordes, donde nuestra aportación consiste en la forma en que se traduce la partitura digital como entrada de datos para la red neuronal.

Las técnicas que hemos mencionado, las hemos tratado en general. Sin embargo, una de las razones por la que han tenido mucho éxito actualmente es debido a que hay implementaciones bien documentadas y eficientes en los lenguajes de programación más populares de la actualidad. En este trabajo, el motor de búsqueda está implementado en *Python*, el cual puede utilizar bibliotecas de código que cuentan con versiones muy eficientes de estos algoritmos. Una de las más populares actualmente es *TensorFlow*, desarrollada por Google la cual es de código abierto. Ésta la usaremos en algunos de los módulos que se presentan en los siguientes capítulos.

Uno de los retos más difíciles en el aprendizaje automático es que se debe entrenar para que sea capaz de clasificar correctamente las instancias de las cuales ya se sabe su clase, y así mismo, las que son nuevas. Cuando un sistema se sobre

entrena, puede pasar el caso que éste aprenda tan bien los ejemplos, que sea incapaz de reconocer casos nuevos. Por otra parte, si se subentrena el sistema, las clasificaciones pueden ser burdas y triviales. En las figuras 7 y 8 se muestra un ejemplo de *Over* y uno de *Under fitting* en una regresión polinomial.

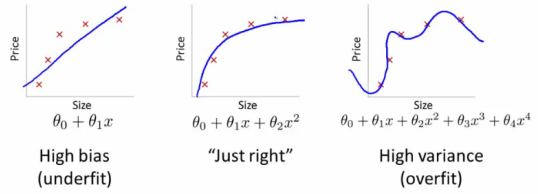

Figura 7: Ejemplo de sobre ajuste, Ng 2015

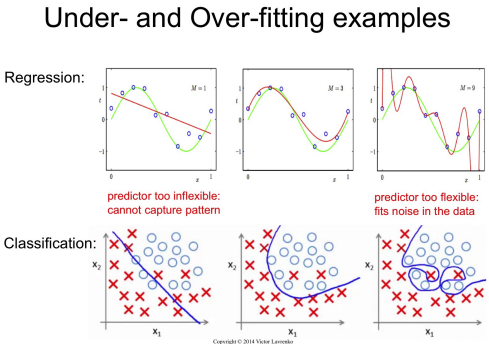

Figura 8: Ejemplo de sobre ajuste, Lavrenko 2014.

Entre las distintas áreas que engloban a la Inteligencia Artificial, la *Lógica difusa* juega un papel importante. Esta

área a diferencia de la lógica clásica, donde las cosas son verdaderas o falsas, consiste en establecer grados de pertenencia para cosas donde la clasificación no es blanco y negro. Un ejemplo clásico de lógica difusa es el clasificar si una temperatura es caliente o fría, dado que estos son conceptos ambiguos y personales, no existe una definición precisa de cuándo empieza uno y termina el otro. Para abordar este tipo de problemas, las clasificaciones por lógica difusa utilizan funciones de pertenencia, que indican un grado de pertenencia a una clase o a alguna otra. Esto se verá más a detalle en el módulo de clasificación y etiquetado de acordes.

La aplicación de técnicas computacionales para el estudio de la música tiene ciertas críticas, en Byrd y Crawford 2002, se menciona que la mayoría de los algoritmos desarrollados utilizan principalmente la información de altura de la nota para la búsqueda y clasificación, sin embargo sugiere que esto no es suficiente para un estudio más profundo. Por otra parte Schubert y Stevens 2006, cuestionan nuestro entendimiento de la percepción de los cambios armónicos, mencionan que el entrenamiento juega un papel fundamental en cómo percibimos la música, de modo que esto se debe tomar en cuenta al programar sistemas que emulen nuestra percepción. Por último, Sturm 2014, hace una revisión de quinientos artículos de identificación, de lo que ellos, llaman géneros musicales automático y hace énfasis que aún falta mucho trabajo por realizar para que estos sistemas de búsqueda y clasificación sean útiles en la vida cotidiana.

Por nuestra parte, el estudio que proponemos involucra varias áreas del conocimiento, lo cual implica una intersección de conceptos y teorías que se complementan entre sí. Por un

lado tenemos el análisis musical, la parte artística y del área de humanidades, mientras que por otro, los procesos computacionales y tecnológicos ofrecen un contrapeso de ideas y técnicas. Es justo en esta intersección que se encuentra el nicho de oportunidad para este trabajo de doctorado. Debido a que cada una de las áreas son maduras en sí mismas, existe actualmente un vacío de herramientas y técnicas que permitan a un músico agregarlas a su uso cotidiano. Por medio de nuestro sistema modular, abrimos una oportunidad de trabajo en conjunto tanto para investigadores en la computación e investigadores de la música. Es la intención que este sistema sea una primera versión que pueda crecer orgánicamente con los nuevos avances conceptuales y tecnológicos.

Este traslape puede considerarse en el terreno de lo que algunos autores como Kirschenbaum 2014 y Abdallah y col. 2017, han llamado *Digital Humanities* (Humanidades Digitales). Como lo mencionan en su trabajo, este término se ha acuñado gradualmente dado el inevitable permeo tecnológico que todas las áreas académicas han experimentado en los últimos años. Aún no está claramente definido, pero consiste en el trabajo inter y transdisciplinario que surge al mezclar proyectos de humanidades con proyectos tecnológicos. En el presente trabajo doctoral nos inscribimos a este concepto de las humanidades digitales, por medio de las técnicas que proponemos en los siguientes capítulos. Nos interesa enfatizar, que más que una herramienta tecnológica, el progreso más importante es iniciar una plataforma de trabajo en conjunto entre las humanidades y las técnicas computacionales para el análisis de la música, comenzando aquí en la UNAM, y que sirva de ejemplo para otros centros de investigación.

Se necesita el trabajo en conjunto, ya que las herramientas computacionales pueden proponer una síntesis de análisis, pero al final del proceso, es el humano, el musicólogo quién decide cómo es la mejor forma de utilizarse.

> *It is the job of the analyst to interpret these observations, and the functions and relationships between them, as well as to abstract and synthesize the information. The decomposition may be done by the computer and the software, but the recomposition must be done by the researcher.*[7] Lande y Vollsnes 1994.

Ellos describen que el problema no es sólo la representación musical de los datos, *MIDI* en su caso. La cuestión es cómo acceder a estos datos de una forma útil para la musicología. Éste es un problema que es vigente hasta nuestros días.

Para lograr una aplicación tecnológica que sea eficaz, es muy importante escuchar las necesidades del usuario en quién el programa está pensado. Es muy común que el diseño de software siga las tendencias tecnológicas de moda, sin embargo, esto no significa que eso sea lo que el usuario realmente necesita. Un ejemplo de esto lo podemos ver en la discusión de McKay 2010, donde explica que existe una tendencia a clasificar las piezas de acuerdo a recomendaciones por lo que ciertos algoritmos dicen que son obras similares, tanto

7 Es el trabajo del analista interpretar estas observaciones, así como las funciones y relaciones entre ellas, también abstraer y sintetizar la información. La descomposición puede ser hecha por una computadora y el software, pero la recomposición debe ser hecha por el investigador. Traducción propia.

en forma armónica, como rítmica, sin embargo, los usuarios han mostrado que prefieren buscar por lo que ellos géneros o estilo, es decir, según ellos, un usuario que busca piezas musicales que se puedan agrupar en clases que él ya conoce, lo cual suele ser ignorado por los desarrolladores. McKay, sugiere que los desarrolladores deben tomar en cuenta las necesidades reales de los usuarios y no forzarlos a usar paradigmas que no son naturales para ellos simplemente por que es más fácil programarlos.

Uno de los problemas más grandes de la minería de datos e inteligencia artificial es que muchas veces los experimentos no son fáciles de reproducir, así que suele pasar que no hay forma de medir si el método propuesto es mejor o no a los anteriores. El trabajo de McKay 2010, presenta algunas de las desventajas de la falta de estandarización en el desarrollo de algoritmos y programas de cómputo. Se necesitan *benchmarks*, es decir formas de pruebas estandarizadas para poder comparar distintos programas para esto, no es necesario que todos usen el mismo software.

Otra forma de hacer la interacción del músico con los resultados computacionales, es por medio de visualizaciones gráficas de los resultados. Por esto, así como existen muchos trabajos acerca de clasificación y análisis de archivos musicales, necesitamos formas de visualizar la información procesada para mostrar al usuario los resultados de una forma que sea fácil de entender sin entrar en tecnicismos. En el trabajo de F. Nielsen y Nock 2009 se ofrecen diversas técnicas muy útiles para la representación visual de datos de diversos tipos y sus resultados se prestan muy bien para la minería de datos.

Con esto terminamos la sección de introducción de términos comunes y una muestra del estado de la cuestión de este tema de investigación. Tomando en cuenta todos estos avances y desarrollos, podemos pasar ahora la forma en que en este trabajo de doctorado se abordará el problema de la extracción de información armónica, melódica, estructural y de reconocimiento de estructura. En los próximos capítulos se muestran el sistema modular de aplicaciones de la inteligencia artificial y minería de datos al análisis musical. Se explica también cómo éste es expandible para que sea una técnica más que los investigadores de la música puedan agregar a su caja de herramientas de análisis musical.

2

BASES DE DATOS DE ARCHIVOS MUSICALES

En este capítulo se presentan las implicaciones de representar el contenido de partituras en formatos digitales, así como distintas situaciones a considerar para el correcto pre procesamiento de los datos que se utilizarán para los algoritmos. Esta parte es fundamental para el trabajo, ya que la calidad de los resultados del análisis está directamente relacionada con la forma en que los datos de entrada están representados.

Una partitura representa una abstracción del contenido musical de una pieza. Ésta cumple al menos dos funciones: la primera, proveer al músico de una serie de instrucciones acerca de cómo se debe interpretar la pieza; la segunda, sirve como un registro de instrucciones para el interprete, esto puede ser en forma de indicaciones de notas por tocar, así como dinámicas y matices de sonido que debe interpretar. Para los fines de este trabajo, la segunda parece ser más apropiada ya que de estas instrucciones podemos extraer información que es útil para analizar la estructura de una obra. De este modo, la partitura, más que un registro de instrucciones, nos sirve como una fuente de información

estructural que se presta para un estudio y análisis de sus componentes.

En el ámbito del análisis digital, la partitura se convierte en la fuente con la que creamos bases de datos que nos permiten utilizar algoritmos altamente eficientes para la búsqueda y clasificación de contenido. La forma en que se traduce una partitura en papel a un formato digital ha sido abordada por distintos equipos de desarrollo. Cada uno utiliza la representación digital que sea más conveniente para su estudio.

Como se mencionó anteriormente, en este trabajo, analizaremos archivos de representación digital de la música. Recordemos, que estos archivos no contienen audio, sino una serie de instrucciones acerca de las notas, tiempos y duraciones en las que deben ser interpretadas.

Uno de los formatos de archivos más populares de representación simbólica es el formato *MIDI*. Consiste en una serie de mensajes binarios con instrucciones temporales de inicio de nota y fin de nota. Este formato no puede leerse directamente por un humano ya que necesita ser traducido primero. Es muy popular en los repositorios en línea y funciona tanto para grabaciones de interpretes en vivo como partituras generadas en editores como Sibelius, Finale o MuseScore. Una de las limitaciones más grandes de este formato es que no proporciona información de enarmónicos, compases ni valores rítmicos. En términos coloquiales, es una lista de números que indican la nota y la posición en el tiempo en que debe iniciar y cuando debe parar.

En contraste con el formato *MIDI*, tenemos al formato *MusicXML*, el cual es una versión del formato *XML* adaptado al contexto musical. Consta de una serie de etiquetas anidadas

que agrupan los elementos. Es un formato muy práctico para trabajar en forma digital ya que existen numerosas herramientas computacionales para procesarlos. No es amigable de leer para un humano, pero esto no es un impedimento ya que todo el procesamiento se hace internamente y al usuario sólo se le presenta la partitura resultante. Este es el mejor formato actualmente para la representación de partituras digitales, ya que a diferencia de *MIDI*, cuenta detalladamente con información de enarmónicos. Por ejemplo, en *MIDI* las notas se cuentan por semitonos, donde 60 representa Do central. Esto significa que para representar a Do sostenido o Re bemol usamos el número 61 y no hay forma de especificar a cuál de los dos nos referimos. En cambio, en el formato *MusicXML* podemos indicar explícitamente si deseamos Do sostenido o Re bemol, con sus respectivos nombres. Esto es muy útil para el análisis, ya que es importante saber cómo una nota está escrita. De igual forma, este formato incluye información más detallada de las duraciones, compases y hasta información gráfica como márgenes y dirección de las plicas de las notas.

Los programas de edición tienen sus propios formatos no intercambiables (*sibelius*, *finale*, etc.). Como son formatos que no permiten compartir entre distintas aplicaciones, no nos interesan para el análisis. Cada una de estas plataformas tiene la funcionalidad de exportar archivos *MusicXML*, de modo que podemos utilizarlos para nuestro análisis. Existen muchos otros formatos como: *MEI, ABC, Humdrum, TinyNotation, lilypond*. Estos funcionan de forma similar a los anteriores, pero no son muy populares actualmente. Para los fines de este trabajo, utilizamos exclusivamente *MusicXML*.

A pesar de la diferencia entre formatos, todos tienen una estructura similar, donde agrupan las notas, acordes, compases, dinámicas, etc, como elementos independientes que se relacionan entre sí. Esta estructura se presta para una representación en lenguaje de programación orientado a objetos, la cual es muy útil para organizar el contenido de la partitura en forma digital.

Muchas de las partituras digitales que encontramos en la red están en formato *PDF*, que es un formato de imágen, los cuales no se consideran archivos de representación simbólica. Esto se debe a que no podemos extraer directamente la información de notas y duraciones como lo haríamos con *MIDI* y *XML*. Sin embargo, podemos utilizar programas de reconocimiento óptico de música como *Audiveris* para convertirlos a *XML*. Este proceso puede ser tardado ya que hay que verificar que la digitalización se realice correctamente de acuerdo al archivo original.

Así como es posible convertir archivos de imágen como *PDF* a *MIDI* o *XML*, es común que se desee convertir un archivo de audio directamente a partitura. Esto es mucho más difícil actualmente, especialmente para material polifónico y con muchos instrumentos. La conversión de archivos de audio a archivos de representación simbólica de la música es un área de investigación que merece un estudio profundo y sale de nuestro enfoque. Sin embargo, cuando los resultados de dichas conversiones sean lo suficientemente buenos en un futuro será muy fácil integrarlos al análisis modular que presentamos en este trabajo.

El sistema que desarrollamos en este trabajo consiste en varios módulos con responsabilidades exclusivas a cada uno.

El motor de análisis está desarrollado en el lenguaje de programación *Python*, el cual es muy eficiente para el análisis de datos. En este lenguaje, la biblioteca de programación *music21* cuenta con funciones adecuadas para organizar y procesar el contenido musical de cada uno de los formatos de archivos mencionados y muchos mas. *Music21* funciona como un traductor del formato de partitura a objetos digitales que pueden analizarse. Una vez que se encuentran en esta representación digital, se pueden utilizar algoritmos de análisis de datos para extraer información útil. Como ejemplo de una biblioteca de código para usar técnicas de *Machine Learning* por medio de *music21*, tenemos el trabajo de Antila y Cumming 2014, en el cual analizan el estilo de contrapuntos.

En nuestro trabajo, se realiza una metodología similar, donde la biblioteca *music21* se utiliza como interfaz entre las partituras y los algoritmos que aquí se desarrollan e implementan.

REPRESENTACIÓN DIGITAL DE UN COMPÁS

Como primer paso del pre procesamiento de datos necesario para el análisis digital se muestra la forma en que se representan diferentes objetos musicales desde el punto de vista numérico y digital. Comenzaremos por un compás, el cual puede analizarse desde distintos ángulos de acuerdo a la estructura que se requiera: melódica, armónica, rítmica o tímbrica.

Una forma interesante de representar de forma digital el contenido de un fragmento musical es la propuesta por el análisis paradigmático mostrado en el trabajo de Anagnosto-

poulou y Westermann 1997, se puede realizar un análisis de las estructuras que aparecen en los compases de un fragmento musical. Ellos sugieren clasificar los siguientes espacios de descriptores (*feature spaces*): contorno melódico, movimiento rítmico, patrones de intervalos, registro de instrumentos, los cuales se consideran suficientes para estas clasificaciones.

Como ejemplo del tipo de represenaciones numéricas del análisis paradigmatico, a continuación se muestran unos resultados obtenidos en este trabajo para extraer de forma digital las estructuras melódicas y rítmicas de una pieza musical. En la siguiente tabla se presenta un ejemplo de los datos de estructura melódica y rítmica que pueden surgir al analizar el Minuet en Do Mayor de la Suite española de Santiago de Murcia.

Vectores de melodías posibles en un compás:

- (C, E, C)
- (G, C, E, G, E, C)
- (G, E, C, C, E, G)
- (E, C, B, A, G, F, G, C)
- (E, G, C, E, G, C, E, G, C)
- (C, B, C, D, E, F, C, E, G)
- (C, F, A, C, F, A, G, G, B)
- (E, A, F, G, F, E, D, G, C, G)
- (C, C, E, G, C, E, G, B, C, E, G, C, C, E, G, D, C, E, G, E, C, E, G, F)

Vectores de ritmos posibles en un compás:

- (1.0)
- (3.0)
- (1.0, 1.0, 1.0)
- (1.0, 0.5, 0.5, 0.5, 0.5)
- (3.0, 2.0, 1.0, 3.0, 2.0, 1.0)
- (3.0, 1.0, 1.0, 1.0, 3.0, 3.0)
- (3.0, 3.0, 3.0, 1.0, 1.0, 1.0)
- (3.0, 1.5, 0.5, 0.5, 0.5, 3.0, 3.0)
- (3.0, 1.0, 1.0, 0.5, 0.5, 3.0, 3.0)
- (3.0, 2.0, 1.0, 3.0, 1.0, 1.0, 1.0)

Como se aprecia, a pesar de que una pieza tenga muchos compases, las estructuras rítmicas y melódicas pueden resultar en una lista pequeña de representantes que se repiten y combinan. Estos patrones nos servirán para realizar clasificación de archivos más adelante, ya que de acuerdo al número de ellos que aparezcan, pueden dar información de la complejidad de la pieza. Este tipo de síntesis de información es múy útil para clasificar y encontrar patrones en la información musical. En el siguiente fragmento se muestra el código utilizado para encontrar los descriptores del análisis paradigmático.

```
1     def __calc_paradigms(self,music_stream):
2         list_paradigms_pitch = []
3         list_paradigms_durations = []
4         lastTimeSignature = []
5         list_paradigms_pclass = []
6
7         c = []
8
9         for m in music_stream.recurse().
              getElementsByClass('Measure'):
10            durations_in_measure = []
11            durmes = []
12            pitches_in_measure = []
13
14            for r in m.recurse():
15                if not type(r) is stream.Measure and r.
                      duration.quarterLength != 0:
16                    durations_in_measure.append(float(r.
                          duration.quarterLength))
17                    durmes.append(r.duration)
18                if  type(r) is chord.Chord:
19                    for c in r.pitches:
20                        pitches_in_measure.append(c.name
                              )
21                if  type(r) is note.Note:
22                    pitches_in_measure.append(r.name)
23
24            if len(durations_in_measure) < 12:
25                list_paradigms_durations.append(tuple(
                      durations_in_measure))
26
27            if len(list_paradigms_pitch) < 12 and len(
                  pitches_in_measure)>0:
28                crd = chord.Chord(pitches_in_measure)
29                list_paradigms_pitch.append(tuple(
                      pitches_in_measure))
```

```
30                  list_paradigms_pclass.append(tuple(crd.
                        normalOrder))
31
32          list_paradigms_pclass=list(set(
                list_paradigms_pclass))
33          list_paradigms_pclass.sort(key=len,reverse=False
                )
34
35          list_paradigms_pitch=list(set(
                list_paradigms_pitch))
36          list_paradigms_pitch.sort(key=len,reverse=False)
37
38          list_paradigms_durations=list(set(
                list_paradigms_durations))
39          list_paradigms_durations.sort(key=len,reverse=
                False)
40
41          return list_paradigms_pitch,
                list_paradigms_durations,
                list_paradigms_pclass
```

Código 2.1: Cálculo de paradigmas.

BASE DE DATOS DE NOTAS

La motivación más importante de representar las partituras en formato digital, es utilizar las técnicas de análisis de bases de datos para extraer información interesante. Una de las funciones más importantes en este contexto es la realización de búsquedas específicas, llamadas *queries*. El objetivo es buscar entre todas las instancias, o elementos, de una base de datos, aquellas que cumplen con unas condiciones específicas, requeridas por el *query*.

Por ejemplo, en una base de datos de personal académico de una universidad, se podría requerir una lista de todos aquellos mayores de treinta años que impartan cuatro horas de clases a la semana. El sistema entonces utiliza algoritmos especializados para obtener una lista con los elementos que satisfagan las propiedades. En un contexto musical, sería de interés hacer búsquedas que ayuden a responder preguntas musicológicas interesantes. Es necesario plantear correctamente las limitaciones y alcances de este tipo de búsquedas, a continuación muestro algunos ejemplos de *queries* sencillos:

- Buscar todos los acordes de séptima disminuida en una pieza.
- Buscar todos los diferentes patrones rítmicos.
- Buscar todas las melodías que cumplan con la línea "Do Sol Re La", o cualquier otro patrón.
- Buscar todos los compases donde aparece un *Pitch Class Set* en específico.
- Buscar los patrones rítmicos más utilizados en una pieza musical.
- Dado un corpus de música, buscar si hay piezas musicales que compartan un cierto motivo melódico o armónico.
- Buscar todos los lugares donde aparece una cierta progresión armónica.

Las búsquedas pueden ser de muchos y diversos tipos. Para que puedan ser útiles en la resolución de problemas musicológicos, probablemente serán mucho más complejas que simplemente contar acordes. Sin embargo, es un buen punto de partida para mostrar el tipo de *queries* que se pueden realizar. Por esta razón, existe la necesidad de establecer búsquedas eficientes. Esta problemática la abordan Corrêa y Rodrigues 2016, donde enfatizan la falta de bases de datos de archivos de representación simbólica, como *MIDI* y *MusicXML*, estructuradas para realizar análisis y proponen una lista con las más utilizadas actualmente. Ellos realizan una revisión de los vectores de descriptores más usuales para clasificar lo que ellos llaman género musical, los cuales se presentan en la figura 9. Estos descriptores los tomamos como base para los algoritmos desarrollados e implementados en este trabajo.

Debido a que no existe una base de datos global para la música, diversos autores han tratado de hacer un compendio de dónde se puede hacer referencia a distintos repositorios en dónde encontrar este tipo de archivos. En el trabajo de Viro 2011, se presenta un intento de reunir las diferentes bases de datos de partituras digitales en una sóla referencia de índices. Mencionan la librería Petruci (*International Music Score Library Project* s.f.) y *Kunst der fuge* (*Kunst der fuge* s.f.), las cuales son unas de las más grandes disponibles. Utilizan una conversión automática de *PDF* a *XML* usando lo que ellos recomiendan como el mejor sistema al momento llamado *Smartscore*.

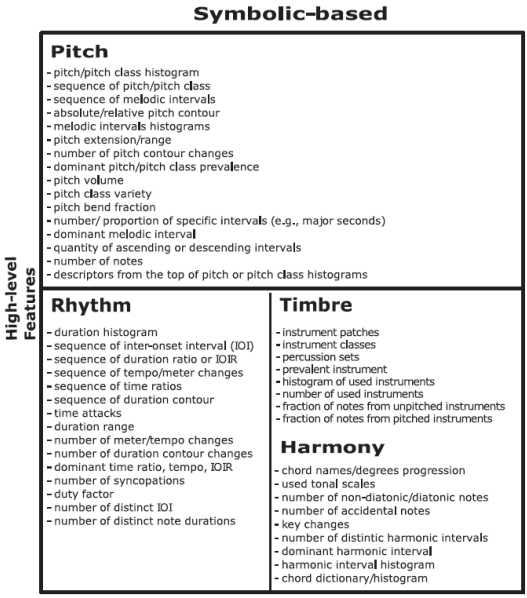

Figura 9: Tipos de descriptores, Corrêa y Rodrigues 2016.

Descriptores estadísticos

Como parte de los trabajos de clasificación de las bases de datos digitales, es útil tener información detallada de las piezas que servirán para el análisis. Para clasificarlas se necesita extraer información descriptiva de distintos parámetros que representan características específicas, a ellos se le llaman descriptores[1]. Estas representaciones estadísticas sirven para clasificar y encontrar información de interés en el análisis de piezas, como lo hacen los autores Beran, Mazzola y col.

1 Se pueden llamar también como características, o en inglés se conocen como *features*

1999, donde utilizan dichos descriptores para el análisis de estructuras métricas, melódicas y armónicas.

En la literatura existen programas que extraen estos descriptores para ayudar a los investigadores a realizar sus búsquedas. La investigación de Lamm 2015, presenta un sistema web que extrae algunas de estas características estadísticas de archivos *MusicXML*. El programa se llama *MusicXML analyzer*, nos sirve como referencia para buscar los descriptores que necesitamos por medio de un sistema web. Su sistema cuenta con solo algunos descriptores básicos, es nuestra intención crear un sistema más robusto con nuevos algoritmos y que por medio del sistema modular que proponemos, se puedan agregar nuevos descriptores de forma gradual, nutrido por el trabajo de muchos investigadores.

En el siguiente fragmento de código se muestra un ejemplo de los cálculos y llamados de funciones utilizados para extraer los descriptores armónicos. El resto del código puede verse en el apéndice digital.

```
1  class Harmonic_Features:
2
3      def __init__(self,dict_lists):
4          list_chords_obj = dict_lists['list_chords_obj']
5          list_chord_roman_obj = dict_lists['
               list_chord_roman_obj']
6          list_chromatic_notes_obj = dict_lists['
               list_chromatic_notes_obj']
7          list_notes_obj = dict_lists['list_notes_obj']
8          list_keys = dict_lists['list_keys']
9          list_diatonic_notes = dict_lists['
               list_diatonic_notes']
10         stream_chordify =  dict_lists['stream_chordify']
11
```

listDiatonicNotes,	music_stream,
list_chord_roman_obj,	number_distinct_chords,
list_chord_str,	number_distinct_durations,
list_chords_obj,	number_distinct_ioi,
list_chromatic_notes_obj,	number_key_changes,
list_contours,	number_measures,
list_durations_int,	number_mel_changes,
list_durations_notes_str,	number_notes,
list_durations_rests_str,	number_rests,
list_durations_str,	number_tempo_changes,
list_instruments,	number_time_ratio_changes,
list_intervals_str,	perc_diatonic_notes,
list_ioi,	perc_non_diatonic_notes,
list_keys,	pitchVolume,
list_measures_obj,	pitch_class_vector,
list_mel_changes,	range_durations,
list_midi,	range_pitch,
list_notes_obj,	stats_intervals,
list_notes_str,	stats_notes]
list_offset_notes_int,	dominant_chord,
list_pitch_classes_int,	dominant_pitch,
list_rest_obj,	dominant_pitch_class,
list_roman_str,	histogram_chords_str,
list_tempos,	histogram_degree_str,
list_time_ratios_str,	histogram_durations_str,
	histogram_notes_str,
	histogram_pitch_class_notes,
	histogram_tempos,
	histogram_time_ratios,
	histrogram_mel_intervals,
	histrogram_midi,

Cuadro 1: Descriptores utilizados.

```
12          self.list_chord_str = [c.pitchedCommonName for c
                 in list_chords_obj]
13          self.list_roman_str =[d.figure for d in
                list_chord_roman_obj]
14          self.perc_non_diatonic_notes = len(
                list_chromatic_notes_obj)/len(list_notes_obj
                )
15          self.list_keys = list_keys
16          self.histogram_degree_str = self.
                __calc_histogram_roman_str()
17          self.histogram_chords_str = self.
                __calc_histogram_chords()
18          self.perc_diatonic_notes = len(
                list_diatonic_notes)/len(list_notes_obj)
19          self.number_key_changes = len(self.list_keys)
20          self.number_distinct_chords =len(self.
                histogram_chords_str)
21          self.string_chords = self.__calc_list_chordify(
                stream_chordify)
22
23      def __calc_histogram_roman_str(self):
24          degreeCollect=collections.Counter(self.
                list_roman_str)
25          histogram_degree_str =list(degreeCollect.items()
                )
26          histogram_degree_str = sorted(
                histogram_degree_str, key=lambda tup: tup
                [1],reverse=True)
27          return histogram_degree_str
28
29      def __calc_histogram_chords(self):
30          degreeCollect=collections.Counter(self.
                list_chord_str)
31          histogram_chords_str =list(degreeCollect.items()
                )
```

```
32          histogram_chords_str = sorted(
                histogram_chords_str, key=lambda tup: tup
                [1],reverse=True)
33          return histogram_chords_str
34
35      def __simpleName(self,name):
36          ret = name
37          if name=="major":
38              ret ="M"
39          elif name=="minor":
40              ret ="m"
41          return ret
42
43      def __calc_list_chordify(self,stream_chordify):
44          string_chords = ''
45          for m in stream_chordify.recurse().
                getElementsByClass('Measure'):
46              for c in m.recurse().getElementsByClass('
                    Chord'):
47                  qual =''
48                  if c.quality != 'other':
49                      qual=c.quality
50                      string_chords+=' '+c.root().name+''+
                            self.__simpleName(qual)
51              string_chords+=' |\n'
52          return string_chords
```

Código 2.2: Cálculo de paradigmas (b).

Ya que tenemos un formato para los archivos y la lista de descriptores sobre la cual realizaremos el análisis y extracción de información, es importante considerar las tecnologías de bases de datos que tenemos accesibles para organizar nuestra información. Dos de las más populares actualmente son *SQL* y *MongoDB*. Ambas consisten en un conjunto de métodos y estrategias para guardar y recuperar la información de la base

de datos. En este trabajo utilizamos la segunda debido a que es muy práctica y fácil de montar. En los próximos capítulos se muestra cómo se implementa en este proyecto.

En este capítulo se mostraron las implicaciones de guardar la información musical en formatos de representación simbólica como *MIDI* y *MusicXML*. Se presentaron las herramientas computacionales con las que procesamos estos datos en el contexto del sistema modular de este trabajo doctoral. Se presentó el concepto de descriptores y cuales utilizamos nosotros para la clasificación de material musical. En los siguientes capítulos mostraremos los módulos de análisis que se basan en las representaciones aquí mostradas.

3

RECONOCIMIENTO AUTOMÁTICO DE ACORDES

En este capítulo se describe el módulo de análisis de acordes de nuestro sistema de análisis. Es muy importante, ya que el análisis de armonías y acordes sirve como base para otros módulos, como el de búsqueda de patrones y similitudes. El objetivo de este módulo consiste en entrenar a un sistema para el reconocimiento automático de acordes en archivos musicales de representación simbólica, es decir *MIDI* y *MusicXML*. Esto se logra por medio de un sistema automatizado de etiquetas basado en técnicas de inteligencia artificial. Se utilizaron dos tipos de algoritmos: clasificación con lógica difusa y redes neuronales. Con ellos se obtuvo un sistema robusto para identificar, clasificar y etiquetar acordes. Al finalizar se exponen los principales beneficios y limitaciones de utilizar estos métodos en el estudio de la música.

En el capítulo anterior se presentaron algunas de las opciones para representar el contenido musical de una partitura en forma digital, de modo que se puedan aplicar algoritmos de inteligencia artificial y minería de datos para segmentar y clasificar conjuntos de notas en los acordes correspondientes.

A continuación se describen los dos métodos que se aplicaron en este capítulo. El primero consiste en utilizar lógica difusa por medio de funciones de pertenencia, para buscar segmentaciones óptimas. El segundo utiliza un sistema de redes neuronales para clasificar acordes. Ambas técnicas tienen la ventaja de que pueden colocar etiquetas correctas incluso en situaciones con ruido, en este caso, secciones donde los acordes no se encuentren presentados con notas no armónicas.

Como se mencionó anteriormente, se propone que los algoritmos trabajen sobre archivos en el formato *MusicXML*, ya que contiene todos los datos de una partitura organizados en una forma que es fácil de leer para un programa de computadora, actualmente es la mejor forma de representar contenido de partituras en forma digital, esto se describe a detalle en el trabajo de Cunningham 2004.

Existen diversos enfoques y algoritmos para la extracción de contenido armónico de una pieza musical. Como se mencionó en el capítulo pasado, una de las herramientas más poderosas para trabajar con partituras digitales desde el punto de vista de extracción de información es la librería del lenguaje de programación python *music21*, Cuthbert y Ariza 2010, ya que permite procesar partituras en distintos formatos, entre ellos: xml, lilypond, *MIDI*, entre muchos otros. Existen otras alternativas, como el *MIDI* toolbox de Matlab, Eerola y Toiviainen 2004, y el sistema *Melisma* desarrollado por Temperley 1999, pero a nuestro parecer, *music21* es el más robusto y flexible.

En cuanto a las técnicas utilizadas para la extracción de acordes, hay varias metodologías, presentamos las más po-

pulares a continuación. Algunos autores utilizan redes neuronales para clasificación, como es el caso de Lin y Peng 2011, que utiliza una técnica conocida como *Particle Swarm Optimization* en canciones de tipo baladas para niños con una métrica de $4/4$, donde obtiene resultados interesantes, pero en un contexto muy limitado. Por otra parte, en el trabajo de Perera y Kodithuwakku 2005, utilizan una red neuronal de una capa para analizar acordes de bloque, es decir sólo acordes donde las notas aparecen simultáneamente, esto limita su uso a música de tipo similar a corales.

Gran parte de los trabajos mencionados asumen una segmentación a priori realizada en la partitura para realizar la extracción de información armónica, lo cual limita el caso de aplicación ya que en los ejemplos de análisis reales, normalmente no tenemos dicha segmentación. En nuestro trabajo hemos realizamos una mezcla de varias técnicas. Proponemos una metodología para etiquetar acordes de forma automática por medio de un diccionario de acordes que se utiliza como base para correlacionar con los segmentos a etiquetar en la obra musical. Dicha correlación está inspirada en el sistema de clasificación de tonalidades de Krumhansl-Schmuckler, como se describe en el trabajo de Temperley 1999, donde se tiene un perfil de distribución de notas para cada tonalidad y se realiza un análisis de correlación entre ellos y los segmentos que se desean analizar. En nuestro caso lo adaptamos para identificar acordes y no tonalidades, como se mostrará a continuación.

CONSIDERACIONES PARA EL ETIQUETADO DE ACORDES

La identificación de armonías, tonalidades y modulaciones de una pieza musical, es un problema complejo que requiere un conocimiento profundo de las estructuras presentes en el sistema armónico tonal. Como parte del análisis armónico musical, la automatización de este proceso es de gran utilidad para el estudio de la música, ya que permite ahorrar tiempo en cuestiones repetitivas y mecánicas de clasificación, para enfocarse en cuestiones musicológicas más profundas.

Los algoritmos que proponemos están basados en un trabajo donde se conectan conceptos de redes neuronales y minería de datos. Se debe crear una red que identifique correctamente acordes en pasajes musicales donde la armonía puede estar escondida detrás de elementos melódicos o rítmicos. El proceso es similar a la metodología que se utiliza para el reconocimiento de letras automático en la escritura cursiva humana que se mencionó en el capítulo de antecedentes. La analogía consiste en que, en la identificación de un carácter puede estar sujeto a diversas variaciones de trazo, mientras que la información armónica de un pasaje puede variar en cuanto a la disposición de las notas. Aunque el reconocimiento de escritura no tiene relación directa con el estudio de archivos musicales, se pretende crear un algoritmo similar, que tome como entrada eventos musicales de archivos *MusicXML*, en lugar de imágenes, para que pueda identificar acordes sin importar su posición, disposición y contexto. La red neuronal propuesta se entrena utilizando una librería de archivos *MusicXML*, se utilizan técnicas de minería de datos como

clustering y reconocimiento de patrones para extraer la información de estas bases de datos necesaria para el correcto entrenamiento de la red.

Las redes neuronales convolucionales son muy eficientes para la clasificación de imágenes. Se utilizan ampliamente en contextos donde se requiere encontrar e identificar elementos en una imagen. Estas redes se utilizan para el entrenamiento de sistemas como el identificador de dígitos escritos a mano. Es decir, se presenta una imagen con un número escrito manualmente y el sistema debe identificar el valor correcto. El modelo debe ser lo suficientemente flexible para identificar variantes en la forma de escritura.

En cuanto al problema de identificación de dígitos que se menciona, la base de datos *MINST* es una de las más populares, Yann LeCun 2017 (*Modified National Institute of Standards and Technology*), la cual consiste en un conjunto de entrenamiento de 60,000 imágenes y 10,000 ejemplos para prueba. Estos se utilizan para probar la eficiencia de distintos métodos de clasificación, como: regresión lineal, *k-nearest neighbor*, redes neuronales conectadas, redes neuronales convolucionales, entre otros. Como hemos mencionado, este trabajo es una de las bases de analogía para nuestro módulo clasificador de acordes.

Cuando se habla de identificación de patrones y etiquetado de acordes, surge naturalmente la pregunta de cuál es el periódo histórico o estilo en el que se está analizando. En el caso de nuestro trabajo, se busca analizar de forma general, sin estar sujeto a un compositor o estilo en particular. De este modo, el sistema busca los patrones de forma general y

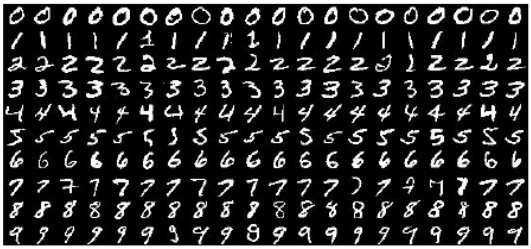

Figura 10: Ejemplos de dígitos en la bilioteca *MINST*, Yann LeCun 2017.

esta puede filtrarse después de acuerdo a las necesidades del investigador.

Por otra parte, para crear la librería de archivos *MusicXML* y *MIDI* que se utilizan para entrenar las redes neuronales, involucra un criterio de selección acerca de las características que deben cumplir dichas piezas musicales.

Para la validación de la eficiencia y precisión de la aplicación final se deben tomar en cuenta cuestiones de análisis musical armónico, donde se puedan equiparar los resultados numéricos del sistema, con información analítica resultante de un estudio manual y tradicional de una obra musical.

Redes neuronales convolucionales

El método de clasificación por medio de redes neuronales convolucionales (RNNC) es uno de los más utilizados para el etiquetado y agrupación de elementos de forma automática. Funciona por medio de la conexión de capas de filtros, donde cada una se encarga de detectar alguna característica del objeto a clasificar, desde muy directas hasta muy abstractas. En el trabajo de Bengio, Goodfellow y Courville 2015, se muestra

una revisión de la teoría detrás de las redes neuronales y las redes neuronales convolucionales.

Por ejemplo, para la detección de un objeto en una imagen, las primeras capas detectan elementos básicos, como detección de contornos, mientras que las capas finales detectan elementos más complejos y abstractos como rostros, que constan de esquinas, contornos y figuras complejas. En la figura 11, se muestra un ejemplo de cómo una RNNC separa una imagen en capas y fragmentos para realizar una clasificación.

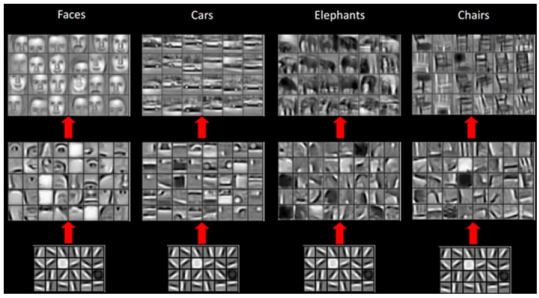

Figura 11: Clasificación de elementos por capas en RNN,RNN 2015.

Las representaciones visuales sirven como analogía para la clasificación que haremos en el contenido simbólico musical. En lugar de tener una imagen, contaremos con la información musical, y en lugar de utilizar filtros de detección de contornos u otras cuestiones visuales, buscaremos que identifique contenido armónico.

Entre los trabajos de clasificación de acordes por medio de este tipo de apendizaje automatico, tenemos trabajos como el de Zhou y Lerch 2015, donde analizan la presencia

y clasificación de acordes en archivos de audio. Al utilizar redes neuronales es difícil elegir el número de capas y nódos necesarios. A esto se le conoce como elección de hiper parámetros, y no existe una metodología exacta, cada problema debe ajustarse y elegir estos parámetros de forma empírica. En el trabajo de Pons y col. 2017, se menciona que es difícil identificar correctamente la arquitectura de una RNNC, ya que hay demasiados parámetros para entrenar. Utilizan distintas combinaciones de filtros. En su trabajo, usan RNNC para clasificar audio, gran parte del problema que deben resolver es la elección correcta de los filtros que la red trabajará.

Como breve nota, es interesante recordar que aunque normalmente se utilizan las redes neuronales para clasificar sonidos, también se pueden usar para resintetizar audio. En el trabajo de Sarroff y Casey 2014, se utilizan una categoría especial de redes neuronales llamados *Autoencoders* para fines creativos, en lugar de clasificar, buscan encontrar los atributos correctos que se pueden obtener del sistema para resintetizar el audio. Ellos proponen que estas técnicas son útiles para representar información sonora reconstruible de una forma compacta.

CLASIFICACIÓN POR LÓGICA DIFUSA

Además de la clasificación por redes neuronales, se trabajó en un sistema por medio de lógica difusa, el cual está inspirado en el trabajo de Müller 2015. Ellos proponen un sistema de segmentación de grupos de notas en acordes donde revisan de forma vertical las notas presentes en un momento dado para etiquetar acordes. Sus funciones de asignación son rígidas,

así que su sistema tiene problemas en identificar notas no estructurales, como notas de bordadura o de paso. En nuestro trabajo, complementamos su propuesta por medio de la introducción de funciones de pertenencia difusa, las cuales explicaremos más adelante.

Para analizar la armonía de una obra musical dentro de la llamada armonía funcional, se siguen varios pasos en los que se busca identificar los acordes y funciones tonales en una obra musical. Generalmente, esto se hace manualmente sobre una partitura impresa, donde el analista identifica las notas que forman parte de un acorde, y luego las agrupa de acuerdo a las tonalidades y funciones adecuadas. Aclaramos que este tipo de análisis aplica a una parte específica del repertorio tradicional de la llamada música de concierto, sin embargo, este tipo de análisis se puede extender a otros contextos armónicos por medio de un ajuste a las funciones de pertenencia.

En este capítulo proponemos una metodología para la clasificación, segmentación y etiquetado automático de acordes, así como identificación de funciones tonales por medio de la clasificación difusa. Como primer paso se crea una tabla ordenada de notas y duraciones tomada de un archivo *MIDI*. A continuación analizamos todas las posibles combinaciones de notas consecutivas, o bien, notas simultáneas, para asignar un acorde y una función tonal a cada uno de ellos utilizando una función de clasificación difusa. Después utilizamos el algoritmo de detección de tonalidad Krumhansl-Schmuckler, descrito en el trabajo de Temperley 1999. Con esto, extraemos lo que ellos llaman tonalidad actual, es decir la tonalidad que domina en el fragmento de la pieza analizada y la usamos para filtrar la lista de acordes de acuerdo a los que tengan

una función tonal de dominante, subdominante o tónica. Como resultado, obtenemos una segmentación y etiquetado de acordes y funciones tonales de acuerdo a lo que resultan del proceso manual y tradicional de reconocimiento de contenido musical.

Preprocesamiento de los datos musicales

El sistema propuesto fue desarrollado en el lenguaje de programación Python. Como entrada, este sistema toma archivos musicales en formato *MIDI* y *MusicXML*. Primero convertimos el archivo *MIDI* a un archivo *csv*[1], el cual es una tabla que contendrá la información musical dispuesta en orden temporal ascendente y contiene los siguientes atributos:

- *Número de nota MIDI*,
- *Duración*,
- *Velocidad*,
- *Track*,
- *Canal*,
- *Posición temporal*.

Con la información contenida en esta tabla, pasamos a la creación de la siguiente *Tabla de segmentos*, en la que cada instancia es un grupo de notas consecutivas en el tiempo, etiquetada con un índice numérico consecutivo. De esta forma,

1 *Comma Separated Values*, Valores separados por comas.

creamos una micro segmentación de la cual se obtendrá el acorde y el etiquetado de la función tonal. Los atributos de la *Tabla de segmentos* son los siguientes:

- Índice de la nota inicial del segmento,
- Índice de la nota final del segmento,
- Etiqueta armónica óptima,
- Valor de pertenencia difusa,
- Función tonal.

Estos atributos serán determinados por medio del uso de las funciones de pertenencia difusa que se presentan a continuación, junto con el cálculo de funciones tonales disponible en la biblioteca de cómputo desarrollado en mi trabajo de tesis de maestría, Bañuelos 2015. De esta tabla se seleccionarán los segmentos con un mayor grado de certidumbre difusa y se les asignará la etiqueta correspondiente.

- Selecciona la primer nota y revisa si es un acorde completo.
- Si lo es, se le asigna una etiqueta armónica.
- Se agrega la siguiente nota consecutiva en el tiempo al acorde pasado.
- Si el acorde sigue siendo el mismo, se extiende la etiqueta del acorde pasado y se agrega esta nota.
- Si el acorde cambia, entonces se guarda esa etiqueta y se comienza de nuevo con la nota nueva.

Funciones de pertenencia difusa para acordes

El objetivo de utilizar funciones de pertenencia difusa es permitir tolerancia a las notas ajenas en la clasificación de acordes. Una de las razones en la que falla el análisis de acordes de bloques es debido a las notas no estructurales. Como ejemplo, en la figura 12, se muestra un acorde Do mayor en estado fundamental, seguido por La menor en primera inversión, conectado por la nota de paso Si. En esta parte del trabajo, no se están tomando en cuenta las inversiones. Al analizar este fragmento como una serie de acordes de bloque, hubiéramos incluido la nota Re en nuestro cálculo. De esta manera, obtendríamos el acorde (Do, Mi, Sol, Re), que no sería reconocido por que no es un acorde común en la música diatónica. En cambio, si pudiéramos ignorar la nota Re, podríamos reconocer el acorde. Esto se puede lograr al asignar un porcentaje de pertenencia de este grupo de notas con el acorde Do mayor. El problema con este método es que necesitamos ser capaces de decidir si una nota es en efecto de paso o no. Para esto, proponemos una definición de las funciones de pertenencia difusa para acordes. Este proceso de identificación es muy natural para una persona con entrenamiento musical, sin embargo, es importante mostrar cómo la computadora *entiende* la música desde los ejemplos básicos para mostrar cómo se utilizan de fundamento para los análisis más complejos.

Necesitamos definir explícitamente una función de pertenencia para cada acorde posible en nuestro diccionario. Para cualquier conjunto de notas, debemos poder calcular grado de correspondencia, con respecto a cada uno de los acordes

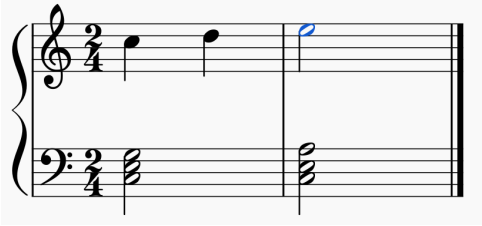

Figura 12: Ejemplo de una nota de paso en una progresión de acordes.

del diccionario. Este se calcula de acuerdo a la función de pertenencia que se muestra a continuación. Como siguiente paso, seleccionamos la etiqueta que tiene mayor correspondencia, de modo que se asocia el acorde más adecuado a un grupo de notas.

La mayor parte del contenido armónico de la música tonal está formada por transponer y transformar una lista de acordes relativamente pequeña. Para este trabajo, buscaremos los acordes de los tipos que se muestran en la tabla 2. Esto se debe a que son los acordes diatónicos que surgen en las tonalidades mayores y menores. Llamaremos a este conjunto de acordes nuestro *Diccionario* y lo denotaremos por *Dict*.

Para poder definir nuestras funciones de pertenencia, establecemos una regla donde revisemos si un arreglo de clases de altura $X = \{x_i\}$ pertenecen a un cierto grado o no, a un acorde en nuestro diccionario. Dado que tenemos doce clases de alturas, y nuestro diccionario tiene once tipos de acordes, necesitamos una lista de 132 funciones, una para cada uno de ellos. Si A es uno de los acordes en nuestro diccionario, y X es un conjunto de clases de alturas por clasificar, entonces

Tipo	Forma normal
M	0, 4, 7
m	0, 3, 7
o	0, 3, 6
+	0, 4, 8
M7	0, 4, 7, 11
D7	0, 4, 7, 10
m7	0, 3, 7, 10
o/7	0, 3, 6, 10
o7	0, 3, 6, 9
mM7	0, 3, 7, 11
D7-5	0, 4, 10

Cuadro 2: Diccionario de acordes para la definición de las funciones de pertenencia difusa.

proponemos la siguiente familia de funciones de pertenencia difusa:

$$\mu_A(X) = \frac{|X \bigcap A|^2}{|X||A|} \tag{1}$$

Es decir, el cuadrado del número de clases de altura comunes entre el acorde del diccionario y el que se quiere clasificar, sobre el producto de el número de elementos en cada uno de los conjuntos. Algo muy importante a considerar es que ambos conjuntos de clases de altura deben estar expresados en su forma normal, es decir, reducidos por octava, sin notas

repetidas y en la forma más compacta. La función propuesta está relacionada con la función de similitud coseno y el índice de Jaccard (Cha 2007), la principal diferencia es que en nuestra definición, estamos interesados en la cardinalidad de los conjuntos y no es sus productos internos. Dos acordes tienen similitud máxima, es decir un valor de pertenencia difusa de 1, si tienen el mismo conjunto de clases de altura en su forma normal. La pertenencia variará de acuerdo a las diferencias de número de elementos y su proporción con las clases de altura comunes.

Una vez que tenemos las funciones de pertenencia difusa, para una colección de clases de altura X, podemos encontrar una lista de los acordes más probables que lo represente. Para esto, evaluamos el conjunto X sobre la lista de las funciones del diccionario y elegimos las que tengan un valor sobre una cota α predefinida, (i.e. $\alpha \in [0,1]$):

$$L_\alpha(X) = \{A \in Dict \mid \mu_A(X) \geq \alpha\} \tag{2}$$

Por simplicidad, podemos encontrar la función que tiene el valor de pertenencia máximo y elegir ese acorde para clasificar X, de este modo se obtiene el valor de pertenencia máximo.

$$L_{\text{máx}}(X) = \underset{A \in Dict}{\text{máx}} \{\mu_A(X)\} \tag{3}$$

Esta ecuación debe usarse con cuidado ya que pueden existir distintas funciones que compartan un valor de pertenencia máximo, y necesitamos información contextual para saber cuál es el adecuado. En este sistema, etiquetamos al conjunto X con el resultado de la ecuación 3, si el valor máximo

es único en el conjunto de acordes posibles, de lo contrario asignamos una etiqueta vacía. Como ejemplo, podemos calcular $L_{0,25}(\{60,62,64,67\})$, mostramos la lista de posibles acordes en la Tabla 3. De entre los posibles 132 acordes en nuestro diccionario, la búsqueda se reduce a un número pequeño de acordes, en los cuales podemos ver que la nota Re (MIDI 62) no afecta a nuestra detección de acorde.

Acorde	**Valor difuso**
CM	0.75
CM7	0.5625
CD7	0.5625
Em7	0.25
Eo/7	0.25

Cuadro 3: Lista difusa de $L_{0,25}(\{60,62,64,67\})$

Si aplicamos la ecuación pasada a los acordes en la Figura 12, probaremos sobre los acordes: (Do,Mi,Sol,Do), (Do,Mi,Sol,Re), (Do,Mi,La,Mi), donde encontramos las etiquetas *Do* mayor con valor difuso 1, *Do* mayor con valor 0.75 y *La* menor con valor 1. De este modo, la nota de paso se manejó correctamente.

Filtrado de acuerdo a funciones armónicas

Una vez que definimos las funciones de clasificación difusas, podemos aplicarlas a la tabla de segmentación que se definió anteriormente, para quitar los valores que no corresponden a acordes del diccionario.

El siguiente paso es filtrar los acordes de acuerdo a la función tonal. Procederemos a calcular la tonalidad de un segmento musical usando el algoritmo de detección de tonalidad *Krumhansl-Schmuckler*, como se describe en los trabajos de Temperley 1999 y Temperley y Marvin 2008. Una vez que se ha identificado la tonalidad posible, se determina la función tonal de cada acorde en nuestra tabla de segmentos filtrados. Sólo se dejarán en la lista los acordes que tengan una función tonal clara y definida en el segmento mayor posible, de este modo, se obtiene un candidato para la etiqueta del acorde y la función tonal.

Para encontrar la función tonal, usamos la biblioteca de código desarrollada por Bañuelos 2015, la cual permite especificar un conjunto de clases de alturas y obtener una lista de las posibles funciones tonales en las veinticuatro tonalidades mayores y menores. En la Tabla 4, se presentan las funciones tonales que identifica el sistema.

Ejemplos de aplicación musical

Para probar el sistema propuesto, usaremos dos ejemplos musicales con una clara estructura armónica, pero con diferente textura en los acordes. El primero los presenta en forma de arpegios y el segundo en acordes verticales en forma de bloque. Con estos dos diferentes estilos podemos contrastar la confiabilidad del método de segmentación.

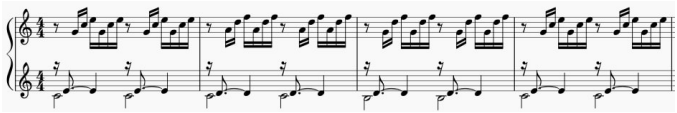

Figura 13: Preludio No.1 en Do mayor, BWV 846, Libro 1, J.S. Bach. Compases 1-4, Mugellini 1964.

Preludio No.1 en Do mayor, BWV 846, Libro 1, J.S. Bach.

Los primeros cuatro compases del Preludio no. 1, en *Do Mayor*, BWV 846, del Clave bien Temperado de Johan Sebastian Bach, son un buen ejemplo para probar el sistema ya que tiene una estructura tonal clara donde los acordes cambian a un paso regular. La progresión armónica está claramente definida como una serie de arpegios, donde se puede ver claramente donde empieza y termina cada armonía.

La progresión armónica que surge es la siguiente:

$$I \rightarrow ii_2 \rightarrow V_5^6 \rightarrow I \tag{4}$$

El resultado de aplicar las funciones de pertenencia difusa se muestra en la Tabla 5. De estos valores se puede apreciar una clara identificación de las funciones tonales de los segmentos que tienen un tamaño máximo y con el valor de pertenencia difusa mayor, lo cual nos da como resultado:

$$I \rightarrow ii_2(ii) \rightarrow V_7 \rightarrow I \tag{5}$$

Por un lado, el sistema claramente identifica la región tonal del acorde subdominante II_7, sin embargo, lo separa en dos diferentes acordes, uno con una séptima y el otro como una triada. Actualmente el sistema no reconoce la inversión

del acorde, solamente el tipo y la función. En este preludio, no aparecen notas de paso, pero nos permite demostrar que el sistema obtiene resultados adecuados para música con acordes arpegiados.

Es interesante observar los acordes de transición en el análisis, por ejemplo el La menor de la segunda instancia de la tabla. Esta no es una función tonal que se utiliza en esta composición, pero aparece debido a que la segmentación difusa intenta crear una fusión entre el acorde Do mayor y Re menor. En el arpegio de transición tenemos un segmento que forma un acorde con las notas Do y Mi del primer acorde y La y Re del segundo. Este acorde es eliminado por el sistema de filtrado que revisa la longitud de los segmentos, en donde sólo aceptaremos etiquetas de conjuntos de notas con longitud mayor a un cierto valor predefinido, el cual se necesita adecuar a la pieza que se analice.

Preludio Op. 28, No. 20 F. Chopin

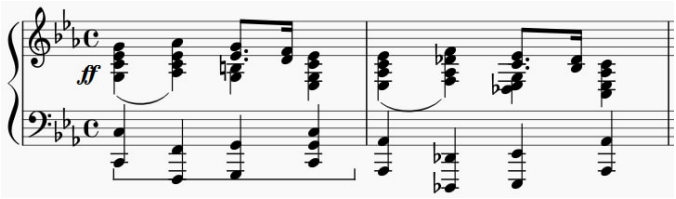

Figura 14: Preludio Op. 28, No. 20 F. Chopin, en Do menor. Compases 1-2, (Theodor 1882).

En el preludio No. 20, Op. 28 de F. Chopin, Theodor 1882, tenemos un fragmento musical con movimiento armónico

que se presenta principalmente en conjuntos de acordes en forma de bloques verticales. A diferencia de la pieza pasada, la tonalidad es Do menor. De acuerdo al análisis realizado en Gilbert 2017, obtenemos la siguiente progresión:

$$i \rightarrow IV_7 \rightarrow V_7 \rightarrow i \rightarrow IV_7 \rightarrow N_\flat \rightarrow DV_7/VI \rightarrow VI \qquad (6)$$

En este preludio, el *tactus* es el doble de tiempo que en el preludio de Bach, de modo que el sistema no tiene suficiente información para determinar cuáles notas son las predominantes del acorde. En el preludio de Bach, cada instancia del acorde presenta entre ocho y dieciséis notas. En el preludio de Chopin aparecen algunas etiquetas que no son correctas, podemos ver en la Tabla 6, que aparecen etiquetas adecuadas, estas se muestran en letra negrita.

Con esto, tenemos un sistema que etiqueta acordes y sus respectivas funciones tonales de un fragmento de archivos *MusicXML* por medio de funciones de pertenencia difusa y filtrado de tonalidades. Estas funciones ayudan con el problema de tener notas no armónicas en los acordes, como notas de paso; sin embargo de acuerdo a los resultados de análisis en las dos piezas: Preludio No.1 en Do mayor, BWV 846, Libro 1 de J.S. Bach y el Preludio Op. 28, No. 20 por F. Chopin en Do menor, vemos que el sistema funciona mejor con fragmentos que tienen más notas por acorde.

En este capítulo se mostraron dos técnicas de identificación y clasificación de acordes en archivos MusicXML. Entre las principales contribuciones de los métodos que presentamos se encuentra una mejor segmentación de archivos por medio de funciones de pertenencia, así como proveer de flexibili-

dad a los sistemas de clasificación de acordes por medio de funciones de pertenencia difusa. Este sistema de clasificación puede extenderse a distintos estilos de armonía si cambiamos el diccionario de acordes, por ejemplo, si se busca analizar música que no es tonal, pero utiliza ciertos grupos de notas, como *clusters*, bastaría con cambiar el diccionario por uno que contenga estos grupos de notas.

Como se verá en los siguientes capítulos, este módulo es solo un fragmento de los métodos con los que podemos clasificar la información contenida en archivos musicales. Al interactuar con los distintos módulos, el usuario puede analizar distintos aspectos de la música de forma interactiva y accesible.

Mayor	**Acorde**	**Menor**	**Acorde**
I	f + 0, M	i	f + 0, m
ii	f + 2, m	ii	f + 2, o
iii	f + 4, m	III	f + 3, M
IV	f + 5, M	iv	f + 5, m
V	f + 7, M	v	f + 7, m
vi	f + 9, m	VI	f + 8, M
vii	f + 11, o	VII	f + 10, M
I7	f + 0, M7	i7	f + 0, m7
ii7	f + 2, m7	ii7	f + 2, o/7
iii7	f + 4, m7	III7	f + 3, M7
IV7	f + 5, M7	iv7	f + 5, m7
V7	f + 7, D7	v7	f + 7, m7
vi7	f + 9, m7	VI7	f + 8, M7
vii7	f + 11, o/7	VII7	f + 10, D7
ii arm	f + 2, o	iii arm	f + 3, +
vi arm	f + 9, +	V arm	f + 7, M
vii arm	f + 11, o7	vii arm	f + 11, o
iv arm	f + 5, m	V7 arm	f + 7, D7
		vii7 arm	f + 11, o7

Cuadro 4: Diccionario de funciones tonales, donde f es la nota fundamental.

Tamaño	I_i	I_j	Normal	Etiqueta	Fuzzy	Función
18	**0**	**18**	**0-4-7**	**CM**	**1**	**I**
6	13	19	0-2-4-7-9	Am7	0.8	VI7
6	14	20	9-0-2-4	Am	0.75	VI
14	**16**	**30**	**9-0-2-5**	**Dm7**	**1**	**II7**
15	**17**	**32**	**2-5-9**	**Dm**	**1**	**II**
7	27	34	9-11-2-5	Bo/7	1	VII7
4	30	34	11-2-5	Bo	1	VII
16	**30**	**46**	**11-2-5-7**	**GD7**	**1**	**DV7**
4	32	36	7-11-2	GM	1	V
4	38	42	11-2-5	Bo	1	VII
4	40	44	7-11-2	GM	1	V
22	**44**	**66**	**0-2-4-5-7**	**CM**	**0.6**	**I**

Cuadro 5: Preludio No.1 en Do mayor, BWV 846, Libro 1, J.S. Bach, análisis filtrado.

Inicio, final	Etiqueta	Fuzzy	Grado
0-7	**Cm**	1.00	I
2-10	AbM	1.00	VI
5-11	**Fm7**	1.00	IV7
6-10	AbM	1.00	VI
6-12	Fm7	1.00	IV7
10-14	**GD7*5**	0.67	*V7
14-20	**GD7**	0.80	*DV7
15-19	**GD7*5**	1.00	*V7
15-21	**GD7**	1.00	*DV7
16-19	**GD7*5**	0.67	*V7
18-24	Cm	0.60	I
19-23	EbM7	0.75	III7
19-27	**Cm**	0.75	I
22-31	AbM7	1.00	VI7
26-33	AbM	1.00	VI
28-34	**Fm7**	1.00	IV7
29-39	DbM	0.75	N
34-40	DbM7	1.00	N
35-39	**DbM**	1.00	N
35-40	DbM7	1.00	N
36-42	AbM	0.75	VI
40-47	EbD7*5	0.67	x
42-48	**EbD7**	1.00	*DV7/VI
43-47	EbD7*5	1.00	x
43-48	**EbD7**	1.00	*DV7/VI
44-47	EbD7*5	0.67	x
44-55	**AbM**	0.60	VI

Cuadro 6: Preludio Op. 28, No. 20 F. Chopin en Do menor, análisis filtrado.

4

ANÁLISIS DE LA FORMA MUSICAL DTW

IDENTIFICACIÓN DE ESTRUCTURA MUSICAL

La extracción de información musical utiliza técnicas para obtener datos estructurales de piezas musicales, sin embargo, existen pocos intentos de identificar la forma musical de archivos digitales.

En esta sección presentamos una implementación del algoritmo Ajuste de Tiempo Dinámico para la identificación automática de la forma musical por medio de una matriz de segmentación en la cual se agrupan los fragmentos de acuerdo a la similitud máxima. El sistema se implementó utilizando la biblioteca *music21* para analizar archivos simbólicos, se probó en dos piezas: La Bagatela No. 25 en La menor de L.V. Beethoven, y la sonata para piano No. 11 en La mayor, K331, tercer movimiento de W.A. Mozart. El sistema obtuvo una identificación correcta de las secciones similares, amabas con una forma rondó. Se prevé que este algoritmo puede extenderse para medir similitud armónica y de esta forma sea capaz de analizar formas más complejas. Las técnicas que se

presentan a continuación fueron desarrolladas en el trabajo de Bañuelos y Orduña 2017.

El resultado de esta sección es uno de los módulos que forman parte de la aplicación de usuario final. Este módulo se encarga de encontrar secciones de similitud dentro de una pieza o entre distintas obras de un corpus. La similitud puede ser melódica, rítmica o armónica.

Extracción de información de XML

Entre los problemas más comunes que *MIR* aborda tenemos los sistemas de recomendación por género, como en Kaminskas y Ricci 2012b, imitación de estilo con inteligencia artificial y aprendizaje automático Dubnov y col. 2003, etiquetado y reconocimiento de acordes Pardo y Birmingham 2002, identificación de instrumentos musicales en un archivo de audio Park y Lee 2015. Sin embargo, existen pocos intentos para analizar y extraer la forma musical, entre ellos tenemos trabajos como Lamere 2012, en el que encuentran relaciones de similitud interna en un archivo de audio digital para agrupar las secciones que puedan servir de transición entre ellas. En Cooper y Foote 2002, analizan la similitud interna de un archivo de audio para generar una representación simplificada del audio completo, es decir, encontrar la mínima cantidad de sonido que representa al archivo completo, esto tiene una gran utilidad para simplificar la busca de sonido en bases de datos.

Entre los trabajos que utilizan mediciones de similitud entre melodías, tenemos el de Aucouturier, Francois Pachet y col. 2002. Para encontrar similitud en archivos de audio,

utilizan comparación tímbrica con coeficentes *cepstrum*. En el trabajo de Martínez y Liern 2017 se muestra una técnica para encontrar similitud de melodías utilizando clasificación difusa.

En el análisis de archivos de partituras digitales existe una necesidad de aplicar estos métodos de análisis de similitud para crear un sistema que clasifique las relaciones internas de una pieza de modo que se pueda identificar la forma musical o bien, las estructuras de repetición.

En este trabajo proponemos aplicar el algoritmo *Análisis de tiempo dinámico*, (*Dynamic Time Warping - DTW*) en archivos de tipo *MusicXML*. Este algoritmo normalmente se utiliza en señales de audio y es útil para medir qué tan diferentes son dos señales de audio por medio de una medida que indica que tan costoso sería transformar una en la otra por medio de estiramientos temporales. La ventaja de aplicar el algoritmo en este tipo de archivos, es que su tamaño es mucho más pequeño, lo cual hace que el proceso de análisis sea más rápido y eficiente.

Para clasificar las estructuras de repetición musical, hacemos una comparación entre todos los posibles sub-segmentos de la pieza y los comparamos entre sí para crear una matriz de similitud, como la utilizada en el trabajo de Cooper y Foote 2002; después agrupamos los segmentos con índice de similitud máxima y los etiquetamos de acuerdo a su grupo. El proceso se repite hasta que todas las similitudes máximas se hayan encontrado. Este proceso se repite a continuación y como ejemplo de este módulo de análisis, lo utilizamos para analizar un par de piezas con la forma rondó.

Ajuste de tiempo dinámico

El algoritmo *DTW* utiliza la técnica de programación dinámica para encontrar la alineación óptima entre dos series de tiempo. Esto se logra al calcular el costo que tiene alinear cada uno de los puntos de la primera con los de la segunda. Una vez que se tiene una matriz de costos, se encuentra la colección de ajustes que minimiza el costo general de transformación. Debido a su flexibilidad para reconocer similitud, consideramos que su aplicación al contexto musical obtendrá buenos resultados.

A continuación se presenta una breve explicación del algoritmo *DTW* como lo describe Müller en Müller 2007. Si tenemos dos secuencias $X := (x_1, x_2, ..., x_N)$ y $Y := (y_1, y_2, ..., y_N)$, con la asociación entre ellos, llamada camino, $p = (p_1, \ldots, p_L)$ se define como la relación entre los puntos x_{nl} y y_{ml}. El camino debe satisfacer las siguientes condiciones de frontera: los primeros y últimos elementos de las secuencias siempre se relacionan entre ellos; la asociación debe ser monótona, es decir, no puede tener saltos temporales; el camino sólo puede avanzar en pasos unitarios; por último, todos los elementos de X deben estar relacionados con los de Y y viceversa, sin repeticiones. Müller define el costo total como la suma del costo de transformar cada uno de los puntos de X a los puntos de Y:

$$c_p(X,Y) := \sum_{l=1}^{L} c(x_{nl}, y_{ml}). \tag{7}$$

Donde c es el costo local, en este caso se utiliza la distancia euclídea como medición de diferencia. Para encontrar el

costo óptimo se busca entre todos los caminos posibles hasta encontrar el que tenga el valor menor.

$$DTW(X,Y) = \text{mín}\{c_p(X,Y) \mid p \text{ es un camino}\} \quad (8)$$

Con esto encontramos el costo mínimo de todos los caminos posibles de transformación de X a Y. Esta medida es útil para comparar la similitud de segmentos en la pieza. El sistema desarrollado en este trabajo utiliza la biblioteca *Fast DTW library* desarrollada por Salvador y Chan 2007, la cual está basada en conceptos de programación dinámica. La distancia *DTW* puede ser utilizada en sistemas de búsqueda de bases de datos donde se compara el segmento de referencia con los elementos de la misma, como se presenta en Hu, Dannenberg y Tzanetakis 2003. Para aplicar esto a los archivos musicales simbólicos, utilizaremos las secuencias de notas obtenidas de archivos *MusicXML*.

Matriz de segmentación

El sistema que se propone fue implementado en el lenguaje de programación Python con la biblioteca *music21*, la cual es óptima para procesar información simbólica de archivos *MIDI* y *MusicXML*. Para aplicar el algoritmo *DTW* se requiere preparar los datos de modo que tomen la forma de una serie de tiempo. Para esto utilizamos la función *flat* de *music21*, la cual convierte el archivo *XML* a un representación secuencial de las notas y sus tiempo. Como siguiente paso se traduce la información musical a una serie de tiempo de todas las notas,

de modo que obtenemos una secuencia en orden temporal de pares ordenados, con la información de la altura de la nota y el tiempo en que aparece.

$$P = \{(time_i, pitch_i)\} \tag{9}$$

Donde $0 \leq i \leq N$, y N es el total del número de notas en el archivo por analizar, llamaremos lista de eventos a la variable P.

Como siguiente paso, se crea una lista de todos los posibles sub segmentos de la lista de eventos, es decir, todos los posibles subconjuntos de P donde los elementos consecutivos de i a j están presentes.

$$Segs = \{U(i,j) \subset P| \text{ if } i \leq k \leq j, \implies (t_k, p_k) \in U(i,j)\}. \tag{10}$$

Utilizaremos la notación $Segs_{i,j}$ para indicar los segmentos de notas consecutivas del elemento i al j en la lista de eventos. Existen N^2 segmentos diferentes y necesitamos agruparlos de acuerdo a su similitud. Para lograr esto definimos la matriz de similitud en donde el elemento (p,q) contiene el índice de similitud entre $Segmento_p$ y $Segmento_q$, es decir

$$(a_{p,q}) \in R^{N \times N} \tag{11}$$

$$a_{p,q} = DTW(Seg_p, Seg_q) \tag{12}$$

En esta matriz, calculamos la medida de similitud DTW entre todos los pares de segmentos, de esta manera podemos clasificar las regiones similares en una pieza. El método para

calcular la medida *DTW* es el que muestra Müller Müller 2007, donde utiliza programación dinámica para encontrar el costo de transformar una serie en la otra, como se explicó en la sección pasada. En el siguiente fragmento de código se muestra un fragmento de la implementación de esta segmentación en *Python*.

```
1  def graph_dtw_multiple_sizes(music, minMeas=1, maxMeas
       =10):
2      music  = mu.remove_breaks(music)
3      music_chords = music.chordify()
4      music_chords= mu.remove_breaks(music_chords)
5      music_measures = music_chords.getElementsByClass(
           stream.Measure)
6
7      numMeasMax= len(music_measures)
8
9      plt.figure()
10     for numMeasures in range(minMeas,maxMeas):
11         mx = numMeasures
12         if numMeasures>=numMeasMax:
13             mx = numMeasMax-1
14         testMeasure = measure_list_to_serie(
               music_measures.measures(0,numMeasures) )
15         listDistances=[]
16         for idx in range(len(music_measures)):
17             mxId = idx+numMeasures
18             if numMeasures>=numMeasMax:
19                 mxId = numMeasMax-1
20             m =music_measures.measures(idx,mxId)
21             currentMeasure = measure_list_to_serie(m)
22             distance, path = fastdtw(currentMeasure,
                   testMeasure, dist=euclidean)
23             listDistances.append(distance)
```

Código 4.1: Algoritmo de segmentación.

PRUEBAS MUSICALES

Para probar el sistema, elegimos dos piezas que tienen la forma rondó, de modo que podamos probar si el algoritmo predice correctamente la sección de repetición en el análisis.

En la Bagatela en La menor, *Für Elise*, de Beethoven, tenemos una forma de rondó, en la cual tenemos una sección A al principio, y la forma musical en general sigue una estructura del tipo A - B - A - C - A. Cuando aplicamos la medida *DTW* a esta pieza, obtenemos un claro indicio de una sección de repetición.

En la figura 15a, se muestra una gráfica del costo de una ventana de prueba. La gráfica toma una ventana inicial de tamaño 30, e itera sobre los demás costos en pasos de 10 notas, donde se indica el costo de comparar la ventana con cada uno de los demás segmentos del archivo. El tamaño inicial de la ventana y el tamaño del paso se eligieron de forma empírica. El programa incrementa el tamaño de la ventana para revisar si una longitud mayor resultaría en un emparejamiento mejor. Las ventanas cambian de tamaño desde 30 a 100 notas en pasos de 10. La figura 15 muestra los costos de distintos tamaños de ventanas.

Podemos ver que existen mínimos locales alrededor de las notas 20, 55 y 130, efectivamente se aprecia que existe una clara similitud entre estas secciones y la ventana de prueba. Llamaremos a esta sección de repetición A, la cual en efecto corresponde a la sección de repetición del rondó. Para analizar si las partes que difieren de A, son en efecto secciones B y C, necesitaríamos aplicar el algoritmo en las secciones que no corresponden a la parte A, y buscar similitudes entre

ellas, esto será abordado en trabajos futuros, de momento el algoritmo muestra buenos resultados de clasificación de una sección a la vez.

En el tercer movimiento de la sonata para piano de Mozart No. 11 en La mayor, K331, también conocido como *Rondo Alla Turca*, tenemos un comportamiento similar al analizado en Beethoven, al aplicar el algoritmo *DTW* obtenemos la gráfica de costos de la figura 15 b. Podemos ver que el inicio de la sección de la pieza se repite en los tiempos 40, 140 y 160. En 140 tenemos una repetición casi idéntica sin importar el tamaño de la ventana, mientras que en otros casos se muestra una discrepancia menor en la estructura, como se muestra por el aumento de los costos obtenidos. Denotaremos con la letra A, esta primera sección. De nuevo, para encontrar el resto de las secciones de la pieza, necesitaríamos aplicar el algoritmo al resto del archivo que no pertenece en A, este análisis se abordará en trabajo futuro.

El uso de *DTW* para comparar las secciones internas de una pieza musical nos da suficiente flexibilidad para comparar secciones por medio del cálculo de costo entre todas las combinaciones de segmentos y tamaños de ventanas, lo cual utilizamos para encontrar patrones de repetición, variación o cambios. El trabajo presente muestra una forma de identificar las secciones de repetición, más adelante se implementará la funcionalidad de identificar múltiples secciones en una misma obra. Uno de los principales retos a resolver es que no se tiene un valor máximo para el valor de la ventana, de modo que debemos decidir un valor de costo de tolerancia.

En trabajos futuros, esto llevará a utilizar medidas de asociación difusa para clasificar adecuadamente los segmentos.

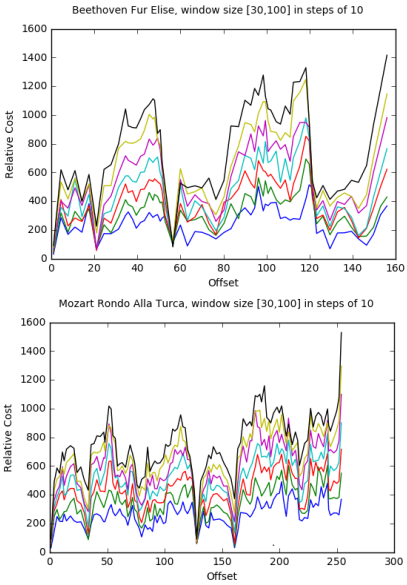

Figura 15: Análisis de costo (a) Für Elise de L.V. Beethoven y (b) Rondo Alla Turca de W.A. Mozart.

La implementación actual sólo toma en consideración la altura de la nota y el tiempo en el que aparece, sin embargo, posteriormente se podrían implementar más parámetros, como distancia armónica, la cual nos podría ayudar a clasificar de forma más robusta. En conclusión, el algoritmo *DTW* nos da una buena estimación de cómo encontrar similitud interna en archivos musicales para encontrar la forma musical. Se prevé que este tipo de enfoques, complementado con otras medidas de similitud puede contribuir a trabajos futuros en la investigación para la extracción de contenido musical

Este módulo de análisis de estructura se complementa muy bien con el de análisis de acordes, ya que entre los dos

se pueden obtener descriptores más complejos que pueden ayudar a la organización y clasificación de archivos. En los próximos capítulos veremos cómo integrar diversos módulos para el análisis de un corpus en específico.

5

SISTEMA DE BÚSQUEDAS

Como parte de los módulos de *MIR* que se implementan en el sistema con arquitectura orientada a servicios (*SOA*), en este capítulo se presenta una aplicación a un problema real de una colega del posgrado en música de la facultad de música de la UNAM, la investigadora Andrea Zamora. En su trabajo necesita encontrar relaciones entre el texto vocal y el contenido musical de la *MISSA PRO DEFUNCTIS* de Tomás Ochando.

Para resolver este problema, presentamos un sistema de análisis de partituras que compara el material de texto vocal con el contenido armónico, melódico y rítmico, para encontrar relaciones de interés musicológico. Se describe un método para realizar un análisis de similitud de texto y contenido musical en el contenido de una pieza musical. Nos interesa encontrar las repeticiones de texto y las repeticiones de contenido musical y analizar si existe alguna relación entre ellos.

EXTRACCIÓN DE INFORMACIÓN DE TEXTO

El problema que nos interesa resolver en esta sección es: por un lado, elegir una frase o palabra y encontrar todas las instancias donde aparece en el corpus. Una vez que se tienen localizadas todas las repeticiones, pasaremos a comparar el contenido musical, para ver si existe alguna relación entre ellas, esto se realiza por medio de análisis con los módulos de identificación de acordes y estructura descritos en los capítulos anteriores.

En y a través de la Misa de Requiem de Tomás Ochando, abordaremos dos problemas de búsqueda: por un lado, buscamos las repeticiones de las frases más representativas de la obra y las repeticiones de contenido musical para ver si existe alguna relación entre ellos. Por otra parte, nos interesa encontrar un análisis de secciones solistas y densidad de aparición de voces en la obra, es decir, saber en qué momentos de la misa aparecen solos, y cómo éstos interactúan con el resto del contenido musical.

Como primera parte, analizaremos las repeticiones de texto vocal y haremos un análisis de acordes, melodías y rítmos en esos puntos de la partitura. Como antecedentes del análisis en conjunto de texto y melodía tenemos el trabajo de Nichols y col. 2009 quienes aplican dicho análisis a la música popular moderna, entre otros, les interesa resolver los siguientes puntos: *Salience Lyrics, Salience music, Melodic peak, Metric position, Relative duration*.

ANÁLISIS DE SOLISTAS

Junto con las relaciones texto - música, para la investigadora es de interés estudiar las secciones de solistas y su relación con el contenido musical. Para lograr esto, se analizan las partes en una partitura orquestal en donde aparecen diferentes combinaciones de instrumentos. Queremos encontrar los lugares donde aparecen todos los instrumentos y lugares donde hay pares y situaciones similares. Esto es en cierta forma esto un estudio del timbre.

En cuanto a la misa, podemos mostrar un análisis por grupos de instrumentos: las voces, los alientos y las cuerdas, como en las siguientes figuras.

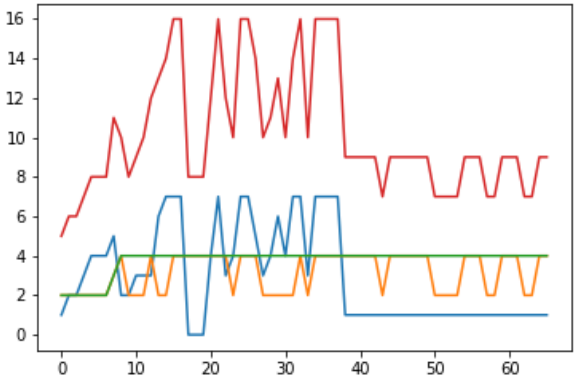

Figura 16: Vector de activación de grupos de instrumentos: voces, alientos, cuerdas y todos.

Es importante separar el análisis por grupos de instrumentos. Por un lado las voces, y por otro, los instrumentos. Para poder identificar claramente los puntos de solistas vocales.

En la figura 16, se muestran las gráficas de distintas combinaciones de voces, es interesante ver la actividad musical que se genera en esos momentos.

El primer objetivo mencionado consiste en elegir una frase o palabra, por ejemplo “Requiem” o “Dona eis Domine” y encontrar todas las instancias donde aparece en el corpus, en este caso, a lo largo de la misa. Una vez que se tienen localizadas todas las repeticiones, pasaremos a comparar el contenido musical, para ver si existe alguna relación entre ellas. Las similitudes que buscamos podrían ser armónicas, melódicas o rítmicas.

Para lograr lo anterior se divide el problema en los siguientes pasos:

- Se cargan en el programa los archivos a analizar, en este caso, se deben poner todos los archivos xml en una carpeta llamada “datos”.

- El programa busca en todos los archivos, las instancias del objeto Nota, y selecciona solamente aquellos que tienen asociado un fragmento de texto, la mayoría de las veces es una sola sílaba. Con esto se crea una lista de todas las notas con su respectivo texto.

- Se define una frase de búsqueda “Requiem” y se introduce al sistema. El programa corre sobre el listado de notas de todos los archivos para encontrar las partes donde aparece y se crea una nueva lista de los resultados.

- Se imprime al usuario el producto resultante de la búsqueda, el cual incluye: el fragmento de la partitura, una lista de las notas que aparecen y una lista de las duraciones.

- Por último se muestran una serie de gráficas de las notas y las duraciones, de modo que el usuario pueda analizar si existe algún patrón de interés.

El segundo objetivo: encontrar las secciones solistas y de densidad orquestal requiere de los siguientes pasos:

- Al igual que en el problema anterior: se cargan al programa los archivos a analizar, en este caso, se deben poner todos los archivos xml en una carpeta llamada "datos".

- Se define una lista de los nombres de las voces que se quieren analizar, por ejemplo: voicenames = ['Tiple 1.1', 'Tiple 1.2', 'Alto 1', 'Tenor 1', 'Tiple 2.1', 'Alto 2', 'Tenor 2', 'Bajo 2']. Es necesario que los nombres de las voces sean consistentes entre archivos, de modo que lo que es Tiple en uno, no puede ser Soprano en otro. Más adelante se podría establecer un sistema de relaciones y similitudes, pero de momento es importante que los archivos sean consistentes.

- Se pide al programa que organice en listas separadas los eventos musicales de cada una de las voces. Se revisa cada compás de la partitura y se le asigna un 1 si aparece alguna nota, y un 0 si el compás tiene sólo silencios.

- Se suman los valores de todas las voces en cada compás y se obtiene el número de voces que aparecen simultáneamente.

- Por último se muestra al usuario una gráfica por cada archivo, donde se evidencia la forma en que las voces interactúan.

Parte de lo anterior puede codificarse de la siguiente manera utilizando *music21*.

```
1     active_measures = []
2     for m in part.getElementsByClass('Measure'):
3         is_playing = False
4         for n in m.recurse():
5             if type(n) == note.Note:
6                 is_playing = True
7         active_measures.append(int(is_playing))
8     return active_measures
```

Código 5.1: Número de compases activos.

En la figura 17; se muestran algunas gráficas de los resultados de la búsqueda: *Requiem*, así como en la figura 16, las gráficas de densidad orquestal. Como se muestra a continuación, este sistema es útil para el análisis ya que nos permite encontrar la interacción de distintas voces. En el caso de Tomás Ochando, nos servirá para entender el uso de las voces y pasajes solistas en el estilo del compositor.

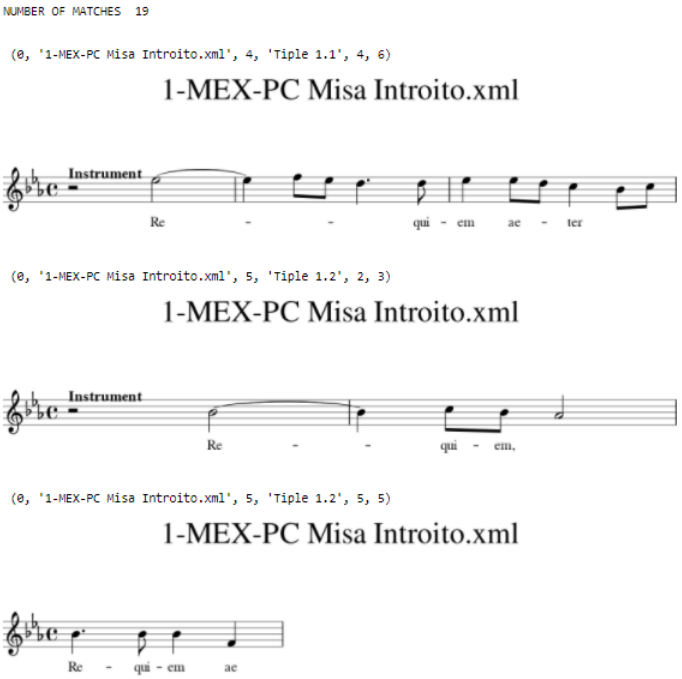

Figura 17: Resultado de búsqueda de la frase “Requiem”.

A continuación se muestra un fragmento del código necesario para realizar las búsquedas de texto.

```
1      list_matches = []
2      for i in range(len(lyrics_db_array)):
3          current_phrase = ''
4          j = 0
5          while len(current_phrase) < len(test_string):
6              list_sylable = lyrics_db_array[(
7                  i+j) % len(lyrics_db_array)]['text']
8              current_phrase += list_sylable
9              test_string_form = test_string.replace(' ',
                   '').replace(
```

```
10                ',', '').replace(';', '').replace('.', '
                      ').lower()
11            current_phrase_form = current_phrase.replace
                  (
12                ',', '').replace(';', '').replace('.', '
                      ').lower()
13            if test_string_form == current_phrase_form:
14                path_name = lyrics_db_array[i]['filename
                      ']
15                part_name = lyrics_db_array[i]['partname
                      ']
16                start_meas = lyrics_db_array[i]['measure
                      ']
17                end_meas = lyrics_db_array[i+j]['measure
                      ']
18                new_entry = {
19                    "query": test_string,
20                    "filename": path_name,
21                    "partname": part_name,
22                    "startmeasure": start_meas,
23                    "endmeasure": end_meas,
24                }
25                list_matches.append(new_entry)
26            j += 1
27
28    return list_matches
```

Código 5.2: Búsqueda de textos.

El módulo desarrollado en este capítulo es importante ya que se nutre de los otros módulos desarrollados en los capítulos pasados, y muestra un ejemplo de aplicación a un problema de investigación real de los algoritmos que se utilizan en *MIR*. En el próximo capítulo, veremos otro ejemplo de clasificación, ahora de archivos, por medio de los módulos pasados y uno nuevo.

6

SISTEMA EXPERTO PARA CLASIFICACIÓN DE ARCHIVOS

A continuación se presenta otro módulo de análisis *MIR* que se implementa en el sistema modular que engloba este trabajo. El problema a resolver es la clasificación de piezas musicales por medio de la traducción de un tratado musical a un sistema experto. Se eligió trabajar con el tratado de guitarra barroca de Gaspar Sanz, *Instrucción de música sobre la guitarra española y métodos de sus primeros rudimentos hasta tañerla con destreza*, ya que incluye reglas armónicas detalladas y bastantes ejemplos de partituras que nos sirven para validar los ejemplos.

En este capítulo se crea un sistema experto de clasificación de archivos musicales de música basado en el tratado de guitarra barroca de Gaspar Sanz. Se eligió este tratado musical ya que contiene dos cosas muy importantes para entrenar un sistema: por un lado, instrucciones específicas de cómo debe comportarse la armonía de los distintos tipos de música que incluye el tratado; por otro, incluye una gran cantidad de ejemplos musicales e instancias de las partituras.

De este modo se puede crear un sistema que se ajuste a los requerimientos mostrados por Sanz.

El sistema resultante toma como entrada una partitura en formato *MusicXML* y como resultado mostrará una propuesta de clasificación de archivos de acuerdo a las reglas y ejemplos del tratado. Por ejemplo si se le introduce una pieza de tipo *folía*, el sistema debe ser capaz de reconocerlo y asignar la etiqueta apropiada.

Varios autores han trabajado con la clasificación automática de lo que ellos llaman estilo musical, que en este contexto consiste en los descriptores melódicos, armónicos y rítmicos de una pieza en particular. Dos de los más interesantes son el enfoque de McKay 2010, y el de Armentano, De Noni y Cardoso 2017. Ambos proponen utilizar una serie de descriptores elegidos de acuerdo a sus necesidades.

Como se explicó en la sección de antecedentes, estas características o descriptores son una serie de parámetros que representan de forma sintética información que caracteriza a una pieza. Entre ellas, podemos tener el rango de notas presentes; las notas más comunes de la pieza; la relación temporal entre los silencios y las notas; el índice en que una obra es diatónica o no, entre muchos otros.

McKay 2010, en su software *jMIR* establece una serie de más de veinte descriptores que pueden utilizarse para clasificar, sin embargo en el trabajo de Armentano, De Noni y Cardoso 2017, se sugiere que basta un número pequeño de *features* simbólicos para hacer una clasificación correcta, el problema es elegir las adecuadas. Ellos sugieren utilizar los descriptores más comunes para el humano. También se pueden usar piezas "ejemplo" para comparar. Explican la

diferencia entre genero musical y estilo musical y al igual que en nuestro trabajo, transforman *MIDI* a *XML*.

Por su parte Shan y Kuo 2003, realizan clasificación de lo que ellos llaman género, por medio de análisis melódico en archivos *MIDI*. Ellos utilizan una armonización automática por medio de algoritmos basados en lo que llaman teoría musical y progresiones de guitarra utilizando un sistema de puntos. Para la clasificación de géneros, algunos autores sugieren mezclar características de archivos de audio, simbólicos y culturales, McKay y Fujinaga 2008.

En el trabajo de Pérez-Sancho, Inesta y Calera-Rubio 2005, se muestra una clasificación de géneros utilizando *Naive Bayes*, en el cual analizan piezas *MIDI* para agruparlas de acuerdo con parámetros estadísticos útiles. En el caso de clasificar compositores con estilos similares, tenemos el trabajo de Herlands y col. 2014. Ellos buscan cómo distinguir entre compositores que escriben de forma parecida. Al igual que nosotros utilizan *music21*, y como característica de su trabajo se enfocan en utilizar descriptores de alto orden.

DEFINICIÓN DEL SISTEMA EXPERTO

Reglas de asociación

Para crear el sistema experto que necesitamos, utilizaremos análisis de asociación para encontrar relaciones de reglas entre los elementos. Necesitamos definir el soporte y la confianza de las reglas, de acuerdo a las definiciones de Tan, Steinbach, Kumar y col. 2006. Para lograr esto se toma la transcripción digital en *MusicXML* del tratado de Gaspar

Sanz y se extraerán las reglas que nos permiten identificar la categoría de similitud de los archivos.

Gaspar Sanz fue un compositor Español del siglo XVII, una de sus más grandes contribuciones a la música, fue su tratado de guitarra española (Sanz 1674), el cual está enfocado a introducir a personas interesadas en el mundo de la guitarra. Fue un trabajo revolucionario, ya que incluye detalladamente una lista de acordes y cómo deben ser tocados en la guitarra barroca[1] con unos diagramas de trastes y los dedos correspondientes con que deben presionarse. Incluye también un diagrama al que llama laberinto donde agrupa los acordes de acuerdo a su sistema.

Por otra parte, el tratado de Sanz, nos muestra una lista de progresiones de acordes posibles de acuerdo al tipo de pieza, es decir, si es una pasacalle, un canario, gallarda, etc. Estos se muestran como una secuencia de letras de acordes separados por líneas verticales. Podemos ver el original de los acordes básicos, el laberinto y un ejemplo de progresiones de una pasacalle en la Figura 18. En la Figura 20 se muestran otras listas de diferentes progresiones de acordes posibles según la pieza. Esto es de gran utilidad para nuestro sistema ya que con el diccionario y las reglas de progresión podríamos implementar un sistema de reglas que nos permita analizar piezas musicales que se ciñan al tratado de Sanz.

1 La guitarra barroca se afina igual que la guitarra moderna, sólo que no tiene la cuerda más grave.

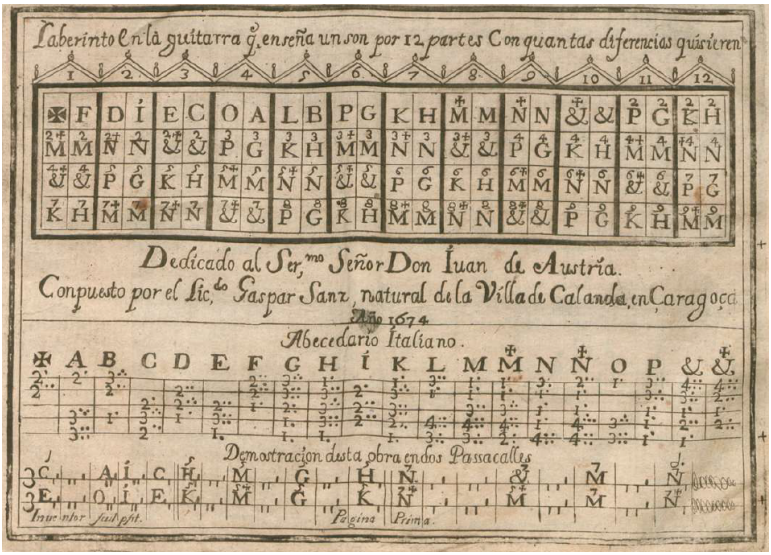

Figura 18: Superior: diagrama de Laberinto. Medio: diccionario italiano de acordes. Inferior, ejemplos de progresiones.

Transcripción

Necesitamos traducir las reglas del tratado de Sanz a una base de conocimiento que corresponde a la primera parte de nuestras reglas de asociación. Para lo anterior se utilizó *music21* y se creó una librería para encontrar la equivalencia de los acordes en notación italiana a nuestra notación actual.

A continuación se muestra una lista con todos los tipos de piezas que se mencionan en el tratado, junto con las progresiones de acordes que sugiere Sanz. Desde el punto de vista computacional, las reglas se guardan en un formato *json*, que es muy útil para representar información relacional.

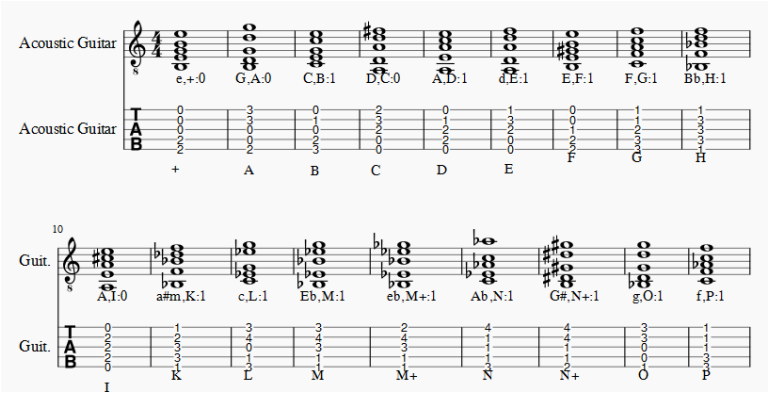

Figura 19: Traducción del diccionario de acordes de Sanz a notación actual.

- Gallardas 1
  DM DM | AM AM | DM GM | AM AM | DM DM | GM AM | DM
- Gallardas 2
  Dm Dm | Dm FM | CM Dm | AM AM | AM Dm | Gm AM | Dm
- Gallardas 3
  AM EM | AM DM | EM EM | EM AM | DM EM | AM
- Villano 1
  DM DM | DM GM | DM DM | DM AM | DM DM
- Villano 2
  AM AM | AM DM | AM AM | AM EM | AM

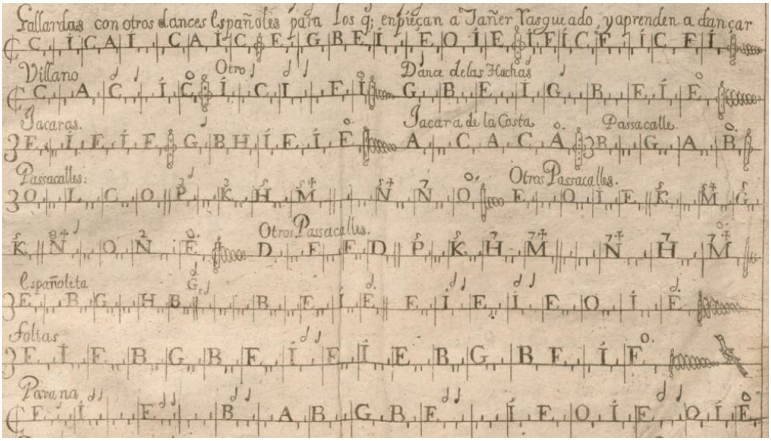

Figura 20: Lista de progresiones posibles de acuerdo al abecedario italiano en el tratado de Sanz.

- Danza de las hachas
  FM | CM | Dm | AM | FM | CM | Dm AM | Dm
- Danza de las hachas 2
  FM FM | CM CM | Dm Dm | AM AM | FM FM | CM CM | Dm AM | Dm
- Jácaras
  Dm | Dm AM Dm AM Dm
- Jácaras 2
  FM FM | CM B-M | AM Dm AM Dm
- Jácara de la costa
  GM | GM DM GM DM GM

- Pasacalle
  CM | CM CM | FM FM | GM GM | CM
- Pasacalles 1
  Gm | Gm | Cm | DM | Gm
- Pasacalles 2
  Gm | Gm | Cm | DM | Gm
- Pasacalles 3
  Gm | Gm | Dm | DM | Gm
- Otros pasacalles
  Dm | Dm | Gm | AM | Dm
- Otros pasacalles 1
  Dm Dm | Dm Dm | Gm Gm | AM AM | Dm
- Otros pasacalles 2
  Dm | Dm | Gm | AM | Dm
- Otros pasacalles 3
  E-other | E-other | Gm | AM | Dm
- Otros pasacalles 4
  Dm | Dm | Gm | AM | Dm
- Otros pasacalles 5
  Am | Am | Dm | Dm | Am
- Otros pasacalles 6
  Am | Am | Dm | EM | Am
- Otros pasacalles 7
  Am | Am | Dm | EM | Am

- Españoleta
  Dm | Dm | FM | FM | B-M | CM
- Españoleta 2
  FM | FM | CM | CM | Dm | AM | Dm
- Españoleta 3
  Dm | AM | Dm | AM | Dm | Gm | AM | Dm
- Folías
  Dm AM | Dm | CM | FM | CM | Dm | AM | Dm | AM |
  Dm | CM | FM | CM Dm | AM | Dm
- Pavana
  Dm | AM | AM | Dm | Dm | CM | GM | CM | FM CM |
  Dm | Dm | AM Dm | Gm AM | Dm | Gm AM | Dm
- Rugero*
  DM | DM | GM | DM

TÉCNICAS UTILIZADAS

Se han probado varias técnicas de clasificación con distintos grados de éxito. Como primer intento de clasificación de acordes se optó por usar funciones de lógica difusa, como en el Capítulo 3, para clasificar entre un diccionario de acordes. Donde tomamos una lista de notas $X = \{x_i\}$ y la comparamos con cada uno de los acordes A del diccionario.

De acuerdo a lo desarrollado en el capítulo del módulo de etiquetado de acordes, tenemos una función de pertenencia A para cada acorde en el diccionario, por lo que para encontrar

la clasificación óptima debemos encontrar el máximo de la lista.

$$L_{\text{máx}}(X) = \underset{A \in Dict}{\text{máx}} \{\mu_A(X)\} \tag{13}$$

Cabe notar que con esta definición puede haber más de un máximo, lo cual resulta en un problema, ya que no obtenemos una clasificación tan fina como desearíamos. Por lo anterior, estas funciones han tenido resultados bastante limitados en nuestras aplicaciones y análisis.

Sin embargo, si utilizamos una metodología de reglas como la que menciona el tratado de Gaspar Sanz, podemos utilizar el sistema de Diccionario y progresiones para codificar un sistema de reglas que nos ayude a clasificar de acuerdo al estilo que necesitemos. A continuación mencionamos cómo se implementaron.

MATRIZ DE CORRELACIÓN DE ACORDES

Una de las razones por la que el enfoque difuso no ha funcionado de manera adecuada, es que estamos representando un acorde como un conjunto de números. Es decir, hay veces que dos notas con notación distinta representan al mismo sonido, por ejemplo, en una afinación igualmente temperada, las notas *C#* y *Db* de la octava central, ambas representan la nota número 61. Sin embargo, al simplificar de esta manera, perdemos información importante ya que la forma en que está escrita la nota, nos reduce el número de acordes a los que puede pertenecer. Es importante saber si es un *C#* o un *Db*.

Por lo anterior decidimos implementar un nuevo sistema donde en lugar de clasificar los acordes por la cantidad de notas que tienen en común con un diccionario, creamos una representación matricial de un acorde, donde la activación de notas toma en cuenta la notación y no sólo el número, podemos ver un ejemplo de esto en la Figura 21. Creamos un diccionario de acordes con estas características y para buscar la etiqueta de acorde óptima, hacemos un análisis de correlación y asignamos la que tenga etiqueta máxima. Para realizar este análisis necesitamos traducir la notación antigua de Sanz a la notación moderna de la guitarra y de este modo crear las matrices perfiles del acordes.

Acorde (F, Ab, C)							
	C	D	E	F	G	A	B
Natural	1	0	0	1	0	0	0
#	0	0	0	0	0	0	0
##	0	0	0	0	0	0	0
b	0	0	0	0	0	1	0
bb	0	0	0	0	0	0	0

Figura 21: Ejemplo de matriz de activación de acorde (F, Ab, C).

Una vez que tenemos la matriz, perfil, de cada acorde, podemos utilizarla para hacer un análisis de correlación con los segmentos de una pieza musical. Para esto, creamos una matriz de activación de acuerdo a las notas que aparecen en ese segmento y la comparamos con cada elemento del diccionario por medio de correlación de Pearson, elegiremos para etiqueta el elemento del acorde con mayor valor de correlación. En la siguiente sección se muestra un ejemplo de esta técnica.

Análisis de piezas

El tratado de Sanz incluye ejemplos musicales que utilizan su diccionario de acordes y las progresiones sugeridas para cada tipo de pieza. Para probar el sistema de matrices de correlación y diccionario de acordes para clasificación, utilizaremos la pieza *Españoleta*, incluida en el tratado.

En la Figura 22, se muestra el primer compás y su matriz, perfil, de activación de notas. Esa matriz se compara con cada una de las del diccionario de acordes de Sanz y se elige como etiqueta la que tenga valor de correlación mayor, en este caso, la triada Re menor. Por su parte, en la Figura 23, se muestra el resultado de este tipo de análisis en cada uno de los compases.

Esta metodología de análisis muestra mucho potencial ya que toma en cuenta los valores de las notas, en lugar de solamente su reducción numérica, lo cual nos permite una clasificación mucho más precisa y fina. Aun falta validar y probarlo con un gran número de piezas del tratado de Sanz, sin embargo, se espera que calibrando los valores correctamente, se obtenga una clasificación adecuada y robusta.

Compas 1

	C	D	E	F	G	A	B
Natural	0	1	0.33	0.5	0.166	0	0
#	0	0	0	0	0	0	0
##	0	0	0	0	0	0	0
b	0	0	0	0	0	0	0
bb	0	0	0	0	0	0	0

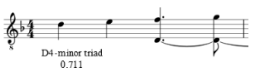

Figura 22: Ejemplo de matriz de activación en el primer compás de la partitura.

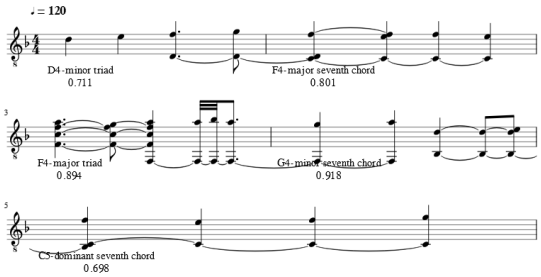

Figura 23: Resultado de análisis la Españoleta de Sanz.

EXTRACCIÓN DE DESCRIPTORES

Para realizar una clasificación de tipos de pieza de acuerdo a ejemplos de las piezas de Gaspar Sanz, realizaremos los siguientes pasos:

- Agrupar las piezas de acuerdo a las clases de títulos.
- Obtener los descriptores de las piezas.
- Tomar esto como centros tonales.
- Comparar otras piezas a ver a cuál se parecen más.
- Como un paso extra, se puede verificar con las reglas de progresiones armónicas dispuestas por Gaspar Sanz.

El sistema de diccionario de acordes presentado, que compara las matrices, perfiles, de los segmentos con el diccionario de acordes muestra mucho potencial para llegar a clasificaciones correctas de acordes.

Cabe notar, que esta metodología puede aplicarse a otros tipos de piezas, sólo se necesita adaptar el diccionario de acordes y las reglas de progresiones de armonías.

7

IMPLEMENTACIÓN

En este capítulo se describen las cuestiones técnicas de la implementación del sistema modular en un servidor. Consiste principalmente de tres pasos. El primero, en establecer las direcciones, o rutas, de acceso del servidor. El segundo en la forma en que la interfaz gráfica accede a dichas rutas. El tercero, en describir cómo el módulo de cálculo realiza las operaciones y genera los resultados.

El sistema modular que de este trabajo se organiza en la arquitectura llamada de *microservicios*. Es decir, cada módulo y parte del programa tiene una responsabilidad bien definida y puede comunicarse con las demás partes por medio de rutas, las cuales consisten en una secuencia de caracteres que indican la operación que deseamos realizar. Estas funcionan de la misma forma que las direcciones de internet, donde se especifica el servicio que deseamos y se agrega la dirección y especificación exacta de cómo acceder a ellos.

Supongamos que nuestro servidor tiene rutas para cada módulo, y que la ruta para realizar el cálculo de etiquetado de acordes se llama *acordes*, y deseamos aplicarlo a la pieza *Minuet.xml*, la ruta podría ser la siguiente:

```
servidor/acordes?pieza=Minuet.xml
```

Donde separamos por diagonales las operaciones que deseamos realizar y pasamos parámetros al final después del signo de pregunta. En forma general sería:

```
servidor/operacion?p1=valor1&p2=valor2
```

Como nota de programación, en general en los lenguajes de programación se trata de evitar el uso de tíldes en las variables y rutas. Por esta razón, omitimos la tilde en la palabra operación del ejemplo pasado.

Al diseño de las rutas así como las conexiones posibles del servidor se le conoce como *API*, *Application Programming Interface*, Interfaz de programación de aplicación. Es importante que un servidor tenga bien definidas las rutas posibles con las que puede trabajar dicha *API*. En este trabajo, creamos una ruta por cada módulo, de modo que tenemos rutas para el etiquetado de acordes, para la identificación de estructuras de repetición, para la búsqueda de secciones de acuerdo a diferentes criterios, así como para la clasificación de archivos, como en el caso del tratado de guitarra barroca de Gaspar Sanz.

El segundo punto importante es cómo el cliente se comunica con el servidor. El cliente, en este caso es el usuario que se conecta por medio de un navegador de internet como Google Chrome o Mozilla Firefox, ya sea desde una computadora personal, o desde un dispositivo móvil. Dicho cliente consiste en una aplicación web, a donde el usuario accede por medio de una dirección donde esté montado el servidor, por ejemplo

```
https://www.servidorprueba.edu
```

Al entrar a esa aplicación web, al usuario se le muestra una interfaz de usuario donde ya no es necesario que él conozca las rutas con las que debe comunicarse con el servidor, ya que esto se lleva a cabo automáticamente cuando el presiona los botones y llena los campos de búsqueda con los que interactúa.

El cliente no realiza ningún cálculo ni operación. Su responsabilidad principal es mostrar al usuario imágenes, partituras, tablas, formas, botones, etc; así como encargarse de enviar las solicitudes de rutas al servidor cuando el usuario presione los botones o envíe una solicitud. La información que se pasa entre cliente y servidor se encuentra en formato *JSON*, que contiene la toda la información con los resultados de los cálculos de los módulos. En estos archivos *JSON* se incluyen las partituras, imágenes, tablas y datos resultantes de las operaciones que requiera el usuario. En el siguiente fragmento de código se muestra un ejemplo de cómo es un archivo *JSON* que contiene los resultados de los módulos. En ese archivo json se muestran etiquetas y sus valores resultantes. Por ejemplo, el nombre del archivo, la dirección donde se encuentra en el servidor y una lista de acordes analizados.

```
1     {
2         "file_name": "44-Jiga",
3         "path": "files/44-Jiga.musicxml",
4         "harmony": {
5           "list_chord_str": [
6             "A-Minor Third",
7             "C-Major Third",
8             "G-Perfect Fourth",
9             "F-major triad",
10            "G-Major Third",
11            "A-Perfect Fifth",
```

```
            "E-Perfect Fourth",
            "G-Major Third",
            "G#-Minor Third",
            "A-Minor Third",
            "C-Major Third",
            "G#-Minor Third"
          ]
        }
      }
```

Código 7.1: Ejemplo de *JSON* con descriptores.

En este fragmento de código se observa cómo *Python* maneja las distintas rutas para algunos de los módulos.

```
file_path = 'files/'

args = sys.argv[1:]
operation = args[0]
if len(args) > 1:
    file_name = args[1]
    file_route = file_path+file_name

if operation == "jsonAll":
    ex.run_json_all(file_path, "musicxml")
    print("Finished json all")

elif operation == "classify_names":
    svg_out = ex.run_classify_names(file_path)
    print(svg_out)

elif operation == "dtw":
    obj_out = ex.run_dtw_api(file_route)
    print(obj_out)

elif operation == "chordify":
    xml_string = ex.run_chordify_api(file_route)
    print(xml_string)
```

```
24
25 elif operation == "stats":
26     json_out = ex.run_stats_api(file_route)
27     print(json_out)
```

Código 7.2: Rutas desde *Python*

CLIENTE REACTJS

Para la creación del cliente se eligió un *framework*, entorno de trabajo, que fuera muy versátil, actual y que permitiera una alta interacción de forma cómoda con el usuario. Elegimos *ReactJS*, el cual es de código abierto y está basado en JavaScript. Este *framework* fue desarrollado por la empresa Facebook y es uno de los estándares en la industria. Se organiza por componentes donde cada uno de ellos tiene una responsabilidad sencilla. Por ejemplo, un componente puede encargarse de mostrar una partitura, otro de mostrar tablas, así como mostrar listas dinámicas de archivos. En nuestro caso creamos una serie de componentes genéricos para mostrar información, así como componentes más especializados para intercambiar información con el servidor.

MOTOR DE BÚSQUEDA

Ahora, el tercer punto, después del servidor y el cliente, es el motor de búsqueda. En realidad es en este punto donde se realizan todos los cálculos de Inteligencia Artificial y Minería de Datos, podríamos decir que es donde está lo realmente interesante del trabajo. Pero es importante que los otros dos elementos existan para ofrecer una experiencia amigable al

usuario. Desde el principio, una de nuestras motivaciones más importantes es acercar al musicólogo a estas herramientas, y es necesario más que el motor de cálculo para hacer esto.

Cuando el cliente hace una petición al servidor, este se encarga de encontrar el módulo de *Python* que debe correr, asi como de pasar los archivos *XML* sobe los que se aplicará el cálculo requerido. Para hacer esto, el servidor tiene acceso al motor de búsqueda por medio de una biblioteca llamada *spawn*. Con esta podemos filtrar las rutas que requiera al servidor y llamar al módulo de *Python* correspondiente. En el siguiente fragmento de código se muestra cómo se corre un archivo de *Python*. Los resultados del módulo de cálculo se guardan en la variable `data` para después ser regresados al cliente en forma de partitura con el comando `res.send`

```
1   const pyProg = spawn('python', [pythonPath, operation,
        fileScore]);
2   pyProg.stdout.on('data', data => {
3     switch (operation) {
4       case "chordify":
5       case "score":
6         console.log("call verovio");
7         vr.xmlToSVg(data.toString(), verov => res.send({
              svg: verov }));
8         break;
9       case "dtw":
10      case "stats":
11        res.send({ svg: data.toString() });
12        break;
13      default:
14        res.send({ svg: data.toString() });
15        break;
16    }
```

```
17    });
```

Código 7.3: Correr *Python* desde node.

Como se mencionó anteriormente, el sistema está montado en un servicio de plataforma en la nube. Existen diferentes opciones y proveedores, entre los más comúnes actualmente tenemos Heroku, Google Cloud y Amazon Web Services. Cada uno de ellos tiene costos y funcionalidades ligeramente diferentes, pero comparten que en todos uno puede montar su servidor en línea de manera relativamente sencilla. Incluso existen paquetes en los que el servicio es gratuito dentro de un límite de ancho de banda y procesamiento. Recordemos que, para usuarios más avanzados, es posible montar el servidor localmente. La ventaja de utilizar un servicio en la nube es que no es necesario dar mantenimiento al servidor y muchas de las cuestiones técnicas de seguridad están cubiertas de antemano.

A pesar de que existen muchas opciones para montar nuestro servicio en línea o de forma local. Es muy importante recordar que estos servidores y servicios son prácticamente intercambiables, esto se debe a que al usuario final le daremos simplemente una dirección de internet, que en un momento dado puede estar apuntando a un servidor en Google Cloud, y si más adelante decidimos cambiar de servicio por ejemplo a Amazon Web Services, basta con que cambiemos a donde apunta la dirección. De esta forma, el usuario no sabe, ni le interesa realmente donde está localizado su servicio, siempre y cuando tenga la dirección correcta.

Una realidad incómoda en el desarrollo tecnológico, es que los desarrollos se vuelven obsoletos a una velocidad vertiginosa. Basta con que revisemos libros de programación de

hace sólo algunos años para darnos cuenta cómo muchas de las cosas que incluye ya no son validas. Por esta razón, lo más importante del desarrollo tecnológico es la organización de ideas y conceptos. En nuestro caso, el manejo de módulos. Esto nos ayuda a que aunque en algunos años, cuando las librerías y *frameworks* que aqui mencionamos ya no se usen, sea fácil de actualizar a las nuevas maneras. Esta es la razón por la cual hacemos énfasis en la modularidad. Por ejemplo, si en algunos años *ReactJS*, nuestro ambiente de desarrollo para la interfaz gráfica, ya no es la opción más viable, podemos cambiarlo por alguna otro que sea más actual sin tocar siquiera el módulo de cálculo y servidor. Esta modularidad permite que el proyecto se pueda mantener de forma más eficiente por mucho más tiempo.

INSTRUCCIONES PARA MONTAR SERVIDOR

Esta sección es solo para el caso de los usuarios que quieran crear una versión local del sistema. Para montar el servidor en un servicio en línea como los que hemos mencionado es necesario tener los siguientes ambientes y programas instalados:

- Node.js 12.10 o superior.
- git 2.7 o superior.
- Python 3.6 o superior.

El projecto se encuentra en un repositorio git de donde se puede clonar.

```
git clone direccionpordefinir
```

Una vez dentro es necesario instalar todas las dependencias, esto se hace de manera automática desde la línea de comando en el directorio del proyecto se debe ejecutar:

```
npm install
```

Esto se encargará de instalar todos las dependencias necesarias. Si todo salió correctamente, podemos comenzar el servidor con el código:

```
npm start
```

Este proceso se debe realizar tanto con el cliente como el servidor, y de esta manera se abrirá localmente un navegador web que nos permite utilizar el sistema.

CREACIÓN DE NUEVOS MÓDULOS

Uno de las ideas más importantes de este trabajo y su modularidad es que se pueda extender y crecer con nuevos módulos creados por otros investigadores. La intención es crear un sistema robusto que sirva en primer lugar a la UNAM, como herramienta de investigación, así como a otros investigadores. Para que los nuevos módulos sean compatibles con el sistema, es necesario que los resultados de los cálculos así como los parámetros de entrada estén en formato *JSON*. De esta forma, se puede agregar fácilmente al sistema simplemente creando y habilitando una nueva ruta de acceso.

```
servidor/modulonuevo?pieza="Preludio1.xml"
```

Al colocar este nuevo archivo de *Python* en la carpeta de módulos, el servidor automáticamente revisa que hay un

módulo nuevo y el usuario podrá verlo dentro de las opciones de análisis.

Debe existir un control de calidad para los módulos nuevos que surjan, ya que es necesario revisar que no consuma demasiados recursos, ni tenga errores. Este nivel de mantenimiento es parte del trabajo futuro que debe realizarse por quién sea el encargado de mantener el servidor.

En este capítulo se mostraron algunas de las implicaciones y requerimientos técnicos para la implementación del servicio modular en una arquitectura orientada a servicios. El código de todo el sistema, se incluye íntegro en los anexos digitales, sin embargo, más importante que las instrucciones exactas para correr el sistema, es el diseño y organización de las ideas de los módulos y las rutas del *API*. En pocos años, este sistema será obsoleto con el código actual, sin embargo, se han seguido las mejores prácticas de desarrollo de software actuales para hacer que el sistema sea "A prueba del futuro", es decir, que requiera poco mantenimiento y adaptación para seguir funcionando. Esto con la idea que el sistema pueda crecer orgánicamente para contribuir al estudio de la música con nuevas técnicas. Para ver el código completo, revisar los anexos digitales, así mismo, se pueden buscar actualizaciones y la continuación del sistema en la página `holomorfo.com`

CONCLUSIONES

A lo largo de este trabajo se han presentado diversas técnicas y metodologías para aplicar algoritmos de Inteligencia Artificial y Minería de Datos al análisis musical de archivos de representación simbólica, principalmente *MusicXML*. Se creó un sistema modular de análisis donde cada módulo aborda una técnica en específico, entre ellos están: extracción de descriptores, etiquetado de acordes, identificación de estructuras de repetición, módulo de búsquedas y clasificación de archivos por medio de similitudes. Estos módulos se implementaron en un arquitectura de cliente/servidor por medio de *Python* con las librerías *music21, tensorflow, keras, scikit-learn* y *NodeJS* para el servidor, así como *ReactJS* para el cliente o usuario. El sistema está montado en un servidor en línea donde se puede utilizar sin necesidad de que el usuario instale nada más que un navegador de internet actualizado.

Una de las contribuciones más interesantes de este trabajo es la implementación de los algoritmos en una arquitectura orientada a servicios. A diferencia de crear una aplicación que el usuario instale en su computadora, este se conecta a un servicio en línea al cual puede acceder desde una computadora personal, teléfono móvil o tableta. Esto sirve para acercar a los músicos a las técnicas de Inteligencia Artificial y Minería de Datos sin necesidad de que tengan que aprender a programar o instalar programas.

Los algoritmos que aplicamos para cada módulo ofrecen un complemento interesante al análisis musical que se podría hacer manualmente. Esto se debe a que muchas tareas repetitivas de comparación y clasificación que a mano llevarían mucho tiempo en completarse, por medio de un algorítmo tarda unos cuantos segundos. Esto se vio al utilizar las redes neuronales y lógica difusa para el etiquetado de acordes, así como para la clasificación de archivos por similitud. Sin embargo su uso conlleva también ciertas limitaciones. En el caso de las redes neuronales, es necesario tener un gran número de ejemplos para entrenarlas correctamente, lo cual no siempre es el caso. Así mismo, los clasificadores siempre tienen un grado de error, lo cual requiere un humano que revise y verifique los resultados para calibrar el sistema en puntos donde haya ambigüedad.

Un ejemplo de las limitaciones que pueden surgir en este tipo de análisis asistido por computadora es en el módulo de etiquetado de acordes con lógica difusa. En un principio, parecía una buena idea asignar funciones de pertenencia para decidir si un conjunto de notas es o no, un cierto acorde. Como se mostró en el trabajo, este resultado no fue óptimo, ya que las utilidad de las funciones de pertenencia dependía de la forma en que se presentaba la marcha armónica en la pieza.

Las limitaciones de cada algoritmo en este trabajo funcionan a nuestro favor. Esto es debido a que, en este trabajo el objetivo principal no es encontrar los algorítmos más eficientes, sino acercar al músico a estas técnicas y que él pueda decidir si en efecto le aportan o no a su análisis. Cuando se encontró que el análisis por medio de lógica difusa no

obtuvo los mejores resultados, se pasó a realizar el análisis de acordes por medio de redes neuronales. De esta forma, la intención del trabajo es crear un sistema donde el analista musical pueda experimenter y probar con algoritmos que de otra manera serían inaccesibles dado el grado de dificultad que involucra programarlos.

El tener un sistema modular para implementar estos algoritmos permite que el sistema tenga vida *más allá de la tesis*, es decir, dada las especificaciones de creación de módulos que se mencionan en el último capítulo, es posible agregar nuevos módulos de modo que otros investigadores puedan implementar nuevos algoritmos y tomar ventaja del sistema de organización y bases de datos que se implementó en este trabajo. Esto le da mucha flexibilidad a la forma en que se pueden encadenar los distintos tipos de algoritmos de análisis.

Los resultados de análisis de cada módulo que presentamos en este trabajo muestran un ejemplo de las posibilidades. Como trabajo futuro es necesario probar el sistema en un ambiente real, con el trabajo de musicólogos, analistas y estudiantes. Esto es de gran ayuda por que servirá de guía para llevar un registro de los módulos más útiles, así como obtener retroalimentación para la creación de nuevos módulos.

A lo largo del trabajo, se ha defendido la idea de trabajar con archivos de representación simbólica para extraer información y realizar los análisis. Estamos conscientes que gran parte de las partituras a las que tenemos acceso están en formato de imagen, o incluso en papel. Esto parecería una limitación, ya que para utilizar el sistema necesitamos tenerlos en formato *XML*, sin embargo, la digitalización de partituras y conversión a representación simbólica es un trabajo aparte.

En algunos años, cuando esa tecnología esté más madura, sus resultados pueden complementarse con los nuestros para incluir partituras en formato de imagen al sistema. Por lo pronto, es óptimo trabajar con archivos *XML*.

El estado actual de la tecnología, con herramientas como *Python, TensorFlow, music21, JavaScript, ReactJS, ExpressJS* nos coloca en un punto particularmente fértil para su aplicación en la música. Como se mostró a lo largo del trabajo, existen muchos puntos de aplicación que aun quedan por explorar. Es necesario continuar el trabajo en conjunto tanto matemáticos, programadores y musicólogos para crear nuevas aplicaciones que ayuden al trabajo del análisis musical.

Es la intención de este trabajo que el sistema sea una representación de las colaboraciones multidisciplinarias que se llevan a cabo en esta universidad y sirva como referencia para otros investigadores que deseen continuar esta línea de trabajo para acercar la tecnología más efectiva y actual, a la investigación musical.

BIBLIOGRAFÍA

International Music Score Library Project (s.f.). http://imslp.org/. Visitado: 2018-01-20.

Kunst der fuge (s.f.). http://www.kunstderfuge.com/. Visitado: 2018-01-20.

Brilliant (2019). *K means clustering*. URL: https://brilliant.org/wiki/k-means-clustering/ (visitado 13-11-2019).

Management, Ophtalmology (2019). *Imagen, The problem: Surgery or Not*. URL: https://www.ophthalmologymanagement.//com/content/archive/2010/August/Supplements/B_L_454/OMD-August_A12_Fig01.jpg (visitado 13-11-2019).

Medium (2019). *Imagen, Logistic regression.* URL: https://cdn-images-1.medium.com/%20//%20max/1428/1*KRhpHnucyX9Y5PMdjGvVFA.png (visitado 13-11-2019).

Abdallah, Samer y col. (2017). «The digital music lab: A big data infrastructure for digital musicology». En: *Journal on Computing and Cultural Heritage (JOCCH)* 10.1, pág. 2.

Armentano, Marcelo G, Walter A De Noni y Hernán F Cardoso (2017). «Genre classification of symbolic pieces of music». En: *Journal of Intelligent Information Systems* 48.3, págs. 579-599.

Bañuelos, Cristian y Felipe Orduña (2017). «Dynamic Time Warping for Automatic Musical Form Identification in Symbolical Music Files». En: *International Conferen-*

ce on Mathematics and Computation in Music, Springer, págs. 253-258.

Géron, Aurélien (2017). *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. .º'Reilly Media, Inc."

Gilbert, John (2017). «Chopin Prelude in C Minor». En: *NYU.edu*.

Martínez, Brian y Vicente Liern (2017). «A Fuzzy-Clustering Based Approach for Measuring Similarity Between Melodies». En: *International Conference on Mathematics and Computation in Music*. Springer, págs. 279-290.

Pons, Jordi y col. (2017). «Timbre Analysis of Music Audio Signals with Convolutional Neural Networks». En: *arXiv preprint arXiv:1703.06697*.

Yann LeCun Corinna Cortes, Christopher J.C. (2017). *THE MNIST DATABASE of handwritten digits*. URL: http://yann.lecun.com/exdb/mnist/ (visitado 07-11-2017).

Corrêa, Débora C y Francisco Ap Rodrigues (2016). «A survey on symbolic data-based music genre classification». En: *Expert Systems with Applications* 60, págs. 190-210.

Bañuelos, Cristian (2015). «Universo Tonal». En: *Tesis Unam*.

Bengio, Yoshua, Ian J Goodfellow y Aaron Courville (2015). «Deep learning». En: *Nature* 521, págs. 436-444.

Lamm (2015). «Music XML Analyzer». En: *GitHub repository* 1.1, págs. 5520-5525.

Müller, Meinard (2015). *Fundamentals of Music Processing: Audio, Analysis, Algorithms, Applications*. Springer.

Ng, Andrew (2015). *Under and overfit*. URL: https://i.stack.imgur.com/t0zit.png (visitado 13-11-2019).

Nielsen, Michael A. (2015). *Neural networks and Deep Learning, Determination Press*. (Visitado 13-11-2019).

Park, Taejin y Taejin Lee (2015). «Musical instrument sound classification with deep convolutional neural network using feature fusion approach». En: *arXiv preprint arXiv:1512.07370*.

RNN (2015). *Imagen, RNN*. URL: https://i.stack.imgur.com/Hl2H6.png (visitado 13-11-2019).

Zhou, Xinquan y Alexander Lerch (2015). «Chord detection using deep learning». En: *Proceedings of the 16th ISMIR Conference*. Vol. 53.

Antila, Christopher y Julie Cumming (2014). «The VIS Framework: Analyzing Counterpoint in Large Datasets.» En: *ISMIR*, págs. 71-76.

Chew, Elaine (2014). «Mathematical and Computational Modeling of Tonality». En: *AMC* 10, pág. 12.

Herlands, William y col. (2014). «A Machine Learning Approach to Musically Meaningful Homogeneous Style Classification.» En: *AAAI*, págs. 276-282.

Kirschenbaum, Matthew (2014). «What Is "Digital Humanities," and Why Are They Saying Such Terrible Things about It?» En: *differences* 25.1, págs. 46-63.

Lavrenko, Viktor (2014). *Under and over fitting examples*. URL: https://i.ytimg.com/vi/dBLZg-RqoLg/maxresdefault.jpg (visitado 13-11-2019).

Pesek, Matevz, Aleš Leonardis y Matija Marolt (2014). «A compositional hierarchical model for music information retrieval». En: *Proceedings of the International Conference on Music Information Retrieval (ISMIR), Taipei*.

Sarroff, Andy M y Michael A Casey (2014). «Musical audio synthesis using autoencoding neural nets». En: *ICMC*.

Sturm, Bob L (2014). «The state of the art ten years after a state of the art: Future research in music information retrieval». En: *Journal of New Music Research* 43.2, págs. 147-172.

Kaminskas, Marius y Francesco Ricci (2012a). «Contextual music information retrieval and recommendation: State of the art and challenges». En: *Computer Science Review* 6.2, págs. 89-119.

– (2012b). «Contextual music information retrieval and recommendation: State of the art and challenges». En: *Computer Science Review* 6.2, págs. 89-119.

Lamere, P (2012). «The Infinite Jukebox». En: *url: www. infinitejuke. com/(Visitado 24 May, 2016)*.

Li, Tao, Mitsunori Ogihara y George Tzanetakis (2011). *Music data mining*. CRC Press.

Lin, Cheng-Jian y Chun-Cheng Peng (mayo de 2011). «Chord recognition using neural networks based on particle swarm optimization». en. En: *Cybernetics and Systems* 42.4, págs. 264-282.

Nielsen, Michael A. (2011). *Neural Networks and Deep Learning*. Determination Press.

Tymoczko, Dmitri (2011). *A geometry of music: harmony and counterpoint in the extended common practice*. Oxford University Press.

Viro, Vladimir (2011). «Peachnote: Music Score Search and Analysis Platform.» En: *ISMIR*, págs. 359-362.

Anagnostopoulou, Christina y Chantal Buteau (2010). «Can computational music analysis be both musical and computational?» En: *Journal of Mathematics and Music* 4.2, págs. 75-83.

Cuthbert, Michael Scott y Christopher Ariza (2010). *Music21 A toolkit for computer aided musicology*. ISMIR.

McKay, Cory (2010). *Automatic music classification with jMIR*. McGill University.

Toussaint, Godfried (2010). «Computational geometric aspects of rhythm, melody, and voice-leading». En: *Computational geometry: Theory and applications* 43.1, págs. 2-22.

Nichols, Eric y col. (2009). «Relationships between lyrics and melody in popular music». En: *ISMIR*.

Nielsen, Frank y R Nock (2009). «Emerging trends in visual computing». En: *Lecture Notes in Computer Science* 6.

Tymoczko, Dmitri (2009). «Three conceptions of musical distance». En: *Mathematics and computation in music*. Springer, págs. 258-272.

McKay, Cory e Ichiro Fujinaga (2008). «Combining Features Extracted from Audio, Symbolic and Cultural Sources.» En: *ISMIR*, págs. 597-602.

Rho, Seungmin y col. (2008). «MUSEMBLE: A novel music retrieval system with automatic voice query transcription and reformulation». En: *Journal of Systems and Software* 81.7, págs. 1065-1080.

Temperley, David y Elizabeth West Marvin (2008). «Pitch-class distribution and the identification of key». En: *Music Perception: An Interdisciplinary Journal* 25.3, págs. 193-212.

Xiao, Linxing y col. (2008). «Using Statistic Model to Capture the Association between Timbre and Perceived Tempo.» En: *ISMIR*, págs. 659-662.

Berman, Tamar (2007). «In Search of Harmony: A Constraint-Based Approach to the Task of Harmony Retrieval». En: *Journal of New Music Research* 36.4, págs. 301-312.

Cha, Sung-Hyuk (2007). «Comprehensive survey on distance/similarity measures between probability density functions». En: *Internationl Journal of Mathematical Modelsin Applied Sciences* 1.4, pág. 1. (Visitado 27-07-2016).

Hartmann, Knut y col. (2007). «Interactive data mining and machine learning techniques for musicology». En: *3rd Conference on Interdisciplinary Musicology (CIM07), Tallinn, Estonia*. Vol. 15, pág. 19.

Müller, Meinard (2007). *Information retrieval for music and motion*. Vol. 2. Springer.

Salvador, Stan y Philip Chan (2007). «Toward accurate dynamic time warping in linear time and space». En: *Intelligent Data Analysis* 11.5, págs. 561-580.

Schubert, Emery y Catherine Stevens (2006). «The effect of implied harmony, contour and musical expertise on judgments of similarity of familiar melodies». En: *Journal of New Music Research* 35.2, págs. 161-174.

Szeto, Wai Man y Man Hon Wong (2006). «A graph-theoretical approach for pattern matching in post-tonal music analysis». En: *Journal of New Music Research* 35.4, págs. 307-321.

Tan, Pang-Ning, Michael Steinbach, Vipin Kumar y col. (2006). *Introduction to data mining*. Vol. 1. Pearson Addison Wesley Boston.

Araokar, Shashank (2005). «Visual character recognition using artificial neural networks». En: *arXiv preprint cs/0505016*.

Kuo, Fang-Fei y col. (2005). «Emotion-based music recommendation by association discovery from film music». En: *Proceedings of the 13th annual ACM international conference on Multimedia*. ACM, págs. 507-510.

Perera, MAP Neshadha y SR Kodithuwakku (2005). «Music Chord Recognition Using Artificial Neural Networks». En: *Proceedings of the International Conference on Information and Au-tomation*, págs. 304-308.

Pérez-Sancho, Carlos, José M Inesta y Jorge Calera-Rubio (2005). «Style recognition through statistical event models». En: *Journal of New Music Research* 34.4, págs. 331-339.

Cunningham, Stuart (2004). «Suitability of musicxml as a format for computer music notation and interchange». En: *Proceedings of IADIS Applied Computing 2004 International Conference, Lisbon, Portugal*.

Eerola, Tuomas y Petri Toiviainen (2004). «MIDI toolbox: MATLAB tools for music research». En: *University of Jyväskylä*.

Hörnel, Dominik (2004). «Chordnet: Learning and producing voice leading with neural networks and dynamic programming». En: *Journal of New Music Research* 33.4, págs. 387-397.

Knopke, Ian (2004). «Sound, Music and Textual Associations on the World Wide Web.» En: *ISMIR*.

Dubnov, Shlomo y col. (2003). «A system for computer music generation by learning and improvisation in a particular style». En: *IEEE Computer* 10.38.

Hirata, Keiji y Shu Matsuda (2003). «Interactive music summarization based on generative theory of tonal music». En: *Journal of New Music Research* 32.2, págs. 165-177.

Hu, Ning, Roger B Dannenberg y George Tzanetakis (2003). «Polyphonic audio matching and alignment for music retrieval». En: *Applications of Signal Processing to Audio and Acoustics, 2003 IEEE Workshop on.* IEEE, págs. 185-188.

Shan, Man-Kwan y Fang-Fei Kuo (2003). «Music style mining and classification by melody». En: *IEICE TRANSACTIONS on Information and Systems* 86.3, págs. 655-659.

Aucouturier, Jean-Julien, Francois Pachet y col. (2002). «Music similarity measures: What's the use?» En: *ISMIR*, págs. 13-17.

Byrd, Donald y Tim Crawford (2002). «Problems of music information retrieval in the real world». En: *Information Processing & Management* 38.2, págs. 249-272.

Cooper, Matthew L y Jonathan Foote (2002). «Automatic Music Summarization via Similarity Analysis.» En: *ISMIR*.

Mazzola, Guerino y col. (2002). *The topos of music*. Birkhäuser, Basel.

Pardo, Bryan y William P Birmingham (2002). «Algorithms for chordal analysis». En: *Computer Music Journal* 26.2, págs. 27-49.

Pachet, François, Gert Westermann y Damien Laigre (2001). «Musical data mining for electronic music distribution». En: *Web Delivering of Music, 2001. Proceedings. First International Conference on*. IEEE, págs. 101-106.

Beran, Jan, Guerino Mazzola y col. (1999). «Analyzing Musical Structure and Performance—A Statistical Approach». En: *Statistical Science* 14.1, págs. 47-79.

Temperley, David (1999). «What's key for key? The Krumhansl-Schmuckler key-finding algorithm reconsidered». En: *Music Perception: An Interdisciplinary Journal* 17.1, págs. 65-100. (Visitado 23-02-2016).

Anagnostopoulou, Christina y Gert Westermann (1997). «Classification in Music: A Computational Model for Paradigmatic Analysis.» En: *ICMC*.

Dannenberg, Roger B, Belinda Thom y David Watson (1997). «A machine learning approach to musical style recognition». En:

Lande, Tor Sverre y Arvid O Vollsnes (1994). «Object oriented music analysis». En: *Computers and the Humanities* 28.4-5, págs. 253-257.

Bel, Bernard y Bernard Vecchione (1993). «Computational musicology». En: *Computers and the Humanities* 27.1, págs. 1-5.

Erickson, Raymond (1968). «Music analysis and the computer». En: *Journal of Music Theory* 12.2, págs. 240-263.

Mugellini, Bruno (1964). *The Well Tempered Clavier*. PWM: Warsaw.

Theodor, Kullak (1882). *Nineteen Sonatas for the Piano*. Klavierwerke.

Sanz, Gaspar (1674). *Instrucción de música sobre la guitarra española y método de sus primeros rudimentos hasta tañerla con destreza.* Zaragoza: por los herederos de Diego Dormer.