Inteligencia Artificial en Educación Musical: Desarrollo de un Entorno de Aprendizaje Basado en Redes Neuronales Generativas

Doctorando: Ing. Carlos Adrián Arana

Director: Dr. Ing. Emilio Picasso

Facultad de Ingeniería
Universidad de Lomas de Zamora

Buenos Aires, Argentina
2022

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1 . Introducción

1.1 Conceptos Básicos

La formación tradicional del músico involucra el aprendizaje de conceptos básicos del lenguaje musical. Sobre los conceptos teóricos y las normas del discurso musical, tanto interpretativo como compositivo, no hay consenso en que sea necesaria una reformulación pedagógica. Los tratados y los diseños curriculares que abordan estos temas han sido desarrollados e implementados por siglos, llegando en los últimos años a desarrollos, tanto en música clásica o académica como en muchas áreas de la música popular, de una calidad pedagógica excelsa. A su vez esa metodología cuenta con un soporte bibliográfico y de acompañamiento didáctico de altísimo nivel, que ha probado su eficiencia y contundencia pedagógica a lo largo de siglos y formado a generaciones de músicos, tanto aficionados como profesionales.(Jorgensen, 2003)

Se denomina música popular a las expresiones artístico-musicales asociadas los folklores regionales y a las tradiciones transmitidas generacionalmente. El proceso de conformación identitaria de los pueblos por intermedio de sus músicas, y de las constantes interacciones a lo largo del tiempo -en particular en estos últimos tiempos donde la tecnología de más avances ha hecho que estos se divulgaran de manera exponencial- han traído aparejado la conformación y asentamiento de géneros y tradiciones musicales populares en las que se evidencian componentes estéticas, técnicas y de virtuosísimo muy refinadas.

Como en cualquier lenguaje, el musical no escapa al hecho de que sus matices y complejidades inherentes estén íntimamente relacionados y formados a partir de la interacción con pares en situaciones reales. Es bien sabido que para aprender un nuevo idioma es recomendable una estadía en una locación en la que este se hable. Es en la constante repetición de situaciones en las que se pone en práctica la

comunicación simbólica en la que reside la mayor dinámica de captación y asimilación de los conceptos intrínsecos del lenguaje en cuestión y de sus formas y vericuetos más difíciles y complejos de aprender. La música en general y la popular en particular no escapan a esta realidad por su condición de lenguaje, y está probada la efectividad de la repetición en sus desarrollos formales y estéticos (Margulis, 2003).

Es por ello que sería muy provechoso y bienvenido que a la ejercitación y los modelos clásicos de aprendizaje de los distintos tipos de lenguajes se le agreguen otros mecanismos y métodos de ejercitación que coadyuven y contribuyan a generar de forma reproducible, especialmente en el aula o en el espacio de estudio, estas situaciones de la vida real que fehacientemente se ha comprobado son las que nos permiten formar aprendices en la complejidad y en los variados matices que caracterizan al lenguaje en estudio.

1.2 Desarrollo de aptitudes en un lenguaje. Caso particular: lenguaje musical

A lo largo de los últimos siglos se han realizado avances sumamente importantes en variados aspectos formales de la pedagogía musical, tanto en lo referente a la formación y educación de músicos compositores como de intérpretes (Jorgensen, 2003). Las herramientas con las que cuenta el educador para fortalecer los conocimientos del aprendiz en cualquiera de estas tareas del saber musical se pueden agrupar en dos grandes conjuntos: uno de índole más formal que comprende al estudio de la historia, la estructura (los tipos de compás, la duración de sus notas, etc.), la notación, la armonía, las escalas, las técnicas de contrapunto, el análisis de obras y la improvisación mediante el usos de herramientas específicas; y otro que esencialmente comprende los diferentes tipos de elementos índole perceptiva y

estética, siendo la audioperceptiva su ejemplo más destacable. Otras áreas del saber musical componen este segundo conjunto y se asocian principalmente a la interpretación propia de cada ejecutante: ornamentación, fraseo, desarrollo de motivos, etc. Un análisis de la literatura de los modelos pedagógicos más desarrollados y probados a lo largo de siglos en la música clásica y de décadas en la música popular -y su respectivo material de soporte asociado- muestra que en su inmensa mayoría estos se especializan en disciplinas y saberes pertenecientes al primer conjunto. (Pike-Rowney, 2016). A la luz de lo anterior, y visto desde una óptica procedimental y didáctica, se puede considerar que contamos con una cantidad de conocimientos que son susceptibles de ser adquiridos por una serie concatenada, ordenada y muy bien formulada de pasos y procedimientos formales, y otros tipos de conocimientos y aptitudes, que precisan de una interacción de tipo vivencial y perceptivo, que podrían ser abordados con mayor profundidad en las metodologías pedagógicas desarrolladas e implementadas a la fecha.

Es en esta segunda línea donde se enfocará esta investigación, pues para la enseñanza y trasmisión de estas habilidades es necesaria la incorporación de ejercicios y actividades que simulen situaciones reales a las que se le presentarán al músico, que someterán a su protagonista a problemas que deberá ir solucionando y encaminando sobre la marcha, y que per-se poseen una impronta de novedad, variedad y sorpresa propias del devenir diario de cualquier actividad en las que seres humanos se comunican de manera fluida por intermedio de un lenguaje. Para este tipo de entrenamiento sólo existe la técnica de la abundante y variada exposición, y para que ello acontezca este trabajo sólo puede ser encarado mediante la presentación continua y progresiva de ejercicios y actividades de complejidad creciente. Pues bien, para cumplimentar este objetivo general y sus acciones en particular debemos contar con un generador de ejercicios y actividades especialmente creado a tal efecto.

1.3 Sobre los Modelos de Ejercitación Musical

La ejercitación asociada al aprendizaje de un lenguaje en cualquier tratado o metodología trajo aparejada históricamente una componente que funciona como limitante en el volumen de ejercicios a presentar, que es el formato del soporte sobre el que está asentada. En los métodos de enseñanza musical en formato papel -sean ellos libros, cuadernos u otros- los ejercicios son limitados por su tamaño, expresado comúnmente en dimensiones y cantidad de hojas. En los métodos o tratados musicales en formato audiovisual, como fueron los casetes/discos/VHS y DVD, existía la misma restricción física, en estos casos asociada a la las dimensiones de la cinta y a las características del soporte de impresión digital, en el caso que la información fuera sometida a una conversión analógico/digital.

Los ejercicios y las actividades que históricamente han complementado a los tratados y los cursos de enseñanza de música han sido siempre del tipo estático. Sus autores, desde Carulli en su clásico “Solfeo de los solfeos” (Carulli et al., 1891) pasando por Leavitt en su método de guitarra utilizado por generaciones y generaciones (Leavitt, 1966), y llegando al autor de esta tesis (Arana, 2008), han intentado ofrecer un contenido pedagógico original, y a la vez variado, que le imprima una personalidad propia a su publicación didáctica, medida en términos de diferenciación con otra que trate el mismo temario o estilo musical. A este trabajo de originalidad en el transcurso de este trabajo de investigación se lo denominará “novedoso”. No obstante que sus autores intenten imprimirle a su método o manual esta necesaria dosis de originalidad y novedad al incluir una cantidad mayor y más variada de actividades y ejercicios, se correría el riesgo de que el contenido empiece a tornarse repetitivo o previsible. (Ubovich, 2015).

La disponibilidad de recursos y tecnologías históricamente no favoreció al desarrollo de metodologías que se enfocarán en la formación de estos saberes asociados al lenguaje musical que hemos analizado, que sabemos exigen de una fluida y variada interacción con situaciones de la vida real. Con el fin de darle un marco de referencia definido al trabajo el presente de investigación nos enfocaremos en la audioperceptiva, subdisciplina musical en la que se evalúa la capacidad de reconocimiento y percepción musical del músico, que como hemos indicado es el área que define por excelencia al segundo conjunto de los componentes del lenguaje musical.

Diversos enfoques han tratado de subsanar esta situación restrictiva. El enfoque más abordado ha sido el desarrollo de software de educación musical. Para poder sortear las restricciones que impone el tipo de soporte físico, este tipo de metodología parece proponer una solución ideal: la generación de ejercicios mediante un algoritmo que permita presentar al estudiante una cantidad virtualmente ilimitada de ejercicios que simulen situaciones de la vida real. Para ello estas soluciones de software ("enlatados" como se los denomina en la industria) utilizan como base del motor generador de contenido musical las diferentes técnicas de composición algorítmica. Los algoritmos tradicionales desarrollados a este fin se dividen principalmente en las dos categorías siguientes:

a) *Basados en reglas*: la idea principal es utilizar reglas o gramáticas específicas para la composición algorítmica. García Salas et al. utilizan la lingüística y la gramática para crear música. La desventaja de este método es que hay que crear reglas diferentes según los distintos tipos de música (García Salas et al., 2011) , y las reglas suelen complejizarse de manera muchas veces prohibitiva para poder adaptarse a diferentes estilos y niveles.

b) *Basados en modelos de probabilidad*: incluye el modelo de Markov y el modelo de Markov oculto (Baum, 1970). David Cope (Cope, 2000) combina las cadenas de Markov y otras tecnologías (como la gramática musical) en un sistema semiautomático de composición algorítmica. Sin embargo, este tipo de modelo sólo puede producir subsecuencias existentes en los datos originales.

Los ejercicios creados por estos algoritmos, colmados de reglas y convenciones musicales, crean melodías y frases musicales con comportamientos inusuales que fácilmente son clasificados como artificiales por el oído de oyentes que posean un grado de entrenamiento musical medio. Es por ello que consideramos que para que estos contenidos musicales generados a partir de sistemas que responden a reglas preestablecidas (que se corresponden al tipo de inteligencia artificial denominado "sistemas expertos") sean utilizados deberemos incorporarles algunas otras funcionalidades a su motor generador. El trabajo de incorporación de un mecanismo accesorio que le otorgue estas características más realísticas o de mayor musicalidad* al generador de contenido musical mediante reglas recién descripto no se justifica por su complejidad y por los potenciales resultados a obtener a la luz del avance de este enfoque en la actualidad. (Akombo, 2019)

Por todo lo expuesto hasta aquí se evidencia la utilidad que traería aparejada el desarrollo de un modelo de ejercitación y formación musical que genere estructuras

* Esta característica de índole subjetiva, que indica si un hecho sonoro posee las características propias y distintivas de un hecho estético-musical, es susceptible de ser mensurada mediante métodos de evaluación objetivos y subjetivos, siendo el paradigma de estos últimos el test de Turing (Yang 2017)

musicales originales y variadas. Es en la inteligencia artificial, y en especial en las redes neuronales generativas, que el autor de este trabajo de investigación considera que existe la posibilidad de desarrollar un modelo generativo de contenido educativo

musical que posea las características mencionadas de variedad, musicalidad y extensión. Para cumplir tal cometido será necesaria la aplicación de variados métodos de evaluación para determinar si un modelo montado sobre redes neuronales es apto para crear melodías y pasajes musicales con la debida variedad y originalidad, y que su vez posea la capacidad de generar una cantidad virtualmente ilimitada de estos.

Capítulo 2 Estado del Arte

Los modelos de Inteligencia artificial especializados en la generación de contenido sintético se basan en arquitecturas de redes neuronales profundas (Harshvardhan, 2020). Como su nombre lo indica estas arquitecturas son un tipo especial de redes neuronales de alta complejidad especializadas en datos no estructurados, como es el caso de la música. Las redes neuronales son un caso particular de modelos de clasificación de aprendizaje automático o *Machine Learning.* Es por ello que se presentará un completo marco conceptual relativo a las características fundamentales de este tipo de modelos, sus variantes, los métodos utilizados para entrenamiento, la evaluación de su performance, y su posterior puesta en producción.

A su vez como el trabajo trata sobre creación de contenido musical, se presentarán los componentes fundamentales del arte musical, sus características básicas, y los elementos constitutivos para su expresión escrita que serán utilizados la hora de la preparación de los datos, el entrenamiento de los modelos y la exposición de los resultados obtenidos.

Al estar enfocado este trabajo en el desarrollo de un modelo generador de contenido musical con fines didácticos, deberemos realizar un exhaustivo trabajo de relevamiento de la literatura científica relacionada con la aplicación de la inteligencia artificial en la educación.

Finalmente , siendo que el generador a desarrollar utilizará las técnicas de *Machine Learning* (en espacial la de *Deep Learning*) se realizará un análisis del estado del arte en materia de generación de contenido musical en base a redes neuronales recurrentes, modelos estos especializados en el tratamiento de datos secuenciales. A

esta subdisciplina de la Inteligencia artificial se la denomina *Deep Music Generation.* Los algoritmos de esta se basan en otra rama del *Machine Learning* que ha tomado mucho vigor en los últimos años que es el Procesamiento de Lenguaje Natural, o NLP por sus siglas en inglés. Es en esta área en el que se han desarrollado los modelos más innovadores y se han obtenido los mejores resultados en la generación de contenido simbólico. Es por ello que al tomarse estos modelos como referencia para los desarrollos a implementar, se realizará un análisis exhaustivo de su literatura científica especifica.

Se ha mencionado también la necesidad de dotar al contenido sintético generado de características de musicalidad. Este concepto inherentemente subjetivo, que mensura las características estético-musicales de un hecho sonoro, se evalúa utilizando diferentes metodologías. Los dos tipos fundamentales de evaluaciones son la objetiva y la subjetiva. Estos técnicas de evaluación se utilizarán para calcular métricas que indiquen el grado de musicalidad del contenido generado, por lo que se presentará el estado del arte de ambos tipos de evaluaciones.

En la literatura científica especializada en procesamiento del Lenguaje Natural (NLP) se presentan modelos de generación de texto que han mejorado notablemente la calidad del contenido generado como consecuencia de la aplicación de una transformación sobre el *output* de las redes neuronales que se utilizan para creación del contenido sintético. Como ya se ha mencionado, este trabajo de investigación toma del NLP técnicas probadas , siendo la utilización de esta transformación, denominado “temperatura de Boltzmann”, una de ellas. Al estar esta transformación por temperatura asociada a la función de probabilidad empírica del siguiente componente del lenguaje a generar por la red neuronal, es que deberemos analizar diferentes métodos de comparación de distribuciones de probabilidad, para así calcular la divergencia o la distancia entre estas. A tal efecto es que al final de esta

sección se presentarán las técnicas utilizadas en la literatura científica para mensurar estas divergencias.

2.1 Aprendizaje Automático

2.1.1 Tipos de Aprendizaje Automático

Los algoritmos de *Machine Learning*, o aprendizaje automático, aprenden a partir de datos. Mitchell (Mitchell, 1997) nos ofrece la definición "Se dice que un programa de ordenador aprende de la experiencia *E* con respecto a una clase de tareas *T* y una medida de rendimiento *P* , si su rendimiento en las tareas de *T* , medido por *P* , mejora con la experiencia *E*".

Las tareas de aprendizaje automático generalmente se describen en términos de cómo el sistema debería procesar un ejemplo. Un ejemplo, o caso en la terminología del *Machine Learning*, es una observación sobre la cual hemos medido ciertos valores de variables de tipo cuantitativo o relevado alguna variable de tipo categórica o cualitativa. Por lo general, representamos un ejemplo como un vector $x \in R_n$ donde cada entrada x_i del vector es el valor de alguna de esta variables. El aprendizaje automático permite resolver muchos tipos de tareas. Algunas de las tareas de aprendizaje automático más comunes, y que son las que utilizaré a lo largo de este trabajo, son las que se detallan a continuación.

2.1.1.1 Clasificación Supervisada

En este tipo de modelos, se pide al algoritmo (recordemos que un algoritmo es una sucesión ordenada y estructurada de instrucciones lógicas y computacionales) que especifique cuál de los *k* posibles valores que puede tomar la variable categórica objetivo (también llamados niveles) corresponde a una entrada. Para resolver esta

tarea, se suele pedir al algoritmo de aprendizaje que produzca una función $f: R_n \rightarrow \{1, \ldots, k\}$. En su gran mayoría los modelos de clasificación son de índole probabilística, por lo que la tarea fundamental de un algoritmo de clasificación es aproximar la función $f(y/x)$, que es una aproximación de la función de distribución de probabilidad de la variable objetivo, o clase, de tipo categórica *y* dados los valores del vector *x*. Un ejemplo de tarea de clasificación es el reconocimiento de objetos, donde la entrada es una imagen (normalmente descrita como un conjunto de valores de intensidades en píxeles), y la salida es un código numérico que identifica el objeto en la imagen. El reconocimiento de objetos moderno se logra mejor con el aprendizaje profundo (Krizhevsky et al., 2012). El reconocimiento de objetos es la misma tecnología básica que permite a los ordenadores reconocer caras (Taigman et al., 2014), etiquetar automáticamente a las personas en una foto o permitir que un vehículo de conducción autónomo diferencie un objeto de otro en su recorrido, y por ende tome las acciones necesarias para interactuar con su ambiente sin incurrir en ningún peligro.

2.1.1.2 Generación de Datos Sintéticos

En este tipo de tarea se solicita al algoritmo de aprendizaje automático que genere nuevos ejemplos similares a los de los datos de entrenamiento. Por ejemplo, en una tarea de síntesis de voz, se le proporciona al modelo una oración escrita y se le pide al programa que genere una forma de onda de audio que contenga una versión hablada de esa oración (Luo et al., 2013). O simplemente se puede querer que de forma aleatoria genere una nueva oración.

2.1.2 Medidas o Métrica de Performance

Para tareas como clasificación o generación la precisión es simplemente la proporción de ejemplos para los que el modelo produce la salida correcta. En clasificación se

conoce el valor de la clase a la cual pertenece cada caso utilizado para entrenar el modelo, por lo que se podrá comparar la predicción, o la salida, del modelo con este valor correcto. A su vez si fuera generación de nuevos datos, al momento de entrenarlos se estarían utilizando los mismos datos de entrenamiento para corroborar qué tan parecido es el nuevo dato generado.

Por lo general, es de interés saber cómo funciona el algoritmo de aprendizaje automático con datos que no ha visto antes, ya que esto determina cómo funcionará cuando se implemente en el mundo real. Por lo tanto, se evalúan estas medidas de rendimiento utilizando un conjunto de datos de prueba que está aislado de los datos utilizados para el entrenamiento del sistema de aprendizaje automático. Este procedimiento se lleva a cabo separando el set original en dos subconjuntos, siendo que en uno de ellos se correrán todos los procesos asociados al entrenamiento del modelo de clasificación o generación, y en el otro sólo se corroborará su efectividad y se medirá su performance. Esta tarea de separación y las correspondientes tareas realizadas sobre cada uno de los subconjuntos generados (entrenamiento y validación, respectivamente) es fundamental en los procesos de aprendizaje automático, y es la clave en la que radica la efectividad de sus procedimientos y la validez y capacidad de generalización de sus modelos.

2.1.3 La experiencia “E”

Los algoritmos de aprendizaje automático pueden clasificarse a grandes rasgos como no supervisados o supervisados por el tipo de procesos que se les permite realizar durante el proceso de aprendizaje. Un conjunto de datos (en inglés: “dataset”) es una colección de ejemplos extraídos del entorno particular de análisis.

Los algoritmos de aprendizaje no supervisado aprenden propiedades de la estructura de este conjunto de datos. En el contexto del aprendizaje profundo, normalmente se desea aprender la distribución de probabilidad que generó un conjunto de datos, ya sea explícitamente como en la estimación de la densidad o implícitamente para tareas como la generación de datos sintéticos.

Los algoritmos de aprendizaje supervisado realizan diferentes procesos sobre un conjunto de datos que contiene características o atributos, pero en este caso se cuenta con un dato fundamental de los ejemplos o casos observados: se conoce el valor de la clase o variable objetivo.

A grandes rasgos, el aprendizaje no supervisado consiste en observar varios ejemplos de un vector aleatorio *x*, e intentar aprender implícita o explícitamente la distribución de probabilidad $p(x)$, o algunas propiedades interesantes de esa distribución, mientras que el aprendizaje supervisado consiste en observar varios ejemplos de un vector aleatorio $\boldsymbol{x}$ y un valor o vector asociado y, y aprender a predecir y a partir de *x*, normalmente estimando $p(y|x)$. El término aprendizaje supervisado tiene su origen en la idea de que el objetivo o clase "*y*" es proporcionado por un "supervisor" que muestra al sistema de aprendizaje automático lo que debe hacer. En el aprendizaje no supervisado, no hay supervisor, y el algoritmo debe aprender a dar sentido a los datos sin esta guía.

2.1.4 Capacidad, Sobreajuste y Subajuste

El objetivo fundamental del aprendizaje supervisado es obtener buenos resultados en datos no vistos por el modelo. La capacidad de obtener buenos resultados con datos no observados anteriormente se denomina *generalización*.

Normalmente, al entrenar un modelo de aprendizaje automático, tenemos acceso a un set de entrenamiento y lo que esencialmente hace el modelo es aplicar un método iterativo que, mediante la evaluación del error que se comete al comparar el valor de *y* que éste arroja con su verdadero valor (dato con el que cuenta en un problema supervisado), que va actualizando paso a paso el valor de sus parámetros mediante un proceso numérico basado en la reducción de este error. Hasta ahora, lo que se ha descripto descrito es simplemente un problema de optimización. Lo que separa el aprendizaje automático de la optimización es que se intenta que el error de generalización, también llamado error de validación (recordar que es el asociado a los valores separados del dataset original), sea también bajo. El error de generalización se define como el valor esperado del error en una nueva entrada. En este caso, la esperanza se calcula sobre diferentes observaciones posibles, extraídas de la distribución de entradas que se espera que el sistema encuentre en la práctica. Normalmente, se estima el error de generalización de un modelo de aprendizaje automático midiendo su rendimiento en un conjunto de ejemplos de prueba que se separaron del conjunto de entrenamiento.

El proceso de entrenamiento de un modelo supervisado consiste en refinar el valor de sus parámetros de modo de minimizar el error de predicción. Esto generalmente se hace de manera iterativa procesando los datos de a uno o en grupos pequeños (batch) y ajustando los valores de los parámetros convenientemente. Cuando el proceso de entrenamiento termina, se valida el modelo mediante los datos de prueba, que se mantuvieron separados del proceso de entrenamiento. Con este proceso, el error de prueba esperado es mayor o igual que el valor esperado del error de entrenamiento. Los factores que determinan el rendimiento de un algoritmo de aprendizaje automático son su capacidad para hacer que la diferencia entre el error de entrenamiento y el de prueba sea pequeña.

Los dos desafíos centrales a la hora de entrenar un modelo y determinar de manera iterativa (utilizando técnicas de optimización) los valores de sus parámetros son el s*ubajuste* y el *sobreajuste*. El subajuste se produce cuando el modelo no es capaz de obtener un valor de error suficientemente bajo en el conjunto de entrenamiento. El sobreajuste se produce cuando la diferencia entre el error de entrenamiento y el error de prueba es demasiado grande.

Podemos controlar si un modelo es más propenso a la sobreadaptación (sobreajuste) o a la inadaptación (subajuste) modificando su capacidad. Se define “capacidad” de un modelo a su habilidad para ajustarse a una amplia variedad de funciones. Los modelos con poca capacidad pueden tener dificultades para ajustarse al conjunto de entrenamiento. Los modelos con alta capacidad pueden sobreajustarse al memorizar propiedades del conjunto de entrenamiento que no sirven en el conjunto de prueba.

Los algoritmos de aprendizaje automático suelen funcionar mejor cuando su capacidad es adecuada para la complejidad real de la tarea que deben realizar y la cantidad de datos de entrenamiento que se les proporciona. Los modelos con una capacidad insuficiente son incapaces de resolver tareas complejas. A estos modelos se los suele denominar modelos “sesgados” o que poseen alto sesgo. Los modelos con una capacidad elevada pueden resolver tareas complejas, pero cuando su capacidad es superior a la necesaria para resolver la tarea actual pueden sobreajustarse. Estos modelos a su vez, al sobreajustarse a sus datos, terminan siendo muy sensibles a los datos de entrenamiento, poseyendo, en consecuencia, una alta varianza a la hora de presentar sus salidas. De hecho, hay muchas formas de modificar la capacidad de un modelo. La capacidad no viene determinada únicamente por la elección del modelo. El modelo especifica qué familia de funciones puede elegir el algoritmo de aprendizaje al variar los parámetros para reducir el error de entrenamiento. Esto se llama la *capacidad de representación* del modelo.

Se debe recordar que, aunque las funciones más sencillas tienen más probabilidades de generalizar (de tener una pequeña diferencia entre el error de entrenamiento y el de prueba), se debe elegir una hipótesis suficientemente compleja para lograr un bajo error de entrenamiento. Típicamente, el error de entrenamiento disminuye hasta asimilar el valor de error mínimo posible a medida que aumenta la capacidad del modelo . Normalmente, el error de generalización tiene una curva en forma de U en función de la capacidad del modelo. Esto se ilustra en la figura 1

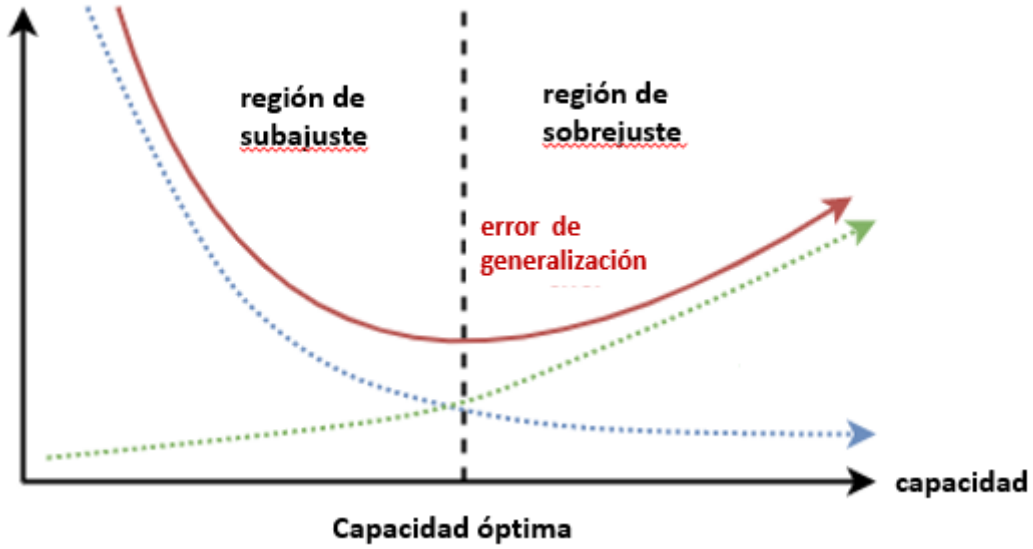


Figura 1. Error de generalización en función de la capacidad del modelo

El modelo ideal simplemente conoce la verdadera distribución de probabilidad que genera los datos. Incluso un modelo de este tipo incurrirá en algún error en muchos problemas, porque todavía puede haber algo de ruido en la distribución. En el caso del aprendizaje supervisado, el mapeo de *x* a *y* puede ser intrínsecamente estocástico, o *y* puede ser una función determinista que implique otras variables además de las incluidas en *x*. El error en el que incurre un modelo que hace predicciones a partir de la verdadera distribución $p(x, y)$ se llama error de Bayes, y es

tomado como el modelo de referencia o ideal al que se puede aspirar en la modelización mediante aprendizaje supervisado.

2.1.5 Hiperparámetros y Sets de Validación y Prueba

En mayoría de los algoritmos de aprendizaje automático se pueden realizar varios ajustes para controlar su comportamiento. Estos ajustes se denominan *hiperparámetros*. Los valores de los hiperparámetros no son adaptados por el propio algoritmo de aprendizaje, o sea que no son calculados a partir del entrenamiento del modelo . Si ponemos como ejemplo la regresión polinómica, hay un único hiperparámetro: el grado del polinomio, que actúa como regulador de la capacidad del modelo.

Normalmente, se utiliza alrededor del 80% de los datos de entrenamiento para el entrenamiento y el 20% para la validación. Como el conjunto de validación se utiliza para elegir los hiperparámetros, el error del conjunto de validación subestimará el error de generalización, aunque normalmente por una cantidad menor que el error de entrenamiento. Una vez completada la optimización de los hiperparámetros, el error de generalización puede estimarse utilizando el conjunto de prueba.

2.2 Aprendizaje Profundo (*Deep Learning*)

Las técnicas tradicionales de aprendizaje automático están limitadas en su capacidad para procesar datos en bruto, pues estos requieren mucho preprocesamiento humano para extraer las características adecuadas con el fin de mejorar su rendimiento. Es por ello que para estos casos se deba utilizar una técnica denominada *Aprendizaje de Representación*. El aprendizaje de representación se desarrolla entonces como un conjunto de métodos que permiten a una *máquina* (llamando máquina aquí a un algoritmo que recibe, realiza alguna serie de pasos especialmente diseñados de

transformación de ese input, y posteriormente entrega un output) que automáticamente procura descubrir las representaciones (características) necesarias para la detección o clasificación de datos (Pedrycs, 2020).

El aprendizaje profundo es una clase de métodos de aprendizaje automático de representación con múltiples niveles de representación. Se compone de varios módulos no lineales en los que cada uno de ellos transforma la representación de los niveles anteriores (empezando por la entrada bruta) en una representación de un nivel superior ligeramente más abstracto. Con la composición de suficientes transformaciones de este tipo, se pueden aprender características e inferencias muy complejas.

En general, los métodos de aprendizaje profundo más utilizados pueden clasificarse en las siguientes categorías: redes de avance hacia adelante (F*eedforward Networks, FFNN*), redes neuronales convolucionales (*Convolutional Neural Networks, CNN*) y redes recurrentes (*Recurrent Neural Networks, RNN*)

En el siguiente apartado se analizarán las redes neuronales recurrentes. En este trabajo le dará especial importancia a las RNN pues son las arquitecturas de modelos de *Deep Learning* más utilizadas para entrenar modelos de clasificación y generación en datos secuenciales, como es el caso de la música.

2.3 Redes neuronales recurrentes (RNN)

2.3.1 Conceptos Básicos

Las RNN son conocidas por su capacidad para procesar y obtener información de datos secuenciales. Por lo tanto, el análisis de vídeo, el subtitulado de imágenes, el procesamiento del lenguaje natural (NLP) y el análisis contenido musical dependen

de las capacidades de las redes neuronales recurrentes. A diferencia de las redes neuronales clásicas (*feedforward y CNN*), que asumen la independencia entre los datos de entrada, las RNN capturan activamente sus dependencias secuenciales y temporales.

Uno de los atributos más definitorios de las RNN es la que comparten sus parámetros. Sin compartir parámetros, el modelo asignaría parámetros únicos para representar a cada dato en una secuencia y, por lo tanto, no podría realizar inferencias sobre secuencias. El impacto de esta limitación puede observarse de forma notoria en el procesamiento del lenguaje natural. Una red multicapa tradicional fallaría porque crearía una interpretación del lenguaje con respecto a los parámetros únicos establecidos para cada posición (palabra) en una frase. Las RNN, sin embargo, son más adecuadas para la tarea, ya que comparten pesos entre los datos espaciados secuencialmente, en el ejemplo del lenguaje, las palabras/notas de nuestra frase/melodía. Además las RNN admiten datos secuenciales de longitud variable, a diferencia de las FFNN.

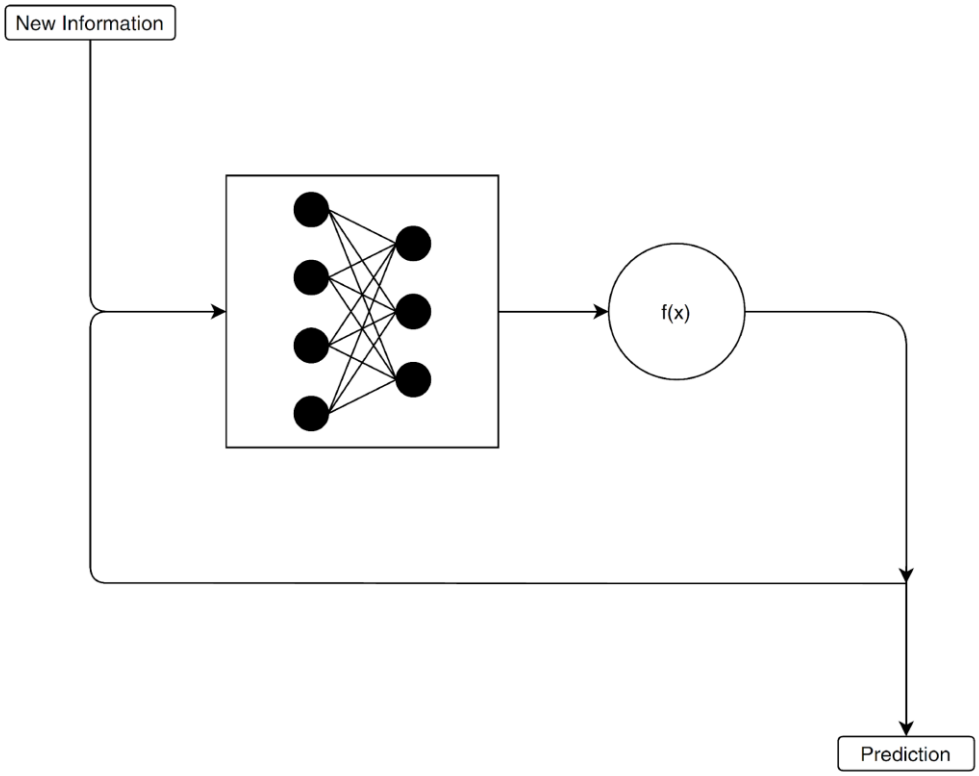


Figura 2. Diagrama de grafo cíclico de una Red Neuronal Recurrente (Charniak, E. 2018)

Las RNN generalmente aumentan la arquitectura de red multicapa convencional con la adición de ciclos que conectan nodos adyacentes o pasos de tiempo. Estos ciclos constituyen la memoria interna de la red que se utiliza para evaluar las propiedades del dato actual con respecto a los datos del pasado inmediato. También es importante tener en cuenta que la mayoría de las redes neuronales están limitadas a un mapeo uno a uno de la entrada y la salida (Goodfellow et al, 2016). Las RNN, sin embargo, pueden realizar mapeos de uno a muchos, de muchos a muchos (por ejemplo, traducir el habla) y de muchos a uno (por ejemplo, identificar la voz). Para representar los mapeos entre las entradas y las salidas y la pérdida se utiliza un grafo computacional. Al desplegar el grafo en una cadena de eventos se obtiene una imagen clara del reparto de parámetros dentro de la red, como se ve en la Figura 3.

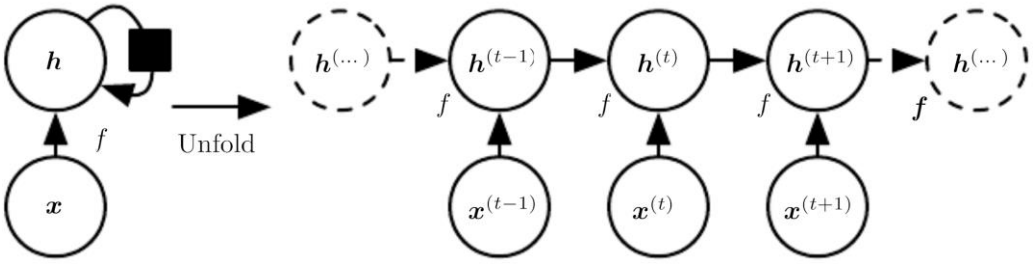


Figura 3. Red recurrente desplegada (Imagen extraída de Goodfellow 2016)

La ecuación generalizada para las relaciones de recurrencia es $s^{(t)}$,

$$s^{(t)} = f(s^{t-1})) \tag{1}$$

que indica el estado del sistema que depende de un paso de tiempo anterior indicado por *t -1*.

Esta ecuación puede reescribirse como

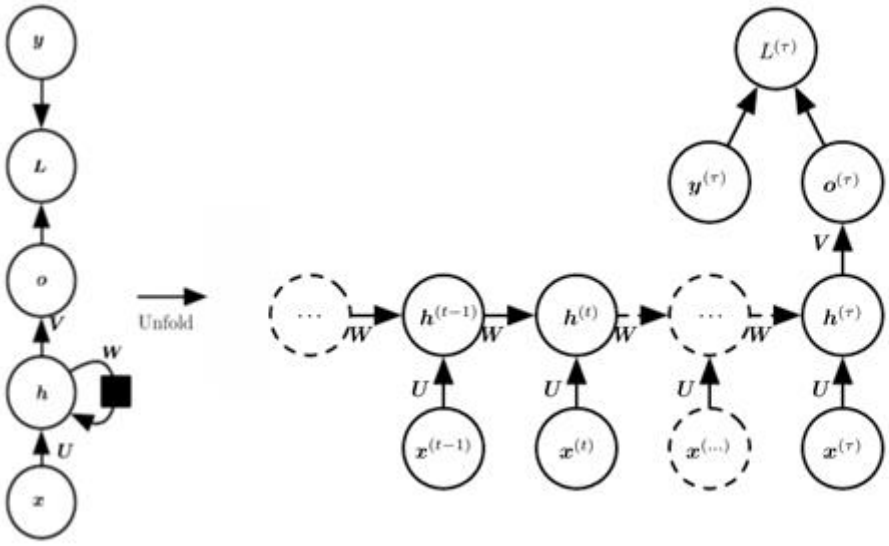
$$h^{(t)} = f(h^{t-1}, x^{(t)}) \tag{2}$$

donde $x^{(t}$ representa la entrada de una instancia de tiempo en particular. La importancia de $h^{(t)}$ es que es una representación de los aspectos relevantes de la secuencia pasada de entradas hasta *t*

2.3.2 Topologías de redes neuronales recurrentes

Dependiendo del problema que se trate de resolver, la red podrá trabajar con diferentes tipos de secuencias de entrada y de salida. Por ejemplo, analicemos el caso de un problema de análisis de series temporales en un mercado, en el que la red deberá tomar como entrada el valor de ciertas acciones en los últimos días y predecir cuál será su valor en el día *n+1*. Otra variante sería el caso de predecir los valores para cada uno de los días, basándose en el valor del día anterior. Entre otros, estos dos ejemplos dan una perspectiva de lo que se puede conseguir con redes neuronales recurrentes si se elige la topología adecuada. Se pueden definir cuatro topologías principales: *sequence-vector, sequence-sequence, vector-sequence, encoder-decoder*

La Figura 4 muestra la topología sequence-vector. La imagen muestra una capa de neuronas recurrentes desarrollada a lo largo del tiempo en la que sólo la salida del último time step es tomada en cuenta como salida de la capa.



.Figura 4. Topología tipo sequence-vector.

En la Figura 5 se presenta la topología sequence-sequence, donde la capa devuelve el resultado procesado en cada período de tiempo. Usualmente, las redes neuronales recurrentes profundas que buscan un comportamiento tipo sequence-vector se diseñan acoplando varias capas tipo sequence-sequence y una última capa de tipo sequence-vector. Como consecuencia de esta estructura, la salida de la red será un único elemento que se calculará en función del resto de los períodos de tiempo del lote.

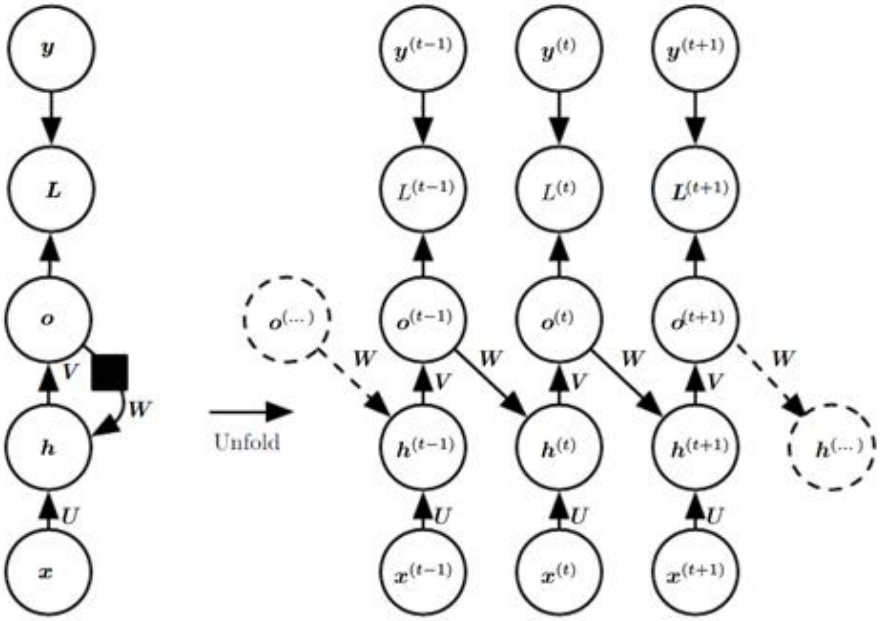


Figura 5. Topología de RNN sequence-sequence (Imagen extraída de (Goodfellow, 2016))

La última de las arquitecturas, Encoder-Decoder (Cho, 2014),deriva de la unión de sequence-vector y vector-sequence. Como puede verse en la Figura 6, la primera parte del modelo, el Encoder, toma como entrada una secuencia generando como última instancia una representación vectorial de la misma (modelo sequence-vector).

La segunda parte del modelo, el Decoder, toma dicha representación vectorial como entrada $h(t)$ y va devolviendo el resultado del procesamiento en cada período de tiempo. El modelo global puede verse como un sequence-sequence en el cual la longitud de la secuencia de entrada puede variar con respecto a la de la secuencia de salida. Esto es útil en problemas como la traducción automática, en el que una frase en un idioma determinado puede tener más símbolos o palabras que su homóloga en otro idioma diferente. Este tipo de modelos están teniendo una relevancia significativa en el campo del Procesamiento del Lenguaje Natural y en el de la composición automática de música, sobre todo desde la aparición de los mecanismos de Atención, cuya inclusión en estos modelos resultó en las arquitecturas Transformer (Vaswani, 2017).

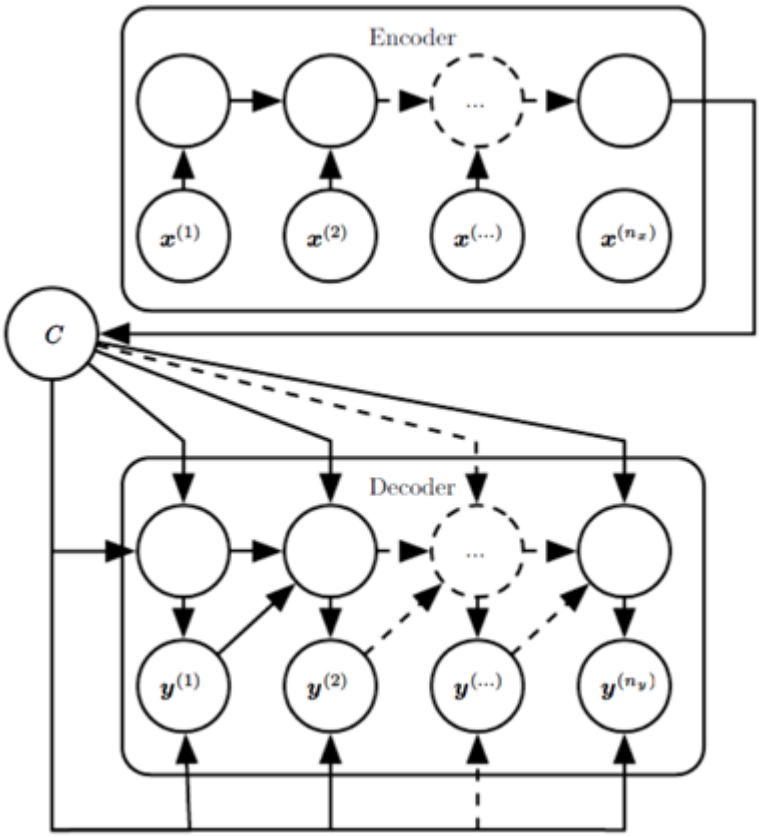


Figura 6: Topología tipo Encoder-Decoder. (Imagen extraída de (Goodfellow, 2016))

2.3.3 El problema de las redes neuronales recurrentes

Hasta este momento se ha visto cómo funcionan las celdas de memoria más básicas, las celdas RNN típicas. Este tipo de estructuras tienen, sin embargo, un problema llamado *short-term memory* (Hochreiter, 1991). Este problema deriva del bien conocido problema asociado al desvanecimiento del gradiente . Como se ha explicado hasta ahora, las redes neuronales recurrentes basan su funcionamiento en el procesamiento de secuencias de información, agregando al cómputo de cada elemento de la secuencia, el resultado del análisis del elemento anterior. Es decir, el resultado final de una RNN será una función que agregará el procesamiento de todos los elementos de la secuencia. Sin embargo, a medida que la red procesa más elementos de la cadena, tiene problemas en recordar información pasada.

Este problema surge como consecuencia del proceso matemático mediante el cual las redes neuronales recurrentes son capaces de entrenarse a partir de datos de un corpus: el algoritmo de propagación hacia atrás en el tiempo o *back-propagation through time* (Williams, 1995). A medida que la RNN recibe elementos de la secuencia, y por tanto se desarrolla a lo largo de los períodos de tiempo, el gradiente del error se propaga hacia atrás del mismo modo que lo haría en una red de avance hacia adelanta clásica. Del mismo modo que sucede en ese tipo de redes, a medida que dicho gradiente alcanza las capas más cercanas al input (en el caso de las RNN, a los primeros períodos procesados), decae exponencialmente. Es por esto que las RNN más sencillas no son capaces de aprender patrones muy extendidos en el tiempo, sino que solo son eficaces en rangos cortos, como por ejemplo secuencias de 10 elementos.

Para mitigar el problema de *short-term memory,* aparece un nuevo tipo de celda de memoria que sí es capaz de extraer patrones de secuencias de mayor longitud. Esta

celda más compleja se conoce como celdas de Memorias de Corto y Largo Plazo (LSTM, por sus siglas en inglés de *Long-Short Term Memory*)

2.3.4 Redes de Memoria Corta y Larga LSTM

Las redes neuronales de memoria de corto y largo plazo (LSTM) son un tipo particular de RNN que solucionan el problema de las RNN asociado a la memoria de corto plazo: el desvanecimiento o decaimiento del gradiente, y su explosión. En la figura 8 todo lo que hay en el recuadro punteado corresponde a una sola unidad de la red. En primer lugar, hay que tener en cuenta que se muestra una sola componente de una LSTM , por lo que a la izquierda se tiene la información procedente del procesamiento del dato asociado al período, utilizando para ello dos estructuras de datos en este caso del tipo tensor, que son arreglos de datos de más de dos dimensiones. En la parte inferior se tiene la entrada con información de la unidad anterior. A la derecha se encuentran dos tensores que salen para informar a la siguiente unidad de tiempo y, como en las RNN simples, se tiene esta información “hacia arriba" en el diagrama para predecir la siguiente palabra y la pérdida (parte superior lado derecho).

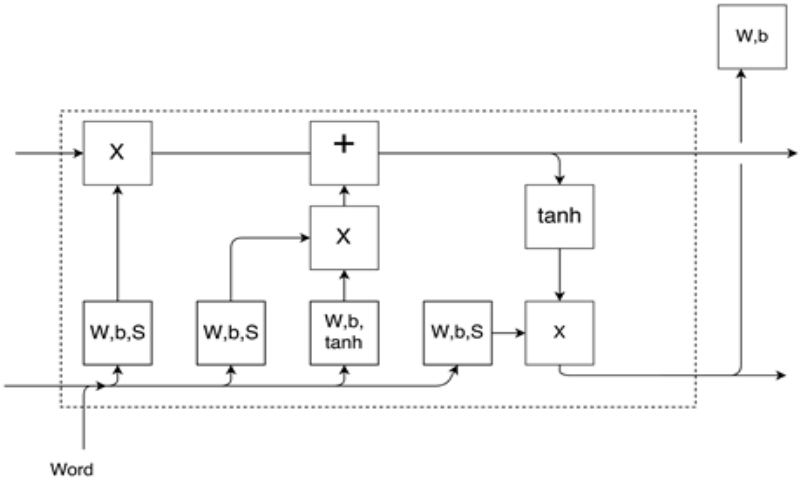


Figura 7 - Arquitectura de una celda LSTM (Imagen extraída de (Charniak, 2018).

El objetivo es mejorar la memoria de la RNN de los eventos pasados entrenándola para que recuerde lo importante y olvide el resto. Para ello, las LSTM procesan dos versiones del pasado. La memoria selectiva "social" está en la parte superior y una versión más local en la parte inferior. La línea de tiempo de la memoria superior se llama el estado de la célula y se abrevia *c*. La línea inferior se llama *h*.

La figura 7 introduce varias conexiones y funciones de activación nuevas. En primer lugar, vemos que la línea de memoria se modifica en dos lugares antes de ser pasar a la siguiente unidad de tiempo. Están etiquetados como tiempos **X**, y **+**. La idea es que las memorias se eliminan en la unidad de tiempo **X** y se añaden en la unidad **+**. ¿Por qué esto? La información actual que viene en la parte inferior izquierda pasa por una capa de unidades lineales seguida de una función de activación sigmoidea, como indica la anotación **W, b, S**. **W** y **b** forman la combinación lineal y **S** es la función sigmoidea de una neurona clásica. En notación matemática la operación es:

$$\mathbf{h'} = \mathbf{h_t} \cdot \mathbf{e} \quad (4)$$

$$\mathbf{f} = S((\mathbf{h'W_f} + \mathbf{b_f}) \quad (5)$$

Se utiliza un punto central para indicar la concatenación de vectores. Para repetir, en la parte inferior izquierda se concatena la línea *h* anterior h_t y el dato actual *e* para obtener h_0, que a su vez se introduce en la unidad lineal de "olvido" (seguida de una sigmoidea) para producir *f* , la señal de olvido que se desplaza hacia arriba en el lado izquierdo de la figura. La salida de la sigmoidea se multiplica elemento a elemento memoria que viene de la parte superior izquierda. (Por "elemento a elemento" queremos decir que, por ejemplo, el $x_{(i,j)}$ a elemento de una matriz se multiplica por $y_{(i,j)}$ elemento de la otra). La ecuación de esta operación se representa así:

(6)

$$\mathbf{c'_t} = \mathbf{c_t} \odot \mathbf{f}$$

Dado que los sigmoides están limitados por cero y uno, el resultado de la multiplicación debe ser una reducción en el valor absoluto en cada punto de la memoria principal. Esto corresponde al "olvido". En general, esta operación sigmoidea que alimenta una multiplicación es un patrón común cuando se quiere una compuerta "suave".

Contrasta esto con lo que sucede en la unidad aditiva con que se encuentra la memoria. De nuevo, el siguiente dato, que ha llegado desde abajo izquierda, pasa ahora por separado a través de dos capas lineales, una con una con activación sigmoidea y otra con la función de activación Tanh, como se muestra en la figura 8. Tanh significa tangente hiperbólica.

$$\mathbf{a_1} = S(\mathbf{h'W_{a_1}} + \mathbf{b_{a_1}}) \quad (7)$$

$$\mathbf{a_2} = \tanh((\mathbf{h_t} \cdot \mathbf{e})\mathbf{W_{a_2}} + \mathbf{b_{a_2}}) \quad (8)$$

Es importante que, a diferencia de la función sigmoidea, Tanh puede tener como salidas tanto valores positivos o negativos, por lo que puede arrojar valores nuevos en lugar de sólo escalar el dato de entrada. El resultado de esto se añade al estado de la celda en la celda etiquetada "**+**":

$$\mathbf{c_{t+1}} = \mathbf{c'_t} \oplus (\mathbf{a_1} \odot \mathbf{a_2}) \quad (9)$$

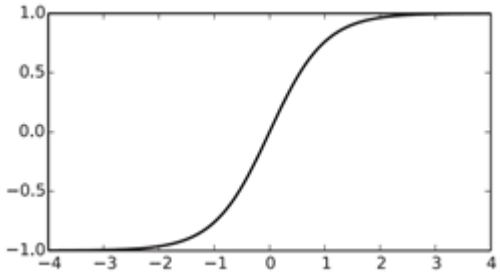


Figura 8. Función Tanh (tangente hiperbólica)

Después de esto, la línea de memoria se divide. Una copia sale por la derecha, y una copia pasa por un Tanh y luego se combina con una transformación lineal de la historia y el valor actual, para convertirse en la nueva línea *h* de la parte inferior:

$$\mathbf{h''} = \mathbf{h'}\mathbf{W_h} + b_h \tag{10}$$

$$\mathbf{h_{t+1}} = \mathbf{h''} \odot a_2 \tag{11}$$

Esta se concatenará con la siguiente entrada, y el proceso se repetirá. El punto a destacar aquí es que la línea de memoria de celdas nunca pasa directamente por las unidades lineales. La memoria se irá desvirtuando (se “olvidan”) en la unidad “X" y se añadirá nueva información del dato actual en “+", pero no habrá operaciones matemáticas ni de combinación lineal ni de escalamiento/transformación no lineal.

2.3.5 Mecanismos de *Attention*

El mecanismo de Attention fue presentado en 2014 por Dzmitry Bahdanau et al. como extensión para mejorar el rendimiento del modelo Encoder Decoder en la tarea de la traducción automática (Bahdanau, 2014). En un encoder decoder convencional, el encoder representa mediante un vector de tamaño fijo (estado oculto final) toda la

información de la frase de entrada. En palabras de los propios autores, “el rendimiento de un encoder decoder básico decae rápidamente a medida que la longitud de la cadena de entrada crece.”. El mecanismo de *Attention* pretende solventar precisamente este problema. Mediante esta técnica, un conjunto de vectores es generado en lugar de un solo vector de dimensión fija. Entre dichos vectores se selecciona un subconjunto cercano al término que se pretende traducir en cada time step.

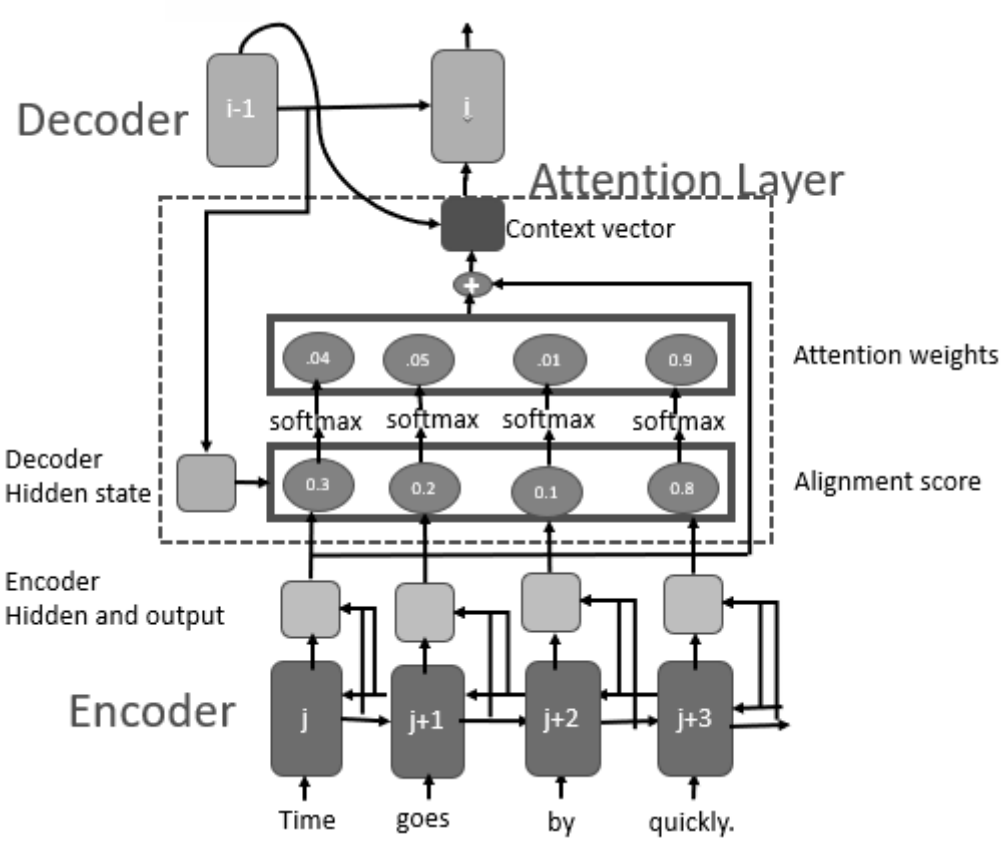


Figura 9: Arquitectura de *attention* (Imagen extraída de https://towardsdatascience.com/sequence-2-sequence-model-with-attention-mechanism-9e9ca2a613a)

En la Figura 9, puede verse un esquema de la arquitectura de un *encoder-decoder* intentando generar la t-ésima palabra objetivo y_t dada la frase $(x_1, x_2, ..., x_T)$. Como se

ve, en lugar de mantener como vector final el estado oculto de la celda en el último step del encoder, se consideran todas las salidas intermedias hasta el momento y se computa la suma ponderada de dichos vectores intermedios. Este cálculo permite determinar qué palabras serán más relevantes para el decoder en ese step.

Existen varias propuestas basadas en este mecanismo:

- **Visual attention:** Fue propuesto en el año 2015 por Xuet et al. El objetivo de esta propuesta consistía en un sistema capaz de alinear la imagen de entrada y producir una palabra que la describiese como salida. La arquitectura consistía en una red convolucional que extraía características de la imagen y a continuación una red recurrente con mecanismos de atención para dar como salida la palabra. El mecanismo de attention consigue focalizarse en ciertos elementos de la imagen que la definen de manera que la frase de salida se corresponda con dichos elementos.(Xuet et al. 2015)

- **Herarchical attention:** Esta propuesta llego en 2016 de la mano de Xu et al (Xu, 2015). Se trata de un sistema de clasificación de documentos que aplica los mecanismos de attention en dos niveles. La arquitectura presenta dos módulos encoder decoder + attention. El primero de ellos aplicado a nivel de frase y el segundo a nivel de palabra.

- **Transformer**: La consecuencia más interesante de la aparición de los mecanismos de attention fue la aparición de los modelos Transformer, propuestos por Vaswaniet et al. en 2017 (Vaswaniet et al, 2017). Esta arquitectura supuso un nuevo nivel de mejora en el campo del Procesamiento del Lenguaje Natural. Este tipo de sistema permite alinear ciertas palabras de una secuencia con otras, calculando una

representación de dicha secuencia mucho más precisa y eficiente que en anteriores modelos.

2.4 Técnicas para la Generación de Contenido en Lenguajes Simbólicos

2.4.1 Conceptos Básicos

Un modelo lingüístico es, básicamente, un modelo de aprendizaje automático que es capaz predecir el siguiente componente a partir de una secuencia previa de componentes básicos (en el caso del texto el componente será la palabra o la letra, y en la música la nota).

En el entrenamiento, suministramos una secuencia de estos componentes como entrada *X* y un componente de destino como salida *y*. Tras el entrenamiento, el modelo lingüístico aprende la distribución de probabilidad sobre el vocabulario de componentes condicional a la secuencia de entrada dada. A continuación, veremos como un modelo de lenguaje estima la función de distribución de probabilidad condicional del siguiente componente del lenguaje dada la secuencia de componentes anteriores como inputs.

El conjunto de valores que puede tomar esta variable es el diccionario completo de este lenguaje, o un subconjunto de éste delimitado previamente. A partir de esta distribución recién estimada es que se deberá seleccionar el siguiente componente de la secuencia (también llamado *token*), y así ir generando un contenido, en este caso texto.. En los modelos de clasificación, como conocemos el valor de la variable objetivo y podremos tomarlo como referencia -o supervisión- de la performance del modelo, nos inclinamos naturalmente a tomar el componente del diccionario que posea la mayor probabilidad, o sea la moda de la distribución. Planteemos

formalmente este proceso de generación mediante modelos autorregresivos de la siguiente manera

$$P(w_{1:T}|W_0) = \prod_{t=1}^{T} P(w_t|w_{1:t-1}, W_0) \quad (12)$$

Siendo $P(w_{1:T}/W_o)$ la probabilidad conjunta de generar una secuencia de tokens $w_{1:T}$ de longitud T condicionada a una secuencia de tokens previa W_o , y $P(w_t / w_{1:T,} W_o)$ la probabilidad condicional de generar el token w_t dada la secuencia $w_{1:T,}$ y W_o . La longitud T de la secuencia $w_{1:T}$ de palabras a generar queda determinada durante el proceso generativo y toma su valor dependiendo de la cantidad de tokens generados al momento en el que el modelo genera el token de fin de oración (o EOS por sus siglas en inglés). Comúnmente este token es la barra o línea diagonal (“/”). Este token del EOS se generará a partir de la estimación de la probabilidad condicional

$$P(w_t|w_{1:t-1}, W_0) \quad (13)$$

La estrategia más utilizada para generar el siguiente token a partir de la distribución empírica de probabilidad condicional $P(w_t / w_{1:T,} W_o)$ es la selección basada en la maximización. Esta estrategia selecciona como siguiente componente del lenguaje a la moda de la distribución empírica calculada.

Este procedimiento posee una falla inherente que se presenta al comparar la elección de la siguiente palabra dada por el modelo con la que normalmente escogería un humano. El lenguaje natural rara vez presenta componentes (palabras en la lengua hablada/escrita) que posean la probabilidad más alta (la moda) en la distribución

condicionada a una secuencia de varios componentes generados, sino que se desvía hacia aquellos de menor probabilidad pero más informativos (Holtzman et al., 2020)

En la siguiente figura se observa como los modelos de selección del siguiente componente por maximización eligen el componente con mayor probabilidad, mientras que los humanos tendemos a seleccionar otros con probabilidades muy dispares. La varianza de la probabilidad de la selección humana es mucho mayor a la del modelo que utiliza un procesamiento de selección por maximización.

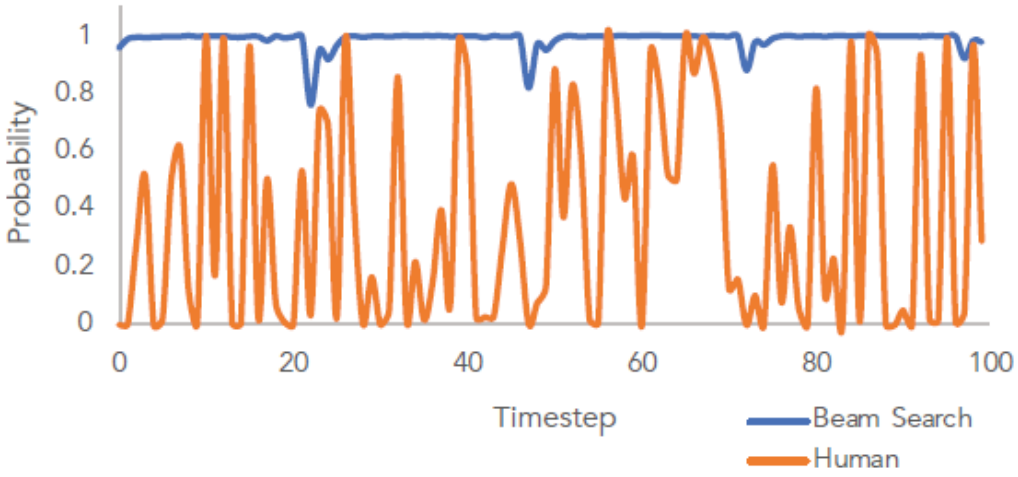


Figura 10. Evolución de la probabilidad del siguiente token en función del tiempo (extraído de Holtzman et al., 2020)

¿Por qué el texto escrito por humanos no es el más probable? Se conjetura en la literatura específica que se debe a una propiedad intrínseca del lenguaje humano por la que se tiende naturalmente a la creatividad y a la sorpresa. Los modelos lingüísticos que asignan probabilidades componente por componente sin un modelo global del texto tienen problemas para captar este efecto. Las máximas de la comunicación de Grice (Grice, 1975) demuestran que las personas optimizan el proceso de selección de las palabras utilizadas en sus discursos utilizando procedimientos que van contra la afirmación de lo obvio. Por lo tanto, hacer que cada palabra sea la moda de la

distribución de probabilidad de las palabras de un diccionario será desfavorable para el desarrollo de un modelo que intente emula el comportamiento humano.

Es por lo anterior que un enfoque más acorde a estos hallazgos se debería basar en la utilización de alguna técnica de muestreo. Se debe comenzar con la más básica, que reemplaza la selección de la moda como siguiente componente por otro muestreado aleatoriamente a partir de la distribución de probabilidad empírica del siguiente componente calculada por el modelo.

2.4.2 Selección por Muestreo

La forma más básica de muestreo en NLP consiste en elegir al azar la siguiente palabra/letra según su distribución de probabilidad condicional:

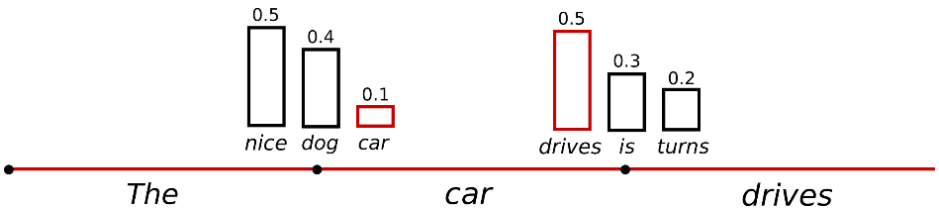


Figura 11. Ejemplo de selección de la siguiente palabra (extraído de Holtzman et al., 2020)

En el ejemplo de la figura 11 la palabra “car" se elige muestreando la distribución de probabilidad condicionada P(sig_palabra/ “The"), seguida de “drives”, que es seleccionada como consecuencia de muestrear a partir de la distribución P(sig_palabra / “The car”)

Sería muy beneficioso poder controlar el grado de creatividad o sorpresa asociado a la selección del siguiente componente ya conociendo su función de distribución de probabilidad empírica, para intentar aproximarnos la variabilidad observada en la elección humana.

2.4.3 Muestreo con temperatura

Otro enfoque común para la generación basada en el muestreo es dar forma a una distribución de probabilidad mediante "temperatura" (Ackley et al., 1985). El concepto de temperatura se toma de la ecuación de Boltzmann, que describe el comportamiento estadístico de un sistema fuera del equilibrio termodinámico . A su vez un concepto similar, el de *annealing*, es utilizado en la máquina de Boltzmann restringida (RBM), un tipo de red neuronal artificial estocástica generativa que puede aprender una distribución de probabilidad a partir de un conjunto de datos de entrada. Su nombre proviene del hecho de que es una forma restringida de una máquina de Boltzmann general (Ackley, et al.)

El mecanismo del muestreo por temperatura consiste en hacer más pronunciada la distribución de la siguiente palabra (condicionada a las anteriores palabras), aumentando la probabilidad de las palabras de alta probabilidad y disminuyendo las de baja. Esta operación se realiza directamente utilizando la ecuación de normalización softmax ahora modificada por la inclusión de la temperatura.

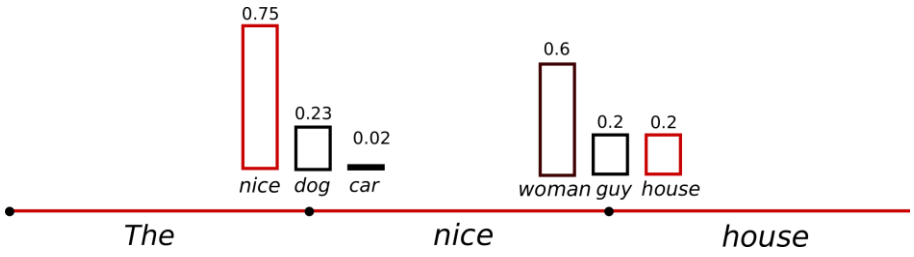


Figura 12. Ejemplo de selección de la siguiente palabra mediante temperatura (extraído de Holtzman et al., 2020)

Formalmente está planteada de la siguiente manera: dados los logits de salida de la red neuronal de entrenamiento, y la temperatura *t* elegida, se recalculan las salidas softmax de la siguiente manera:

$$p(x = V_l | x_{1:i-1}) = \frac{\exp(u_l/t)}{\sum_{l'} \exp(u'_l/t)}. \qquad (13)$$

En el siguiente gráfico se observa la distribución de probabilidad sobre el siguiente componente condicionado a los componentes anteriores calculada a partir de la salida de un generador de texto sin aplicar temperatura (o sea con *t* = 1)

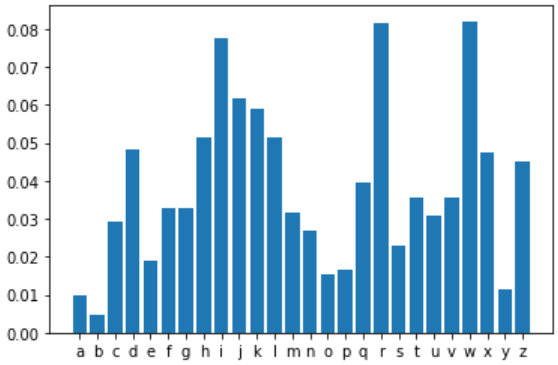


Figura 14. Distribución de probabilidad del siguiente token

Un ejemplo de la función en lenguaje Python que genera el siguiente elemento a partir de la distribución generada, sería la de la figura 15

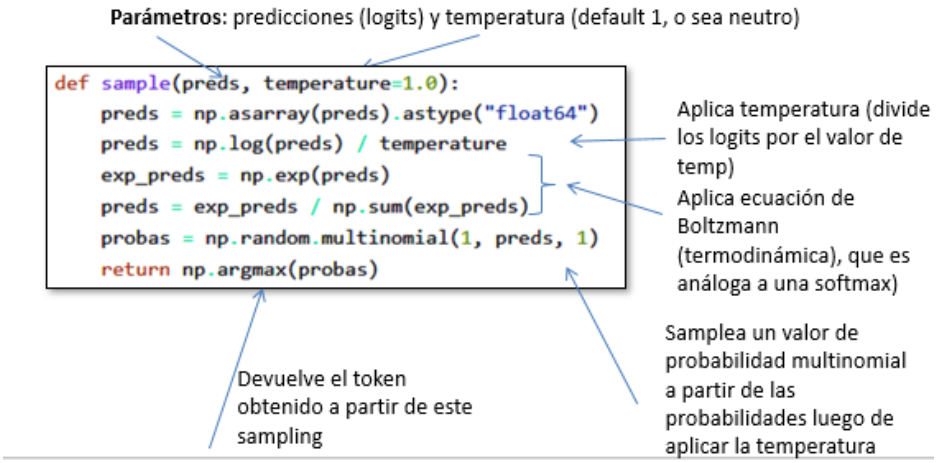


Figura 15. Función de Python para el muestreo por temperatura

A continuación observamos la forma de diferentes distribuciones de probabilidad de los elementos del diccionario luego de las operaciones de softmax para calcular la distribución utilizando diferentes temperaturas de Boltzmann

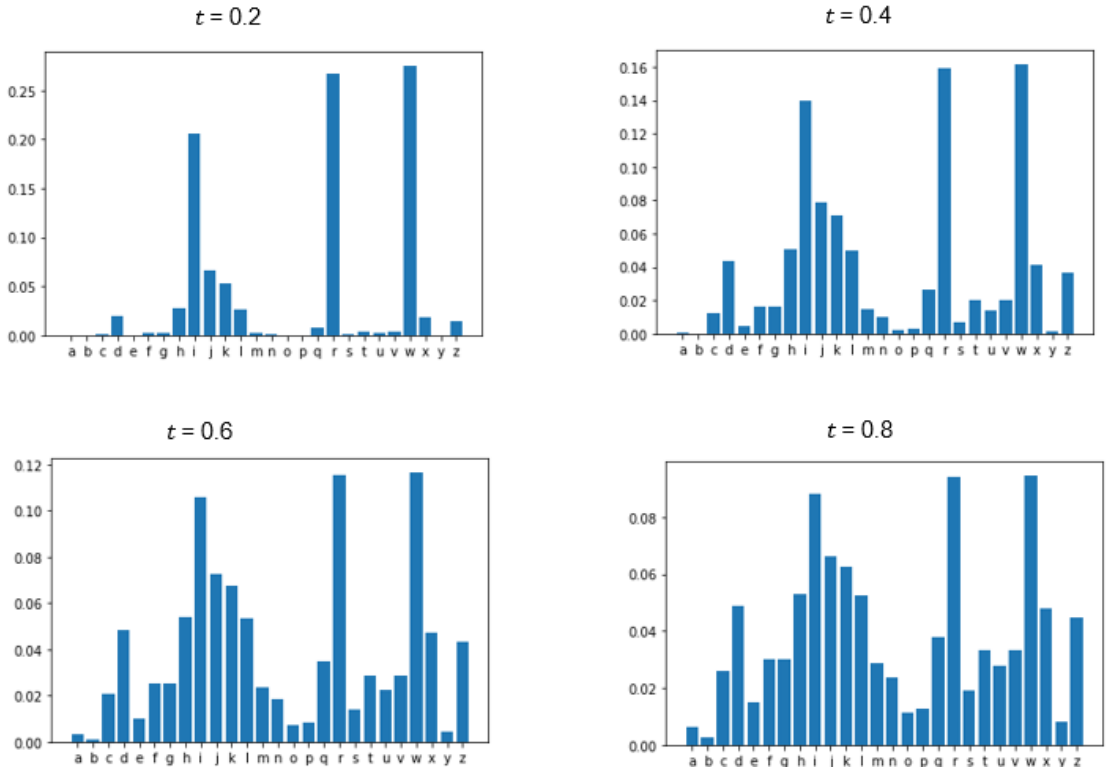


Figura 16 – Función de distribución de probabilidad del siguiente token para diferentes temperaturas (en el ejemplo los tokens son letras)

Las temperaturas más bajas hacen que el modelo tenga más propensión a seleccionar los tokens más “lógicos” , o sea que tenga cada vez más confianza en sus opciones principales, mientras que las temperaturas superiores a 1 disminuyen esta confianza, haciendo más probable la selección de tokens más sorpresivos u originales. La temperatura 0 equivale a la probabilidad *arg max* (o selección de la moda) mientras que la temperatura infinita corresponde a una distribución uniforme.

Utilizando esta técnica de sampleo por temperaturas Caccia et al han demostrado que analizando diferentes temperaturas se pueden desarrollar modelos que mejoren el rendimiento generativo medidos en términos de creatividad y variedad. (Caccia et al., 2018).

2.5 Vocabulario Musical y Representaciones

2.5.1 Conceptos Musicales Básicos

2.5.1.1 Notas musicales y tonos

El vocabulario musical se articula sobre estructuras de información dinámica, que le imprimen el carácter secuencial propio de los lenguajes. Su componente básico es la nota, que es un sonido que posee una determinada altura y una cierta duración. El concepto de nota recién presentado esta más asociada a la escritura musical, siendo que en la práctica también se lo denomina así a la altura del sonido. La altura de una nota es una característica perceptiva que indica que tan alto o bajo es el sonido. Y como se sabe que el sonido es originado por un objeto vibrante (como las cuerdas de un violín o las cuerdas vocales de una cantante) que genera desplazamientos y oscilaciones en las moléculas del fluido en el que se desplazan, en este caso el aire, que originan variaciones de presiones. Estas variaciones se transportan por el aire en forma de ondas. Estas ondas son las que percibe el oído, que luego de recorrer

variados caminos y procesos tanto auditivos como neuronales, generan la percepción sonora. Es por ello que al poder representar a la nota por una onda, se sabrá qué frecuencia fundamental este posee. Entonces la percepción de la altura de una nota, también llamada su afinación, va a tener un correlato directo con su frecuencia, siendo la nota más “alta” si la frecuencia es mayor, o más baja si es menor. Dos tonos cualquiera cuya relación entre sus frecuencias sea igual a cualquier potencia de dos (p.e., la mitad, el doble, cuatro veces, etc.) son percibidas por el oído como muy similares. Por eso es que se las puede clasificar dentro de la misma “clase de nota.” Esto deriva en el concepto de octava, que es el intervalo que existe entre dos notas cuando la frecuencia de uno de ellos es el doble de la del otro. Ahora se podrá definir el concepto de escala, que es un conjunto finito de notas comprendido en una octava. En el gráfico siguiente se ven las escalas de C mayor y menor

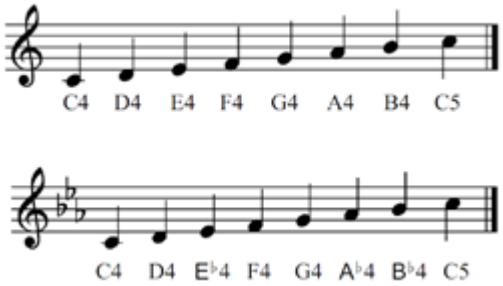


Figura 16. Escalas de C mayor y C menor

2.5.1.2 Concepto de compases.

En la música occidental a la octava solo la se puede dividir en una cantidad máxima de 12 notas equidistantes (recordar que la percepción auditiva no es lineal, si no logarítmica, por ende la distancia entre estas notas será medida en logaritmos de sus frecuencias). A la diferencia de altura que hay entre dos cualesquiera notas de estas doce posibles se la denomina semitono y es la mínima diferencia de tono que puede haber entre dos notas.

La música suele organizarse en unidades temporales, denominadas pulsos. Al pulso se suele marcar naturalmente cuando se mueve el pie contra el piso de forma regular, o al aplaudir al escuchar música. Las secuencias repetidas de pulsos acentuados y no acentuados, a su vez, forman patrones temporales superiores que se denominan "compás" y conforman la primera unidad aglutinadora del discurso musical. Un compás es un segmento de tiempo definido por un número determinado y fijo de pulsos. La división de la música en compases no sólo refleja su naturaleza rítmica, y le da marco a la presencia de los restantes componentes de la música (melodía y armonía), sino que también proporciona puntos de referencia regulares dentro de ella, dando así la sensación de contención necesaria para comprender, asimilar y disfrutar del mensaje musical.

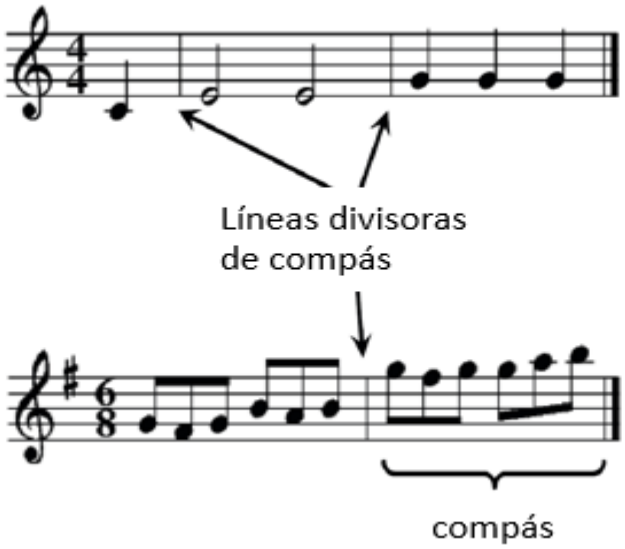


Figura 17. Compases

2.5.2 Representaciones simbólicas

2.5.2.1 Sobre las diferentes representaciones simbólicas

Las partituras proporcionan una representación visual de una obra mediante un lenguaje formal basado en símbolos musicales y letras, que se representan de forma gráfico-textual. Al leer las partituras, un músico puede una interpretar la obra de un compositor siguiendo las instrucciones dadas. Esta representación visual de la música mediante partituras se representa comúnmente en formato impreso.

Esta representación de la música es eminentemente simbólica y sólo será interpretada por un músico o lector humano que conozca las reglas y las convenciones propias de esta forma de plasmar el discurso musical en papel. Si se digitalizara una partitura y se la transformara en información digital de intensidad de grises en un pixel (lo que hace cualquier software de escaneo) todo el significado que posee cada símbolo se vería perdido y ningún software podría interpretar y decodificar el mensaje musical. Por tanto, ni las imágenes escaneadas ni las grabaciones musicales digitalizadas se consideran formatos musicales simbólicos. Aun así, existe una amplia gama de lo que puede considerarse como representación simbólica de la música.

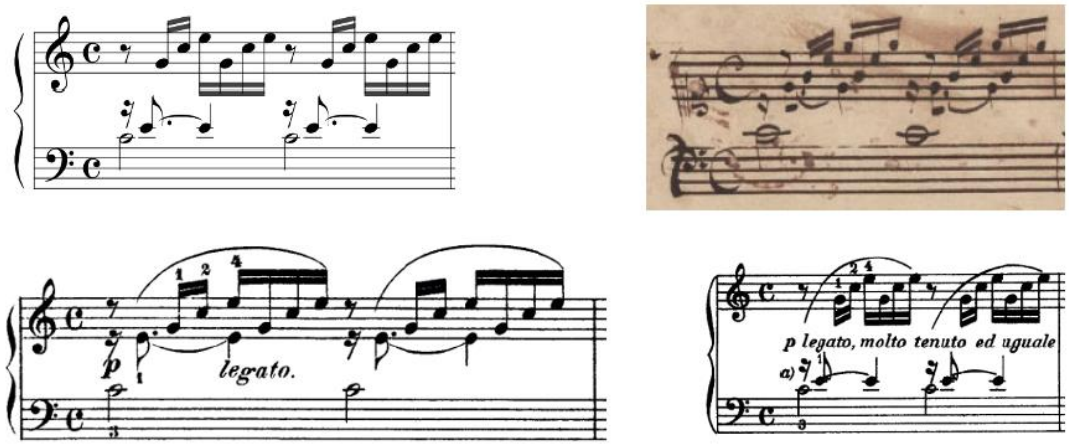


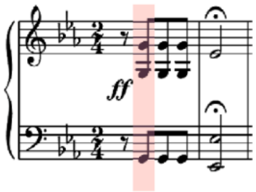
Figura 18 – Diferentes tipos de partituras

2.5.2.2 Representación MIDI

El MIDI es un protocolo de comunicación que se desarrolló originalmente como un estándar industrial para que instrumentos musicales electrónicos digitales de diferentes fabricantes funcionaran y pudieran ser utilizados en conjunto. MIDI son las siglas de "Interfaz digital de instrumentos musicales" (*Musical Instrument Digital Interface*). El protocolo MIDI le permite a un músico controlar a distancia y automáticamente un instrumento electrónico en tiempo real. Como ejemplo, un piano digital MIDI, en el que un músico pulsa una tecla del teclado del piano para iniciar un sonido. La intensidad del sonido se controla con la velocidad de la pulsación. Al soltar la tecla, el sonido se detiene. Toda esta información (que contiene la nota, su duración, intensidad tipos de sonido, etc.) será codificada en formato MIDI permitiendo reproducir a posteriori, o en simultáneo por otro instrumento o dispositivo, las mismas secuencias con las mismas características, tanto musicales como interpretativas.

Para los fines de este estudio, los mensajes MIDI más importantes son los comandos note-on y note-off, que corresponden al inicio y al final de una nota, respectivamente. El número de nota MIDI es un número entero entre 0 y 127 y codifica el tono de una nota.. Al igual que las representaciones de partituras, el MIDI puede expresar la información de tiempo en términos de entidades musicales.

(a)



(b)

Time (Ticks)	Message	Channel	Note Number	Velocity
60	NOTE ON	1	67	100
0	NOTE ON	1	55	100
0	NOTE ON	2	43	100
55	NOTE OFF	1	67	0
0	NOTE OFF	1	55	0
0	NOTE OFF	2	43	0
5	NOTE ON	1	67	100
0	NOTE ON	1	55	100
0	NOTE ON	2	43	100
55	NOTE OFF	1	67	0
0	NOTE OFF	1	55	0
0	NOTE OFF	2	43	0
5	NOTE ON	1	67	100
0	NOTE ON	1	55	100
0	NOTE ON	2	43	100
55	NOTE OFF	1	67	0
0	NOTE OFF	1	55	0
0	NOTE OFF	2	43	0
5	NOTE ON	1	63	100
0	NOTE ON	2	51	100
0	NOTE ON	2	39	100
240	NOTE OFF	1	63	0
0	NOTE OFF	2	51	0
0	NOTE OFF	2	39	0

Figura 19 – Correspondencia entre partitura y representación MIDI

2.5.2.4 Formato XML

El formato Music XML ha sido desarrollado para servir como formato universal para almacenar archivos musicales y compartirlos entre diferentes aplicaciones de notación musical. Siguiendo el paradigma general de XML (*Extensible Markup Language*), MusicXML es un formato de datos textual que define un conjunto de reglas para codificar documentos de forma que sean legibles tanto por personas como por máquinas. Por ejemplo, la figura 20 muestra cómo se codifica una nota E^b4. En la codificación MusicXML de la nota blanca E^b4, las etiquetas <note> y </note> marcan el principio y el final de un elemento de nota MusicXML. El elemento pitch, delimitado por las etiquetas <pitch> y </pitch>, consta de un elemento de clase pitch E (que denota el nombre de la letra del pitch), el elemento alter -1 (que cambia E a E flat), y el elemento octave 4 (que fija la octava). Así, la nota resultante es un E^b4. El elemento <duration>2</duration> codifica la duración de la nota medida en negras, siendo la negra la nota de referencia por estar asociada su duración a la de un pulso musical (*quarter_note duration*). Por último, el elemento <tipo>half</type> (*half-note* en inglés

significa una duración de blanca) nos indica cómo se representa realmente esta nota en la partitura gráfica.

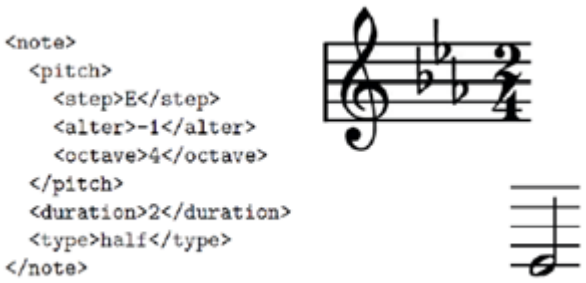


Figura 20 – Equivalencia entre la misma nota en formato XML y en partitura

Hay varias maneras de generar representaciones de partituras digitales. Por ejemplo, se puede introducir manualmente la información de la partitura en un formato como MusicXML. Sin embargo, este es un procedimiento tedioso y propenso a errores. Los programas informáticos de notación musical o de escritura de partituras ayudan a los usuarios en la tarea de escribir y editar partituras digitalizadas. Ejemplos de estas soluciones de software es Sibelius (Avid, 1987) . Estos programas permiten al usuario introducir y modificar cómodamente los objetos de las notas mediante dispositivos de entrada de ordenador estándar o teclados electrónicos.

2.5.2.5 Codificación de Representación Simbólica MIDI - XML

Una vez se ha escogido convenientemente cuál va a ser el formato en el que se representarán los datos, es el momento de transformar dicha información en valores numéricos de manera que los modelos que se usen puedan trabajar con ella. Dependiendo de la representación de los datos, las estrategias de codificación podrán variar.

En el caso de trabajar con representaciones simbólicas, como es el caso de la mayoría de las propuestas en el ámbito del Deep Learning, se deben aplicar técnicas específicas para transformar los datos en valores numéricos interpretables por los modelos. En su libro, Briot et al. proponen una clasificación de posibles métodos de codificación para este tipo de datos (Briot et al., 2020):

Valor numérico continuo: Este método consiste en tomar el valor de la frecuencia en Hz de la nota (el tono) para representarla unívocamente. El dominio en este caso serían los números reales.

Valor numérico discreto: La segunda opción consiste en hacer uso del valor que el sistema MIDI le otorga a la nota en función de su tono.

One-hot encoding: Esta técnica tradicional de codificación consiste en representarlos datos mediante un vector de dimensión igual al número de elementos diferentes que existen (vocabulario). Todos los elementos del vector serán ceros excepto aquel que represente a la nota en cuestión, que tendrá el valor 1. Esta estrategia es apropiada para líneas monofónicas, sin embargo para cuando hay varias notas o pistas sonando a la vez, los autores proponen las opciones que siguen a continuación (Briot et al., 2020)..

Many-hot encoding: Es igual que one-hot encoding pero en este caso todas las notas que están siendo tocadas se representan con un 1 en el vector.

Multi-one-hot encoding: Se trata de usar un vector con formato one-hot encoding para representar cada una de las voces o pistas sonando en ese momento.

2.6 - Inteligencia Artificial aplicada a la Educación

En este apartado se evaluará el impacto en los diferentes estratos y procesos educativos de las diferentes tecnologías y disciplinas que componen el paradigma de la Inteligencia Artificial. Más concretamente, el estudio tratará de evaluar cómo la IA ha afectado a la enseñanza y al aprendizaje

2.6.1. Modelos Interactivos Y Sistemas De Tutoría Inteligentes

Una de las responsabilidades más importantes del profesor o instructor es la de dar una devolución apropiada y específica al alumno (Wang, 2015). Este simple y fundamental acontecimiento propio del aula física tradicional puede ser sumamente complicado de consumarse si es que la clase se dicta frente a un alumnado numeroso, máxime si ésta se configura de forma virtual, como es el caso de los MOOCs (por sus siglas en ingles de *Massive Open Online Courses*). Es por ello que muchas instituciones educativas, especialmente universidades, implementan entornos de aprendizaje interactivos ILE, por sus siglas en inglés de *Interactive Learning Environments.* En estos el fin que se persigue es mejorar la calidad de la enseñanza mediante la retroalimentación y la tutoría personalizada. ILE es un término complejo, que exige que sus modelos sean una combinación de técnicas de e-learning, en los que a los sistemas de gestión del aprendizaje (LMS por sus siglas en inglés de *Learning Management Systems*) se le adicionan funcionalidades propias de un modelo interactivo. El objetivo, y la idea rectora sobre la que se articulan los entornos de aprendizaje interactivos, es la comprensión profunda de los conceptos y contenidos curriculares por parte de los estudiantes, tomando especial consideración de su experiencia personal y de la asimilación de los conocimientos presentados, para a partir de ellas realizar las interacciones apropiadas que mejoren y optimicen el proceso de aprendizaje.

Chassignol, en su completo trabajo de revisión de publicaciones científicas del tema (Chassignol, 2018), presenta los siguientes cuatro componentes fundamentales de los modelos de aprendizaje interactivo:

- Contenido
- Métodos de enseñanza
- Evaluación
- Comunicación.

Contenido: se refiere al conjunto de conocimientos que los profesores enseñan y que se espera que los alumnos aprendan en una determinada asignatura o área temática. Se incluye bajo este componente tanto el contenido educativo como su personalización, que como veremos será una de las áreas fundamentales de la aplicación de las tecnologías basadas en IA, en particular las relativas a los procesos de decisión autónoma.

Métodos de enseñanza: representan los diferentes principios y procedimientos didácticos utilizados por los profesores para transmitir el conocimiento correspondiente y asegurarse la correcta asimilación por parte de los estudiantes.

Evaluación: se refiere a la amplia variedad de métodos o herramientas que los educadores utilizan para evaluar, medir y documentar la preparación académica, el progreso del aprendizaje, la adquisición de habilidades y/o las necesidades educativas de los estudiantes.

Comunicación: se refiere a la interacción entre estudiantes y profesores, en lo referente a consultas, ejercitación, corrección, evaluación, etc.

En la Figura 21 podemos apreciar la interacción existente entre estos componentes.

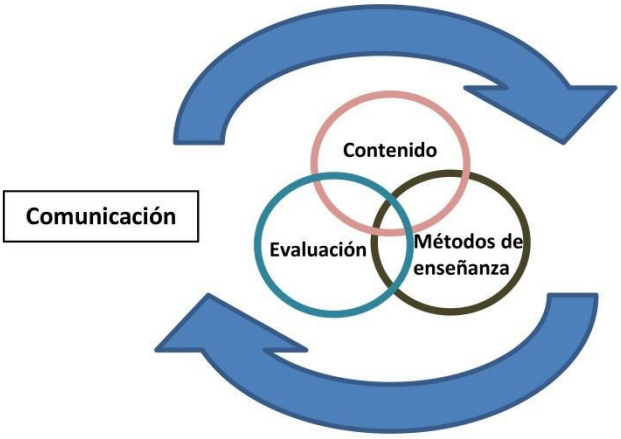


Figura 21: Modelo Interactivo de Aprendizaje (*Fuente: Adaptación de* Chassignol et al., 2018)

2.6.2 Sistemas de Tutoría Inteligente

A los modelos de aprendizaje interactivos ILE a los que se les aplican técnicas de IA se los denomina Sistemas De Tutoría Inteligente ITS, por sus siglas en inglés de *Intelligent Tutoring Systems*. Existe un importante consenso en la comunidad científico-educativa que indica y afirma que el uso de sistemas de tutoría inteligente ITS redunda en un mayor rendimiento en la asimilación y comprensión por parte de los estudiantes comparado al que se consigue utilizando los métodos tradicionales de aula basados en el estudio de materiales impresos (Ma et al. , 2014).

Variadas disciplinas científicas y campos del conocimiento han aprovechado los beneficios de la implementación de sistemas educativos montado sobre las técnicas de ITS. En la bibliografía podemos encontrar su aplicación en variados ámbitos académicos como Física (VanLehn K. et al. , 2002), Matemática (Sabo et al. (2013) , Lengua y Literatura (Mahmoud et al., 2016), Informática (Koedinger et al. (2013) y Medicina (Frize et al. (2000), entre otras. Mahmoud et al. describen muchos ejemplos de diferentes sistemas de tutoría, como The *AutoTutor* (Al Emran et al. , 2014), Why2-

Atlas (Sung et al. (2016), Beetle II System (VanLehn et al. , 2002) etc. La Tabla 1 presenta algunos modelos basados en ITS, las disciplinas en los que se aplican y sus estudiantes destinatarios.

Tabla 1 – Modelos basados en ITS (Fuente: Adaptación de Chassignol et al., 2018)

ITS	Disciplina	Destinatario
ActiveMath	Matemática	Secundarios
Beetle II System	Programación	Secundarios
EER-Tutor	Informática	Universitarios
MATHia	Matemática	Secundarios
The AutoTutor	Informática	Secundarios
Why2 Atlas	Física	Universitarios
COMET	Medicina	Universitarios
VIPER	Medicina	Universitarios

Los sistemas de tutoría inteligentes proporcionan instrucción y retroalimentación oportunas y personalizadas tanto para los alumnos como para los instructores. Están diseñados para mejorar el valor y la eficiencia del aprendizaje basándose en múltiples tecnologías informáticas, especialmente la inteligencia computacional, los sistemas expertos (basados en una serie de reglas concatenadas, desarrolladas en base al conocimiento aportado por especialistas en la disciplina o saber) y el aprendizaje estadístico .

2.6.2.1 Modelos de Sistema de Aprendizaje y Tutoría Inteligente

El modelo general de sistema de aprendizaje y tutoría inteligente, propuesto por primera vez por Peter Brusilovsky, profesor e investigador de la Escuela de Ciencias de la Información de la Universidad de Pittsburgh, se divide en dos módulos fundamentales: el módulo de sistema (que incluye el modelo de alumno, el modelo de enseñanza, el modelo de conocimiento y el modelo de interfaz) y el motor adaptativo inteligente o de tecnologías inteligentes (Brusilovsky et al., 2003).

***Modelo del alumno**:* Durante el proceso de aprendizaje se generan en tiempo real, y en simultáneo, datos multidimensionales sobre el comportamiento de los alumnos. En el sistema de aprendizaje adaptativo inteligente, el modelo de alumno es fundamental para mejorar la capacidad de aprendizaje. El primer nivel del modelo de aprendizaje se centra en el pensamiento y la capacidad de los estudiantes y se utiliza para evaluar sus habilidades de aprendizaje. El segundo nivel del modelo de estudiante mapea los puntos de conocimiento, para dar una imagen completa del dominio de los conocimientos de los estudiantes (Deloitte Res, 2019). El modelado de los datos del alumno -que refleja las relaciones entre los resultados del aprendizaje y una serie de variables que incluyen los contenidos y el material didáctico, los recursos de aprendizaje y los comportamientos de enseñanza- puede utilizarse para predecir el nivel de asimilación de los conceptos por parte de los estudiantes. Este tipo de análisis inteligente proporciona a los estudiantes y a los profesores recursos, actividades y hasta planes de estudio, adaptados a sus necesidades y a los logros de los alumnos. Contando con más información objetiva proporcionada por el procesamiento de estos datos, docentes y profesores pueden optimizar su planificación educativa, ajustando y administrando los contenidos y los planes educativos según el estado de aprendizaje de los alumnos.

***Modelo del campo de conocimiento*:** El modelo de campo de conocimientos describe la estructura de los conocimientos y establece un mapa conceptual que incluye contenidos y material de aprendizaje detallado, conocimientos especializados, errores regularmente cometidos por los alumnos, y reglas para solucionar fallas de conceptos o confusiones (Deloitte Res, 2019). Es mediante esta configuración y planificación que luego se podrán realizar todo tipo de constataciones y mediciones del grado de avance académico y pedagógico de los estudiantes, que son la esencia del modelo de enseñanza.

Modelo de enseñanza: El modelo de enseñanza define las reglas sobre cómo obtener acceso a cada campo de conocimiento basándose en la información proporcionada por el modelo del alumno. Combinando el modelo de campo de conocimiento y el modelo de alumno, el modelo de enseñanza permite a los profesores determinar qué estrategias y acciones de instrucción deben llevar a cabo. Los ITS deben estar siempre preparados para decidir "qué hacer a continuación", lo que viene determinado por el modelo de enseñanza.

Modelo de interfaz: La interfaz de usuario explica la actuación de los alumnos a través de múltiples mecanismos de entrada (voz, teclados y mouse) y proporciona la salida (textos, video, figuras, dibujos) a través de diferentes medios. Además de la función tradicional de la interfaz hombre-máquina, algunos sistemas también tienen otras funciones como la interacción mediante lenguaje natural, reconocimiento del habla y detección de las emociones de los alumnos.

Motor adaptativo inteligente: El motor adaptativo inteligente recoge información sobre las características de los alumnos y los objetos de conocimiento a partir de la base de datos de modelos de alumnos y la base de datos de modelos de campos de conocimiento y de enseñanza. A continuación veremos con más detalle las principales técnicas y tipos de modelos computacionales y algorítmicos utilizados para

esta tarea (considerada como esencia y fundamento de los ITS) que es la responsable de dotar a estos sistemas educativos de las características de adaptabilidad y criterio propio de un sistema inteligente.

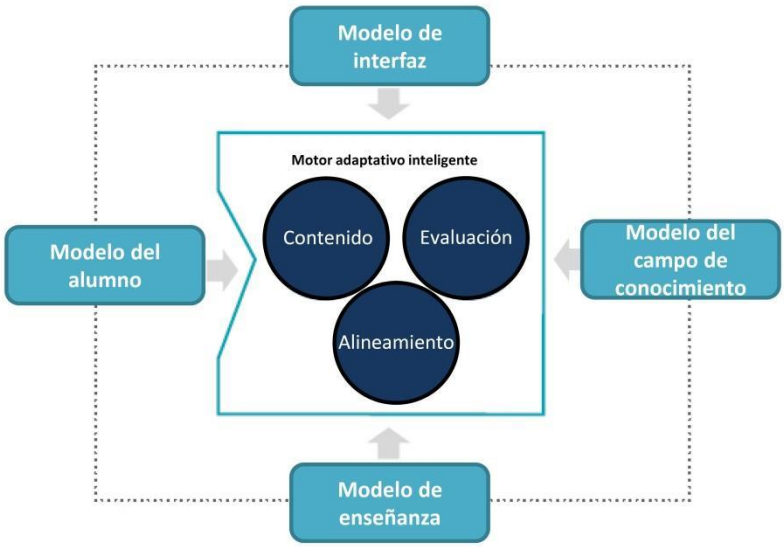


Figura 22: Modelo de referencia y componentes de un Sistema Adaptativo Inteligente. (*Fuente: Adaptación de Deloitte* Res, 2019)

2.7.3 Tecnologías Utilizadas En El Motor Adaptativo Inteligente

Las disciplinas comprendidas dentro del universo de la Inteligencia artificial más utilizadas como asistentes de sistemas educativos son el Aprendizaje Automático (*Machine Learning*) y su variante el Aprendizaje profundo o *Deep Learning*,

2.6.3.1 Aprendizaje Automático

Una de las aplicaciones fundamentales del aprendizaje automático es la recomendación del contenido más adecuado para cada alumno en función de sus

preferencias, hábitos de aprendizaje y estilo. El sistema educativo montado sobre modelos de aprendizaje automático registrará las preferencias de cada estudiante y realizará de manera autónoma las recomendaciones pertinentes. Basándose en el estado del aprendizaje y los objetivos curriculares, el sistema proporcionará automáticamente las lecciones y las actividades que mejor se adapten a las necesidades del estudiante, con el nivel de dificultad adecuado y en la secuencia correcta, para que éste no pierda la confianza ni se sienta frustrado, o desincentivado, por haberse fijado objetivos demasiado altos o demasiado bajos, respectivamente.

Los modelos de aprendizaje automático también pueden utilizarse para el análisis de la educación y el aprendizaje visto desde un enfoque pedagógico general. A estas áreas de aplicación se la denomina “Analítica del Aprendizaje”. Estos modelos miden y analizan el proceso y las actividades de aprendizaje, recopilando todo tipo de datos a partir de la experiencia de los estudiantes incluido el tiempo de aprendizaje, la duración y el rendimiento de las pruebas, seguido de un análisis de estos datos que permitirán el desarrollo de modelos de aprendizaje personalizados para diferentes alumnos. Estos modelos a su vez predicen y supervisan las puntuaciones de los estudiantes en los exámenes, que permiten tomar medidas y cursos de acción e intervención pertinentes para la mejor preparación de los alumnos y la formulación óptima de las evaluaciones.

La analítica del aprendizaje ayuda a desarrollar objetivos de aprendizaje personalizados, con incentivos adaptados a cada estudiante, a su vez proporcionando a los profesores datos detallados de los alumnos, como el tiempo que han invertido en cada actividad y su nivel de comprensión, ayudando así al sistema, y a los propios profesores, a mejorar los métodos de enseñanza.

2.6.3.2 Aprendizaje profundo

Los modelos de *Deep Learning*, entre muchas otras aplicaciones en la educación, nos ayudan a trazar el recorrido de estudio óptimo para que los estudiantes maximicen la eficiencia del aprendizaje. Este modelo recomienda la siguiente lección adecuada en función de los objetivos de aprendizaje preestablecidos y su estado actual de aprendizaje, y ajusta las lecciones en tiempo real en función de la evolución de su comprensión de los conocimientos. Después de recibir los diferentes tipos de datos de los estudiantes generados a partir de los contenidos de aprendizaje que se han presentado en clase, el sistema desarrollará perfiles de los alumnos que contemplen sus hábitos, intereses y métodos de aprendizaje, al tiempo que optimiza automáticamente su lógica de presentación y exposición.

2.6.4 Aplicaciones Fundamentales De La Inteligencia Artificial En Educación – Panorama a Futuro

A partir los ya mencionados y analizados Sistemas de Administración de la Educación (LMS) y de Tutoría Inteligente (ITS), surge la disciplina que trata sobre la integración de los ITS con el aprendizaje automático, denominada Minería de Datos Educacional, o EDM por sus siglas en inglés. Su definición formal es (Romero et al., 2010): “La EDM se ocupa de desarrollar, investigar y aplicar el aprendizaje automático, la minería de datos y los métodos estadísticos para detectar patrones en grandes colecciones de datos educativos que de otro modo serían imposibles de analizar“

Luego de relevar y analizar los diferentes sistemas y modelos en los que se aplica la IA a la educación, tanto mediante el uso de las técnicas del aprendizaje automático clásico como de las redes neuronales artificiales y el *Deep Learning*, ahora presentaremos el panorama a futuro. En su artículo de investigación Hernández-Blanco et al. (Hernández-Blanco et al. (2019), luego de una exhaustiva y detallada

revisión de artículos científicos presentados en diferentes revistas y congresos de *Educational Data Mining*, nos presenta las áreas que no han sido suficientemente explotadas, siendo por ende susceptibles de ser las destinatarias de futuras líneas de investigación. Las más destacables son las que detallo a continuación, siendo la última, "Creación de Material Didáctico". la que motiva y justifica la presente investigación y el consiguiente modelo educativo a desarrollar.

2.6.4.1 Análisis de redes sociales:

El objetivo es obtener un modelo de alumno que contemple las diferentes relaciones posibles entre los estudiantes, vistas desde la óptica de la teoría clásica de redes sociales, pudiendo establecer relaciones y conexiones de diferentes grados de afinidad en busca de patrones de toda índole.

2.6.4.2 Desarrollo de mapas conceptuales y curriculares:

Los mapas conceptuales son herramientas gráficas para organizar y representar el conocimiento. En esta categoría de aplicaciones el objetivo es desarrollar mapas conceptuales de diversos aspectos para ayudar a los educadores a definir y organizar los diferentes componentes del proceso educativo. Algunos ejemplos de mapas conceptuales son: la jerarquía de los temas en el material del curso, las relaciones entre las habilidades y los ítems de los exámenes, la correlación entre los ítems de los exámenes y los componentes del conocimiento, etc. Agrawal et al. (Agrawal et al., (2014)) presentan un navegador para estudiar libros de texto electrónicos que va creando referencias cruzadas que relacionan diferentes conceptos, permitiendo así correlacionar los varios aspectos y temáticas que abarca cada obra.

2.7.3.3 Creación de material didáctico:

El objetivo aquí es ayudar a los educadores a crear y desarrollar material didáctico que se adapte a las particularidades y alcances de las disciplinas y saberes a transmitir, utilizando para ello las funcionalidades y prestaciones de la IA, en particular las redes neuronales generativas. A su vez estos contenidos educativos generados de forma sintética se podrían adaptar en función del grado de avance propio de cada estudiante y realimentarse a partir de los resultados pedagógicos obtenidos.

2.7 Generación de Contenido Musical Mediante el uso de Redes Neuronales Profundas

2.7.1 Introducción

La relación entre la partitura y el sonido real es similar a la que existe entre el texto y el discurso. La partitura es una expresión visual altamente simbólica y abstracta que puede registrar y transmitir eficazmente sucesos de índole musical. El sonido es una forma de señal continua y concreta que codifica todos los detalles que podemos escuchar. Se pueden describir estos dos tipos de formas en diferentes niveles de abstracción, con la partitura en la parte superior y el sonido en la inferior. Así, el proceso de generación de música suele dividirse en tres etapas; en la primera etapa los compositores producen partituras; en la segunda los músicos ejecutan las partituras y le imprimen su propia interpretación y en la última etapa, la interpretación se convierte en sonido añadiendo diferentes timbres (instrumentos) y siendo percibida por seres humanos (oyentes). A partir de lo anterior, se divide la generación automática de música en tres niveles: el nivel superior corresponde a la generación de partituras, el nivel medio corresponde a la generación de interpretaciones y el nivel

inferior corresponde a la generación de audio. La generación de música de nivel inferior puede estar condicionada por los resultados de la generación de nivel superior, por ejemplo, la predicción de las características de la interpretación a través de las características de la partitura.

La diversificación de las tareas de investigación en generación musical ha ido en aumento en estos últimos años. Desde la generación inicial de melodías hasta la actualidad, la generación de partituras abarca la generación de música polifónica, la generación de acompañamientos, la transferencia de estilos, la generación interactiva, el *inpainting* musical, etc. La partitura tiene en cuenta características musicales como el tono, la duración y la progresión de acordes, y su representación es discreta, mientras que la interpretación implica una información más abundante sobre la dinámica y el tiempo, determinando diversas expresiones musicales (Shuqi, 2018).

El rápido progreso de las redes neuronales artificiales está difuminando poco a poco la frontera entre las artes y las ciencias. De hecho, hubo varios intentos de automatizar el proceso de composición musical mucho antes de la era de las redes neuronales artificiales (Yamshchikov, 2017). La primera obra musical generada por ordenador apareció en 1957 gracias a un programa de síntesis de sonido desarrollado por los Laboratorios Bell. "The Iliac Suite" fue la primera partitura creada por ordenador (García Salas, 2011), haciendo uso de modelos estocásticos (cadenas de Markov) para la generación, así como de reglas para filtrar el material generado según las propiedades deseadas. Iannis Xenakis, renombrado compositor de vanguardia, utilizó profusamente algoritmos estocásticos para generar sus composiciones. Koenig, otro compositor, implementó el algoritmo PROJECT1 en 1964, utilizando la composición en serie y otras técnicas (como la cadena de Markov) para automatizar la generación de música (Ames, 1987).

2.7.2 *Deep Music Generation*

Se denomina *Deep Music Generation* a la utilización de ordenadores de arquitecturas de redes de *Deep Learning* para generar música de forma artificial. Las redes neuronales recurrentes, RNN , son la arquitectura adecuada para el aprendizaje de datos secuenciales y es también el primero tipo de red neuronal utilizado para la generación de música. Ya en 1989, Todd (Todd, 1989) utilizó una RNN para generar una melodía monofónica por primera vez. Sin embargo, debido al problema del desvanecimiento del gradiente, es difícil para una RNN almacenar información histórica de larga data sobre las secuencias. Para resolver este problema, Hochreiter et al. (Hochreiter, 1997) diseñaron una arquitectura RNN especial-LSTM para ayudar a la red a memorizar y recuperar la información de la secuencia. En 2002, Eck et al. (Eck, 2002) utilizaron por primera vez la LSTM en la creación musical, improvisando música de blues con buen ritmo y una estructura razonable. Boulanger et al. (Boulanger-Lewandowski, 2012) propusieron el modelo RNN-RBM (Redes Neuronales Recurrentes y Máquinas Restringidas de Boltzmann) en 2012, que resultó ser superior al modelo tradicional de generación de música polifónica en varios datasets, pero a la que todavía le fue sumamente difícil capturar la estructura de la música con dependencias de largo plazo. En 2016, el equipo Magenta de Google Brain propuso el modelo RNN Melody (Waite, 2016), mejorando aún más la capacidad de RNN para aprender estructuras a largo plazo. Más tarde, Hadjeres et al. (Hadjeres, 2017) propusieron Anticipation-RNN, un novedoso modelo de RNN que permite aplicar restricciones posicionales definidas por el usuario. Johnson et al. (Johnson, 2017) propusieron TP-LSTM-NADE y BALSTM utilizando un conjunto de redes recurrentes paralelas.

2.7.3 Análisis de los Debilidades y Desafíos en *Deep Music Generation*

En este apartado se analizarán diferentes características propias del discurso y de la forma musical que según la literatura específica del tema son los que más dificultades y falencias han presentado en las obras musicales creadas de manera sintética. Estos ítems son naturalmente considerados por los compositores humanos, pero no se ha llegado a poder comprender la forma de representarlos mediante una estructura adaptativa basada en redes neuronales profundas . Al ser el objetivo de este trabajo el desarrollo de un modelo generador de contenido musical, es menester conocer cuáles son estas limitaciones e indicar posteriormente como repercutirán en su desarrollo, evaluación e implementación.

2.7.3.1 Motivos y repeticiones

La música generada mediante estas arquitecturas presenta la característica de carecer de estructura a largo plazo, es decir, de temas, motivos y patrones recurrentes. Las obras generadas por estos modelos se vuelven gradualmente aburridas pues no hay ningún nuevo estímulo externo durante el proceso de generación. El caso más extremo es que las notas generadas sean siempre las mismas. Cuando la longitud de la música aumenta, es difícil modelar la estructura local y la estructura global al mismo tiempo. Los patrones globales se refieren a estructuras largas que abarcan varios compases. Los patrones locales se refieren a elementos del pasado o incluso del pasado lejano, que se repiten y se desarrollan para crear contraste y sorpresa (Zhang, 2020). Aunque se ha trabajado mucho para generar piezas musicales más largas, aún se está lejos de alcanzar el objetivo de generar una longitud normal o estándar de una pieza musical promedio compuesta

por un ser humano. Hay relativamente pocos estudios centrados en las estructuras autorrepetitivas (Jhamtani, 2019). La generación de motivos de duración intermedia a larga (Mao, 2018) sigue siendo un área inexplorada.

2.7.3.2 Creatividad

La creatividad es la extrapolación de patrones atípicos más allá de la distribución observada. Sin embargo, los regímenes de aprendizaje automático actuales son capaces de manejar principalmente tareas de interpolación y no de extrapolación (Hakimi, 2020), lo que hace que la música generada tienda a no ser innovadora. Además, un porcentaje significativo de las secuencias generadas, a pesar de su similitud estadística con los datos de entrenamiento, son calificadas como erróneas, aburridas o incoherentes, cuando son evaluadas por oyentes humanos experimentados (Yamshchikov 2017.). La frontera entre la creatividad musical y la simple torpeza es muy incierta y difícil de cuantificar. Por ahora, hay relativamente pocas investigaciones centradas en la creatividad de la música generada por modelos de *Deep Music Generation* (Chen, 2018).

2.7.3.3 Representación

La mayoría de los estudios existentes sólo utilizan el tono y la duración para representar la música. Estas dos características del hecho sonoro no poseen la capacidad de poder representar adecuadamente la diversidad de símbolos musicales, como los vibratos y varios ornamentos. En el futuro, se espera que se propongan nuevas formas de representación que permitan plasmar en ellas la diversidad de símbolos musicales. A su vez , la representación de acordes es muy simple. La

mayoría de las investigaciones sólo se centran en la representación y generación de tríadas, ignorando otros tipos de acordes con mayor densidad de voces.

2.7.3.4 Aplicación

El objetivo de la composición algorítmica consiste en reproducir la capacidad real de los seres humanos de crear obras musicales, en lugar de una compleja imitación (Chen, 2018). A pesar del empeño de los investigadores en esta materia en desarrollar arquitecturas novedosas y en demostrar que la calidad de la música generada por estas arquitecturas es comparable a la música creada por seres humanos, la composición algorítmica sigue siendo una investigación de laboratorio, mientras que otras tecnologías de inteligencia artificial, como los agentes de búsqueda y el reconocimiento facial han tenido un desarrollo mucho mayor y se implementaron en variadas industrias (Chen, 2018). La aplicación del los modelos de generación de música en la vida real está limitada por dos razones principales. En primer lugar, la mayoría de los modelos imponen restricciones a la naturaleza de la entrada, pues es limitado en el número y el tipo de canciones y/o fragmentos musicales que conforman el corpus de entrenamienrto. En segundo lugar, los usuarios no pueden controlar el proceso de generación de forma detallada, lo cual es crucial para que el sistema sea útil en el contexto de la composición asistida por ordenador (Ens, 2020).

2.8 Métodos de evaluación

2.8.1 Introducción

La característica diferenciadora de los modelos de composición automática es que el ser humano no tiene interacción de ningún tipo con el sistema más allá de la previa

elección del conjunto de datos de entrenamiento y la configuración de los parámetros e hiperparámetros para conseguir un buen resultado. Es decir, el propio algoritmo devuelve una composición musical propia, sin intervención humana. Esto supone un problema a la hora de evaluar dichos modelos, ya que al contrario de lo que pasa en, por ejemplo, los problemas de clasificación, la música generada no es algo que pueda medirse numéricamente para optimizar algún funcional de performance como sería una función de pérdida o de error. Es por eso que después de obtener la pieza musical sintética, es necesario seguir un proceso de validación externo en el que sí intervenga un ser humano, sea por el diseño específico del mismo o por su participación en el experimento en el que se realice la evaluación.

El método más inmediato en el que se puede pensar para llevar a cabo una valoración de algo tan subjetivo como la música, es el Test de Turing (Oore, 2020). Como cabría esperar, el procedimiento de este sistema de evaluación consiste en presentar obras generadas artificialmente junto con otras creadas por humanos a varios oyentes. Estos deberán decidir si se trata de una obra generada por computador o compuesta por un ser humano. (Eck, 2002).

Sin embargo, no todos los métodos de evaluación están basados en la intervención humana en el sentido de tomar parte del proceso de juicio. Existen estudios que desarrollan métricas más empíricas que permiten evaluar de manera objetiva la calidad de una obra sintética utilizando para ellos diversos criterios que la definen. Yang en su trabajo (Yang, 2017) divide estas métricas en dos tipos fundamentales:

- *Medidas probabilísticas sin conocimiento de dominio musical*. Inicialmente concebidas para tareas de generación de imagen, también aparecen relacionadas con creación musical autónoma. Estas métricas se corresponden con cálculos de errores típicos de modelos estadísticos, como por ejemplo log-likelihood.

- *Métricas basadas en el conocimiento del dominio musical.* En este caso no se usarán métricas estadísticas clásicas ni tampoco métricas desarrolladas para probar la efectividad de un modelo en concreto. Se genera así un conjunto de métricas aplicables a la mayoría de los sistemas de generación automática musical, puesto que evalúan aspectos intrínsecos a la música como el tono, la frecuencia, etc.

2.8.2 Medidas para la Comparación de Distribuciones de Probabilidad

2.8.2.1 Introducción

Para iniciar nuestro análisis de puntuaciones y métricas de que permitan comparar dos distribuciones de probabilidad es preciso presentar un concepto fundamental de la teoría de la información, que es la entropía de una distribución. El objetivo principal de la teoría de la información es determinar la cantidad información que posee un conjunto de datos. La métrica más importante en la teoría de la información se llama *Entropía* (Shannon, 1948). La definición de Entropía para una distribución de probabilidad es:

$$H = -\sum_{i=1}^{N} p(x_i) \cdot \log p(x_i) \tag{14}$$

Si utilizamos logaritmos en base 2 para su cálculo podemos interpretar a la entropía como el número mínimo de bits necesarios para codificar la información contenida en esos datos. La clave del concepto de entropía se basa en que simplemente

conociendo el límite inferior teórico del número de bits que necesitamos, tenemos una forma de cuantificar exactamente cuánta información hay en nuestros datos. Sabiendo como cuantificar la entropía, se podría cuantificar cuánta información se pierde al sustituir la distribución observada por una aproximación parametrizada o por otra distribución, en el caso de este estudio la del contenido generado.

2.8.2.2 Divergencia Kullback-Leibler

La divergencia de Kullback-Leibler es una modificación de la entropía que permite calcular la divergencia o diferencia de una distribución de probabilidad con respecto a otra. Su desarrollo se debe a Solomon Kullback y Richard Leibler, y se la suele denominar “entropía relativa“ (Kullback, 1985).

La divergencia de Kullback Leibler de la distribución ***q*** con respecto a la distribución ***p*** se define mediante la diferencia entre los logaritmos de las probabilidades ponderadas por la distribución ***p***:

$$D_{KL}(p||q) = \sum_{i=1}^{N} p(x_i) \cdot (\log p(x_i) - \log q(x_i)) \tag{15}$$

La divergencia entonces es la esperanza matemática de la diferencia entre los logaritmos de las probabilidades de los datos en la distribución original y en la distribución aproximada. Si se lo analiza en términos de $\log_2$ se puede interpretar como la cantidad de bits que se pierden por aproximar a ***p*** con ***q***.

Sus fórmulas , dependiendo del tipo de variable ,son:

- *Discreta*

$$D_{KL}(P||Q) = \sum_i P_i \log(\frac{P_i}{Q_i}) \tag{16}$$

- *Continua*

$$D_{KL}(P||Q) = \int P(x) \log(\frac{P(x)}{Q(x)}) dx \tag{17}$$

La intuición de la divergencia KL es que cuando la probabilidad de un evento según P es grande, pero la probabilidad del mismo evento en Q es pequeña, hay una gran divergencia. Cuando la probabilidad según P es pequeña y según Q es grande, también hay una gran divergencia, pero no tanto como en el primer caso. Es a partir del logaritmo del ratio de las probabilidades *P(x)* y *Q(x)* que se calcula esta divergencia, siendo su valor la esperanza matemática de este logaritmo.

Esta métrica puede utilizarse para medir la divergencia entre distribuciones de probabilidad tanto discretas como continuas. Es importante destacar que la divergencia KL no es simétrica (Bishop, 2006), es decir mide cuanto diverge una distribución con respecto a otra que asume el rol de referencia, y por lo tanto no es una distancia.

$$D_{KL}(P|Q) \neq D_{KL}(Q|P) \tag{18}$$

2.8.2.3 Distancia de Jensen-Shannon

Una simetrización bien fundamentada de la divergencia KL es la distancia Jensen-Shannon (Lin, 1991), también llamada “discriminación capacitiva” (Sason, 2015). La distancia Jensen-Shannon es otra forma de cuantificar la diferencia (o disimilitud) entre dos distribuciones de probabilidad, pero con las ventajas de normalización y simetría. Esto significa que la divergencia de P con respecto a Q es la misma que la de Q con respecto a P. De hecho, su raíz cuadrada cumple los axiomas de una distancia.

Su fórmula es:

$$D_{JS}(P||Q) = \frac{1}{2}(D_{KL}(P||M) + D_{KL}(Q||M) \quad (19)$$

donde,

$$M = \tfrac{1}{2}(P + Q) \quad (20)$$

La distancia de Jensen-Shannon es más útil como medida de disimilitud ya que proporciona una versión suavizada y normalizada de la divergencia KL, con puntuaciones entre 0 (idénticas) y 1 (máxima diferencia), cuando se utiliza el logaritmo en base 2.

Capítulo 3 Hipótesis

3.1 Introducción

A partir del análisis del estado del arte de las aplicaciones de la Inteligencia artificial en educación, en especial del Machine Learning y del Deep Learning, hemos podido dar cuenta de las amplias posibilidades que ofrecen los modelos desarrollados a partir de estas técnicas, destacando a su vez los posibles campos de aplicación en los que aún no se ha explorado su potencial (Hernández-Blanco et al. ,2019). Es por ello que nos concentraremos en una de estas áreas específicas, que es el desarrollo de contenido didáctico, (sección 2.6.4.3) , analizando a su vez su implementación a partir de modelos de generación de contenido musical mediante redes neuronales profundas.

A su vez evaluaremos las limitaciones de estas metodologías de creación de contenido sintético musical ahora vistas no desde la emulación de la creatividad y el formalismo propio de una obra compuesta por un ser humano, sino a la luz de la generación de ejercitación musical en la subdisciplina audioperceptiva. Se selecciona esta ramas del saber musical como marco de referencia de esta investigación de generación de melodías para el entrenamiento musical por considerarla prototípicas del conjunto de saberes musicales indicados en la sección 1.2.

3.2 Requisitos asociados a La Generación De Ejercicios Musicales Con Redes Neuronales Profundas

En este apartado presentaremos cuales son los requisititos que debe cumplir el contenido generado por los modelos de ejercitación musical enfocados en la subdisciplina de audioperceptiva. Para ello se presentara en este apartado un detallado análisis del cumplimento de los requisitos y de los resultados experimentales

obtenidos a partir de la aplicación de los modelos de aprendizaje profundo para creación de obras musicales presentados en la sección 2.7.3 de este trabajo , ahora observados desde la óptica especifica de la creación de contenido musical para la ejercitación de la subdisciplina musical indicada.

Uno de los puntos considerados fue la estructura del contenido musical generado. Repeticiones y patrones de duración variada, y muchas veces extensa, son parte fundamental de la gran mayoría de las obras musicales. El oído entrenado detecta y valora esta referenciación del compositor a motivos e ideas musicales de largo alcance. Esta limitación no será de importancia en nuestro desarrollo pues no se pretende generar contenido musical de extensión media o alta, por lo que la ausencia de motivos de duración extensa no sería una limitante. Estas estructuras no son necesarias en los ejercicios , pues esencialmente se trata de generar una percepción melódica de corta o, a lo sumo, de mediana duración. Claramente se intentará evitar la generación de melodías autorrepetitivas sin musicalidad alguna, apelando a la utilización de recursos especialmente diseñado para ellos y de probada eficacia.

Específicamente se ha definido estadísticamente a la creatividad como la extrapolación de patrones anómalos más allá de la distribución de las melodías observadas en el corpus. Este requisito de creatividad no sería menester en su acepción más estricta en el caso de la generación de melodías para entrenamiento auditivo, pues justamente se quiere desarrollar una percepción de los componentes musicales esenciales. Es por ello que será aceptada una dosis de emulación, o citación, de fragmentos de frases presentes en el corpus. No obstante ello se deberá controlar la presencia de un cierto grado de creatividad que consiga que las melodías generadas no se tornen monótonas o predecibles. Para imprimirle la correcta dosis

de creatividad a los ejercicios generados se deberá apelar a la utilización de las técnicas de evaluación, tanto objetivas como subjetivas, anteriormente presentadas

Respecto a la interacción: en este entorno de generación de melodías no necesitaríamos interponer ninguna intervención al discurso musical en cuestión, pues el objetivo es el desarrollo de la percepción musical, siendo esta, por esencia, hechos musicales pasivos.

Los puntos analizados en el apartado de interpretación, como son los trinos, vibratos y ornamentos, aportan y contribuyen al enriquecimiento de las frases musicales. Sin duda el entrenamiento auditivo o interpretativo de las mismas sería muy valorado, pero no es menester en la etapa de formación del músico en la que se enfoca este trabajo, que es la del desarrollo de la audioperceptiva. A su vez al no ser necesario la modelización de ornamentos y recursos interpretativos espcíficos para estas áreas del entreamiento musical, es factibe la utilización de archivos que utilicen representaciones simbólicas sumamente probadas, como son las MIDI y XML que como hemos indicado sólo pueden plasmar en ellos aspectos básicos de la interpretación musical

Es en la evaluación del contenido generado para el fin de nuestro entorno de aprendizaje en el que tendremos que enfocarnos en particular. El desarrollo de la percepción exige que las frases posean las características esenciales del lenguaje del género musical del corpus. Sin duda, a la luz de lo visto en los últimos párrafos, los requisitos serán menores pero no por ello bajos.

3.3 Planteo general de la Hipótesis

A la luz de todo el análisis de la literatura tanto en Inteligencia Artificial aplicada a la Educación como en la de generación de contenido musical a partir de redes neuronales generativas, se evidencia el potencial innovador de la utilización de estas tecnologías y modelos de aprendizaje automático y generación para el desarrollo de un modelo educativo musical que subsane las limitaciones inherentes de los soportes pedagógicos clásicos y de los basados en sistemas de reglas. A su vez se ha analizado en detalle la viabilidad de este desarrollo al contemplar los limitantes y los requisitos que recaen sobre la creación de obras musicales utilizando los modelos presentados en la literatura científica del tema, para luego ahondar en las métricas que evalúen el cumplimiento de estos requisitos tanto en lo formal como en lo creativo.

Las hipótesis de trabajo para esta investigación son_:

1) Es factible generar cantidades virtualmente ilimitadas de contenido musical novedoso para ser utilizado con fines didácticos y de ejercitación.

2) Es viable un modelo generativo capaz de crear sintéticamente material de estudio y ejercitación musical original y creativo que respete las características esenciales del lenguaje musical y del género en el que se basa.

.

Capítulo 4. Metodología

4.1 Trabajo de Base

Tal cual hemos mencionado en el apartado de generación de contenido en lenguajes simbólicos, mediante la utilización de la técnica de muestreo por temperatura Boltzmann, Caccia et.al han demostrado que analizando diferentes temperaturas se pueden desarrollar modelos que mejoren el rendimiento generativo medidos en términos de creatividad y variedad. (Caccia et al., 2018). Recordemos que en el apartado 2.3 referido a las características de los métodos de aprendizaje tradicionales habíamos destacado como falencias inherentes a estos las restricciones en variedad y creatividad asociadas al soporte en el cual se alojan o al sistema de reglas a partir del cual son generadas. Es por ello que tomar como referencia lo demostrado por Caccia en NLP allana el camino para la confección de un sistema de evaluación del cumplimiento de ambos requisitos, variedad y creatividad, en el generador de contenido musical a desarrollar.

4.2 Métricas de Evaluación Objetivas y Subjetivas

Es fundamental analizar diferentes métricas, y arribar al valor óptimo de cada una de ellas que aseguren el cumplimiento de los objetivos de investigación planteados. Es por ello que analizaremos diferentes métricas presentadas en la literatura consultada, adicionando algunas otras generadas por el autor de esta tesis a tal efecto, y procurando hallar el valor de estas que nos aseguren generar melodías que cumplan los requisitos planteados.

Es por lo anterior que los experimentos que se llevarán a cabo en esta investigación procurarán determinar los valores de la temperatura de Boltzmann óptima en base al cálculo de métricas de evaluación objetivas y subjetivas especialmente diseñadas a

tal efecto, que permitan que el modelo calibrado con estos valores de temperatura cumpla con los requisitos de generar contenido de ejercitación musical en cantidades virtualmente ilimitadas, y percibido por oyentes entrenados como creativos y musicales.

4.3 Procedimiento

4.3.1 Introducción

Como se ha mencionado en las secciones anteriores los modelos de *Deep Learning* precisar para la determinación de sus parámetros un proceso de entrenamiento en base a un corpus previamente curado y pre-procresado. En nuestro caso el estudio a realizar se basará en un corpus de canciones del género Tango.

4.3.2 Diseño del Experimento para la Determinación de la Temperatura óptima

En la primera parte de la fase de experimentación se analizarán los resultados de las métricas obtenidas a partir de los diferentes corpus generados mediante la red neuronal recurrente LSTM a partir de diferentes valores del hiperparámetro temperatura de Boltzmann. La lista de los valores de temperatura con los que generarán los diferentes corpus de melodías es (0.1 , 0.2 , 0.3 , 0.4 , 0.5 , 0.6 , 0.7 , 0.8 , 0.9 , 1.0 , 1.1 , 1.2 , 1.3 , 1.4 , 1.5)

Para cada uno de los corpus se relevarán tres métricas por cada compás:

- Cantidad Alteraciones
- Salto Interválico Máximo
- Ratio de Cantidad de Duraciones Diferentes sobre Cantidad de Notas

A posteriori se calculará la distribución de masa de probabilidad empírica de estas métricas, quedándonos 15 distribuciones para cada variable, una por temperatura.

Para el cálculo de le temperatura óptima para cada una de estas métricas se procederá a calcular las divergencias Kullback-Leibler en ambos sentidos y a partir de ellas se obtendrá la distancia de Jensen-Shannon. Como ya se ha mencionado en la presentación de estas medidas de divergencia entre distribuciones ésta última es la más apropiada por poseer las virtudes de ser adimensional y porque su rango está acotado entre los valores 0 y 1.

Con el fin de arribar a la determinación de una temperatura de Boltzmann óptima que aglutine los tres resultados a los que se arribará, se promediarán sus respetivos valores, obteniéndose de esta manera una medida única que resuma los resultados del experimento.

4.4 Sobre el Corpus

4.4.1 Fuentes y características fundamentales

Para el entrenamiento de las redes mencionadas se utilizará un corpus de 250 canciones de Tango en total, extraídas de diferentes formatos (XML, partituras en papel, repositorios, etc.) . El detalle de cada una de esta presentaciones fue analizado en la sección de Vocabulario Musical y Representaciones de la presente tesis.

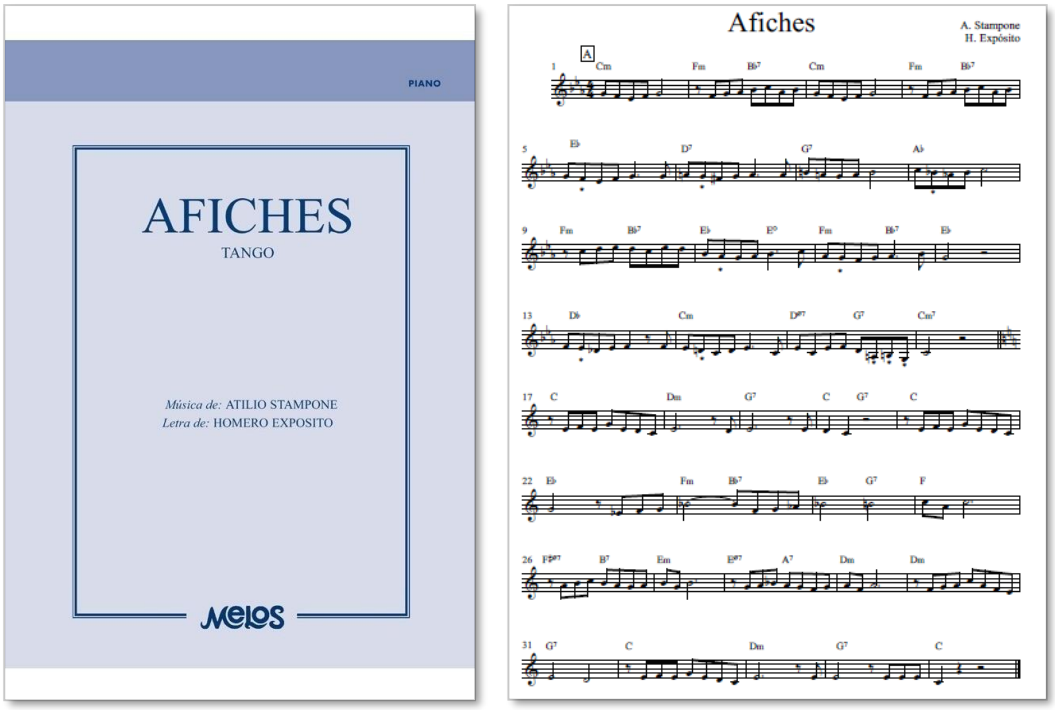


Figura 23. Partituras en formato papel

```xml
<?xml version="1.0" encoding='UTF-8' standalone='no' ?>
<!DOCTYPE score-partwise PUBLIC "-//Recordare//DTD MusicXML 3.0 Partwise//EN" "http://www.musicxml.org/dtds/partwise.dtd">
<score-partwise version="3.0">
 <work>
  <work-title>Caminito</work-title>
 </work>
 <identification>
  <creator type="composer">J. de Dios Filiberto
C.C Peñaloza</creator>
  <encoding>
   <encoding-date>2018-12-13</encoding-date>
   <encoder>hugos</encoder>
   <software>Sibelius 8.0.0</software>
   <software>Direct export, not from Dolet</software>
   <encoding-description>Sibelius / MusicXML 3.0</encoding-description>
   <supports element="print" type="yes" value="yes" attribute="new-system" />
   <supports element="print" type="yes" value="yes" attribute="new-page" />
   <supports element="accidental" type="yes" />
   <supports element="beam" type="yes" />
   <supports element="stem" type="yes" />
  </encoding>
 </identification>
 <defaults>
  <scaling>
   <millimeters>210</millimeters>
   <tenths>1200</tenths>
  </scaling>
  <page-layout>
   <page-height>1697</page-height>
   <page-width>1200</page-width>
   <page-margins type="both">
    <left-margin>109</left-margin>
```

Figura 24. Contenido de archivo en formato de representación simbólica. XML

4.4.2 Proceso de extracción de contenido

Ambas representaciones simbólicas utilizadas, partitura y XML, traen aparejadas sus particulares inconvenientes a la hora de extraer su contenido para ser procesado mediante herramientas de análisis musicológico. Por ejemplo el proceso de escaneado de una partitura en papel en una importante proporción de los casos no permite la captura clara y específica de todos los elementos gráficos utilizado en la simbología de partitura. Los softwares de escaneado realizan inferencias de forma automatizada a la hora de encontrarse ante una de estas figuras indescifrables acertando en algunas oportunidades y errando en otras. En estos últimos casos el proceso de corrección puede ser particularmente tedioso si el error de escaneo es frecuente, cosa que aconteció por ejemplo en algún tipo de partituras con el silencio de corchea. Este símbolo musical tiene una forma particular que lo asemeja a un 7, generando de errores obvios al equipararlo con, por ejemplo, el número de compas correspondiente.

Para subsanar estos errores se realizó un exhaustivo y minucioso trabajo de verificación de consistencia. Los errores mencionados por ejemplo alteraban la cantidad de notas que contenía el compás, dándonos por resultado compases de métricas extremadamente irregulares, pues poniendo como ejemplo el caso recién mencionado, estaríamos en presencia de un compás al que le faltaría una corchea, que en una en una obra que esté en 4/4 nos generaría un cambio de compas que sería considerado bastante inusual hasta en un compositor caracterizado por la innovación.

Algunos de los trabajos de preprocesamiento realizados sobre el corpus fueron: control de consistencia , creación de compases, análisis de figuras faltantes y/o erróneas, etc.

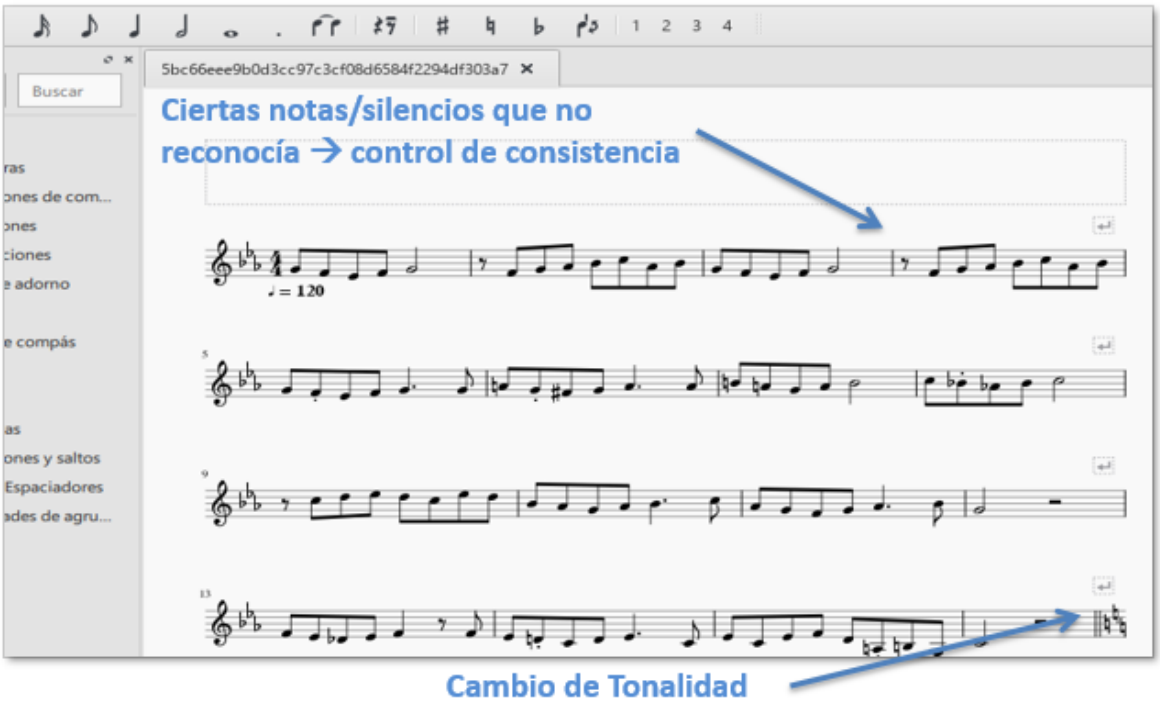


Figura 25. Ejemplo de control de consistencia

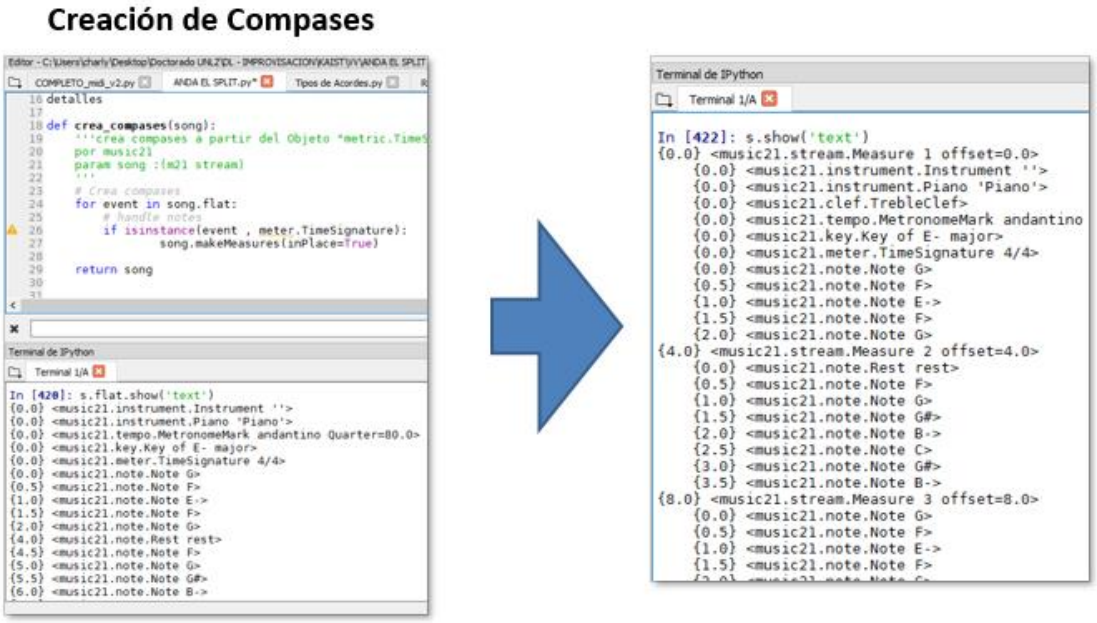


Figura 26. Creación de compases

4.4.3 Cambios de Tonalidad

En Deep Learning, en especial en visión computarizada, se utiliza la técnica de *data augmentation*. En el entrenamiento de modelos de visión computarizada se suele utilizar una enorme cantidad de imágenes para evitar así el sobreajuste. Ante la sumamente frecuente situación de no contar con una abultada cantidad de imágenes para cumplir con este cometido es que se utiliza esta técnica, basada en generar nuevas imágenes a partir de las existentes (Shorten , 2019). Las transformaciones se basan en los siguientes procesos:

- *Transformaciones geométricas* : traslación, rotación, reflexión.
- *Espacio de colores*: cambios de color, sustracción o combinación de algunos de los canales RGB
- *Corte de alguna parte de la imagen*
- *Inyección de ruido*: se adicionan pixeles aleatorios mediante una distribución gaussiana.

Algunos autores e investigadores de *Deep Music Generation* consideran útil realizar un tratamiento análogo a los recién presentados realizando transposición de las melodías del corpus de entrenamiento a las restantes 11 tonalidades. En el caso de este estudio se tomó el camino inverso, pues se consideró que este tipo de transformaciones utilizadas en imagen no serían útiles en la generación de música, pues esencialmente en música lo que se intenta modelar es la relación entre las notas (los intervalos que las separan entre sí), y este trabajo de aumentado por el mero hecho de transponer las melodías no aportaría ninguna novedad relacionada a las relaciones interválicas entre las notas. Recordemos que en esta instancia el generador trabajara solo con melodías y la esencia de una melodía es la relación interválica entre las notas que la componen.

Es Importante destacar aquí que las canciones de Tango suelen tener una parte B, a veces denominada puente, que representa un cambio fundamental y muy típico en las canciones pertenecientes a este tipo de tradiciones folclóricas. En una gran proporción de los casos estas segundas partes suelen cambiar de tonalidad por lo que se tornó necesario separar las diferentes partes de cada canción. Así fue que el trabajo de preprocesamiento involucró la generación de nuevos archivos en formato XML, a partir de los cuales se realizó posteriormente el entrenamiento. Este trabajo de identificación y posterior extracción de las partes componentes de una canción del corpus seleccionado fue fundamental para el proceso de uniformización de tonalidad llevado a cabo. Este consistió en transponer todas las partes a la tonalidad mayor de C y a la tonalidad de A menor, su relativa menor, pues ambas poseen la característica distintiva de no poseer ninguna alteración (ni sostenidos ni bemoles) en sus armaduras de clave. Estas tonalidades no representan particularmente nada especial a la hora del análisis y del entrenamiento de la red neuronal generativa, pues lo importante es que la red neuronal puede formarse una representación válida y precisa de los intervalos que componen una melodía. Sólo se las tomó como referencia por la mencionada característica distintiva de no poseer alteraciones .

Figura 27. Ejemplo de cambios de tonalidad

Es preciso mencionar aquí que el proceso de transposición a diferentes tonalidades si se utilizó a la hora de generar melodías luego de entrenado y seleccionado el mejor modelo. Esta transposición se realizó al azar, muestreando el intervalo de transposición. La utilidad de este procedimiento fue la de generar melodías variadas que no resultaran monótonas para el oyente que participaría de los test asociados de la evolución subjetiva, Huelga aclarar que al momento de entrar en producción el modelo se determinará la tonalidad de la melodía generada mediante un procedimiento prestablecido, que podrá ser al azar, o siguiendo algún camino didáctico.

4.4.4 Herramientas de preprocesamiento y curado del corpus

Para los procesos recién enumerados y descriptos se utilizó la Herramienta de procesamiento Musicológico Music21 (desarrollada por M.I.T.) y la aplicación de creación y edición de partituras MuseScore

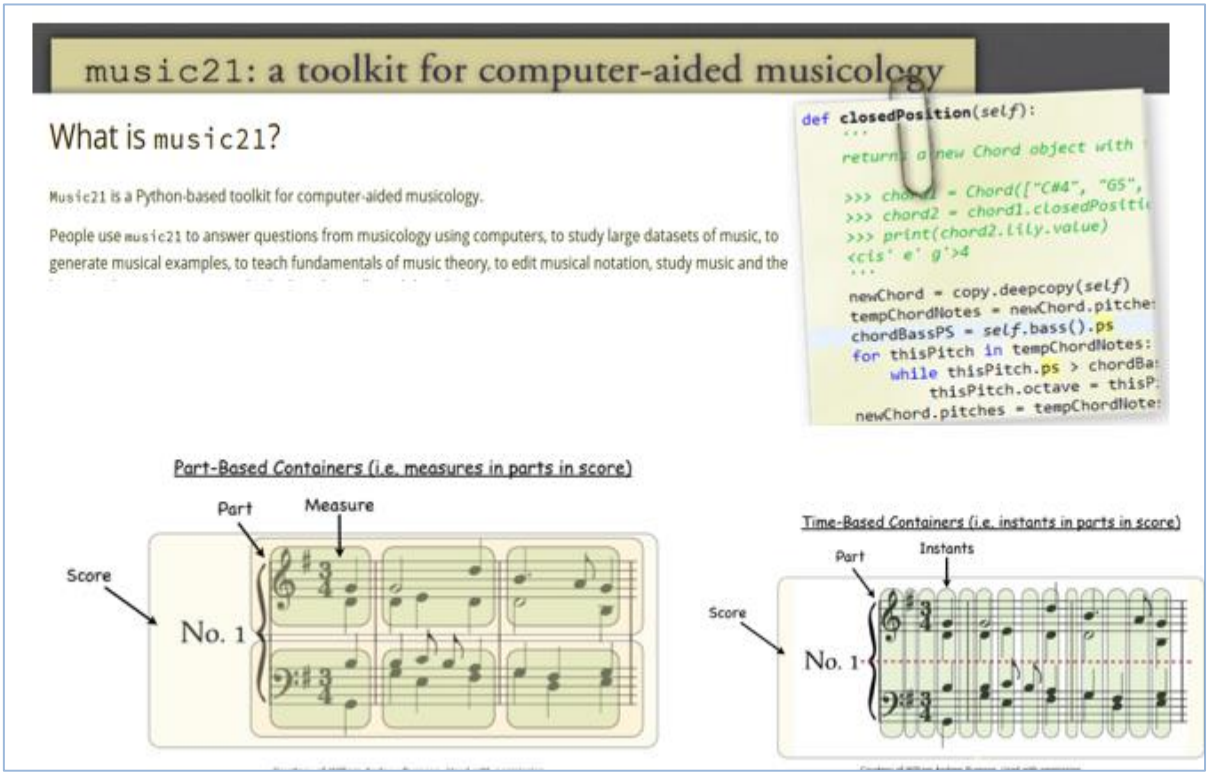


Figura 28. Herramienta de procesamiento Musicológico Music21 (desarrollada por M.I.T.)

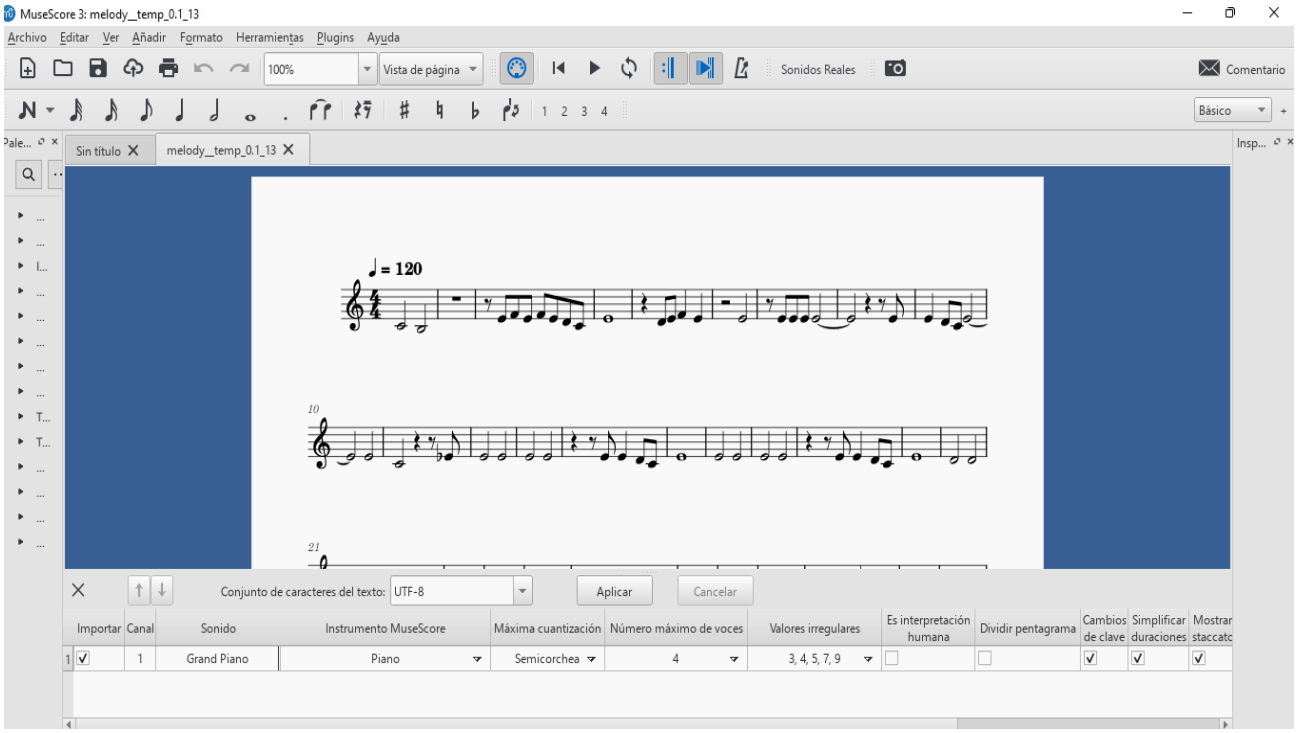


Figura 29 - Software de creación y edición de partituras MuseScore

4.5 Entrenamiento del Modelo de Red Neuronal Recurrente LSTM

4.5.1 Determinación del diseño básico

Para presentar la estructura del modelo elegido para el entrenamiento del modelo y su posterior puesta en producción, se ha seguido la metodología desarrollada por Briot (Briot et al, 2022) para definir la estructura y las características de los modelos de redes neuronales profundas para la generación música. Para ello se responderán una a una las preguntas que plantea Briot en sus metodología:

Objetivo

- ¿Qué contenido musical hay que generar?

Respuesta: melodías en estilo Tango

¿Para qué destino y para qué uso?

Respuesta: destinadas a la creación de ejercitación musical relacionada a las subdisciplinas Lectura y Audioperceptiva

Representación

- ¿Cuáles son los conceptos que se van a manipular?

Respuesta: notas obtenidas a partir de escaneo de archivos pdf y de archivos en formato XML

- ¿Qué formato se va a utilizar?

Respuesta: MIDI y XML

- ¿Cómo se codificará la representación?

Respuesta: en el estilo de DeepBach (Briot et al, 2022), con un valor discreto para la nota siguiendo la nomenclatura del protocolo MIDI, un símbolo “r” para indicar un silencio, un signo “_”, que indica el agregado de una semicorchea (0,25 de negra, “quarterLength”) a la nota anterior. y un signo “/” para indicar separación entre canciones (fin y comienzo) o relleno de semillas .

Arquitectura

- ¿Qué tipo de red neuronal profunda se va a utilizar?

Respuesta: : una red neuronal recurrente, del tipo LSTM

Desafío

- ¿Cuáles son las limitaciones y los retos pendientes?

Respuesta: limitaciones de capacidad de procesamiento. Esto se debe a que se ha realizado la gran mayoría del entrenamiento y de las pruebas utilizando plataformas de acceso libre como Google Colab, procurando contar con la posibilidad de usar procesadores gráficos (GPU) . Para las fases finales y el entrenamiento definitivo de los modelos y las pruebas finales se contrataron servicios de Amazon SageMaker.

4.5.2 Arquitectura

El modelo es del tipo sequence-vector. Este recibe como entrada una secuencia de datos y devuelve su predicción recién al llegar al último time-step de la secuencia. A continuación, en la figura 30 puede verse la arquitectura implementada.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, None, 32)]        0

 lstm (LSTM)                 (None, None, 256)         295936

 lstm_1 (LSTM)               (None, 256)               525312

 dropout (Dropout)           (None, 256)               0

 dense (Dense)               (None, 32)                8224

=================================================================
Total params: 829,472
Trainable params: 829,472
Non-trainable params: 0
```

Figura 30 . *Summary* de Tensorflow de la arquitectura de red LSTM utilizada

Los hiperparámetros escogidos son:

```
LOSS = "sparse_categorical_crossentropy"
LEARNING_RATE = 0.001
EPOCHS = 90
BATCH_SIZE = 64
DROPOUT_RATE = 0.2
```

Y aquí la presentación en formato de diagrama de flujo:

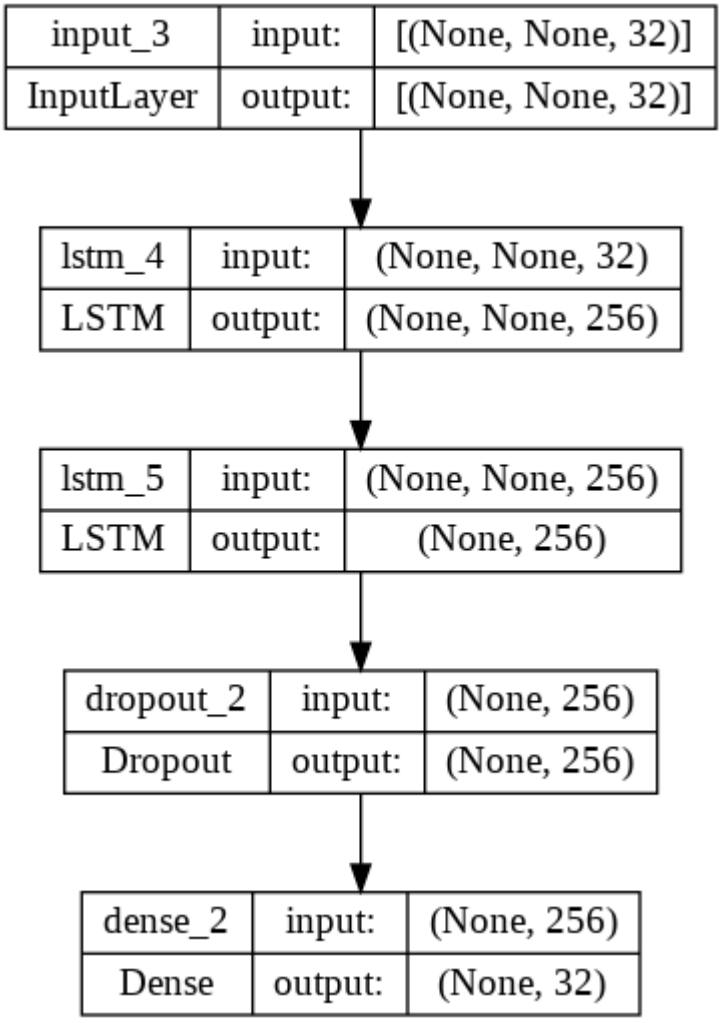


Figura 31 . *Flowchart* de Tensorflow de la arquitectura de red LSTM utilizada

En primer lugar se dispone una capa recurrente con 256 celdas LSTM de tipos *sequence-sequence*. Recibe una secuencia de entrada y a cada time step devuelve el resultado a la capa posterior. La segunda capa, también formada por 256 celdas LSTM, es de tipo sequence-vector. Además esta capa cuenta con una capa de dropout con valor 0, 2.

A continuación se dispone una capa densa con 256 unidades, con función de activación del tipo ReLu. Por último, se dispone una capa densa con número de neuronas igual al total del vocabulario utilizado. La función de activación de esta última capa será una Softmax.

En resumidas cuentas, el sistema recibe una serie de notas a lo largo del tiempo y deberá clasificar la salida en una de las notas vocabulario indicado.

4.5.3 Proceso de Entrenamiento

Una vez diseñada la arquitectura, es el momento de comenzar el entrenamiento de la red. Para ello, se deben preparar los datos de entrenamiento y adecuar su estructura para ser procesados por la red.

Para el proceso de codificación de los datos se emplea, como se ha indicado, la estrategia de valores numéricos discretos y símbolos “r” y “_” para silencios y prolongación de nota respectivamente, y “/” para separación entre canciones o relleno de semillas. Al final de cada canción se incluirá una serie de estos signos de separación en una cantidad determinada por el largo de la secuencia (en nuestro caso 64). Cada uno de los diferentes tokens del conjunto de datos será mapeado a un entero diferente. El conjunto de valores será normalizado previamente a la ingesta por

parte del modelo. El tamaño del vocabulario se determina luego de preprocesar todas las canciones válidas del dataset de entrenamiento, quedando 47 símbolos diferentes. En los anexos se detallarán todos los procesos llevados a cabo para el armado de las secuencias de entrada.

El modelo es tipo *sequence-vector*. Atendiendo a esto, los datos de entrada serán entonces vectores con cierta longitud, determinada por el parámetro *sequence length*, que en nuestro caso es de 64 elementos , y los datos del target, que será el elemento posterior a dicha secuencia.

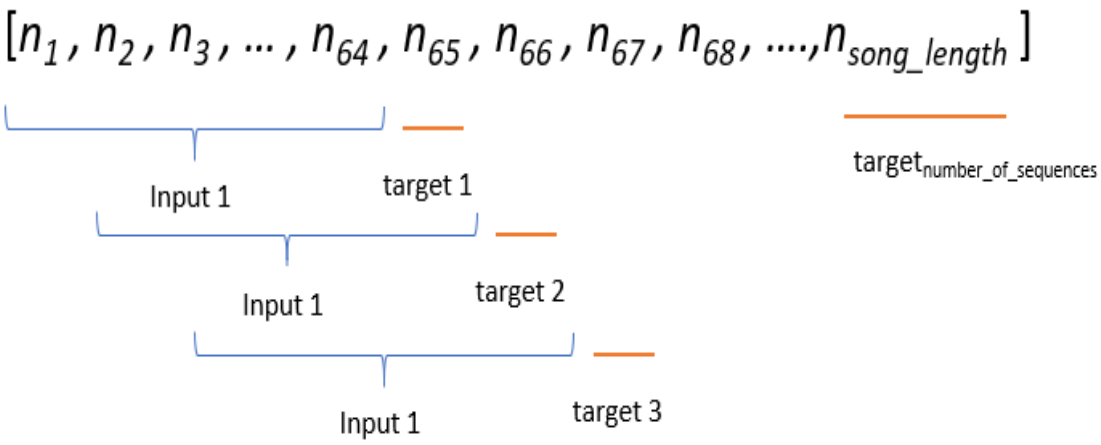


Figura 31 : Ejemplo datos de entrenamiento con longitud de secuencia igual a 64.

La Figura 31 muestra un ejemplo de estructuración de datos para el proceso de entrenamiento con el hiperparámetro de longitud de secuencia igual a 64. Como se ve, se toman los primeros 64 elementos del total de datos disponibles y siendo 65avo el que se dispone como valor de salida deseado. La siguiente secuencia de entrenamiento se construye desplazando todo un elemento a la derecha. Los secuencias se empaquetarán en minibatches de procesamiento. El tamaño del batch se determinó en 32 secuencias.

4.6 Fase de generación

Una vez que la red fue entrenada se comienza con la generación de contenido. Como se ve en la Figura 31, la red recibe una secuencia de la misma longitud que en la fase de entrenamiento, es decir, de 64 elementos y generar un elemento extra que se añadirá a dicha secuencia. En el siguiente paso de generación, la secuencia recibida por la red contiene los últimos 64 elementos, siendo el último el generado y excluyendo al primero de la secuencia anterior. Mediante esta aproximación, se generarán secuencias novedosas de longitud a determinar en base al modelo entrenado.

Para iniciar este proceso se le ingresa una secuencia de arranque llamada, semilla, o *seed* . la semilla tiene una longitud de 64 notas. Si se excediera de esta longitud, se la trunca, y si fuera menor, se le adicionan notas según un criterio preestablecido.

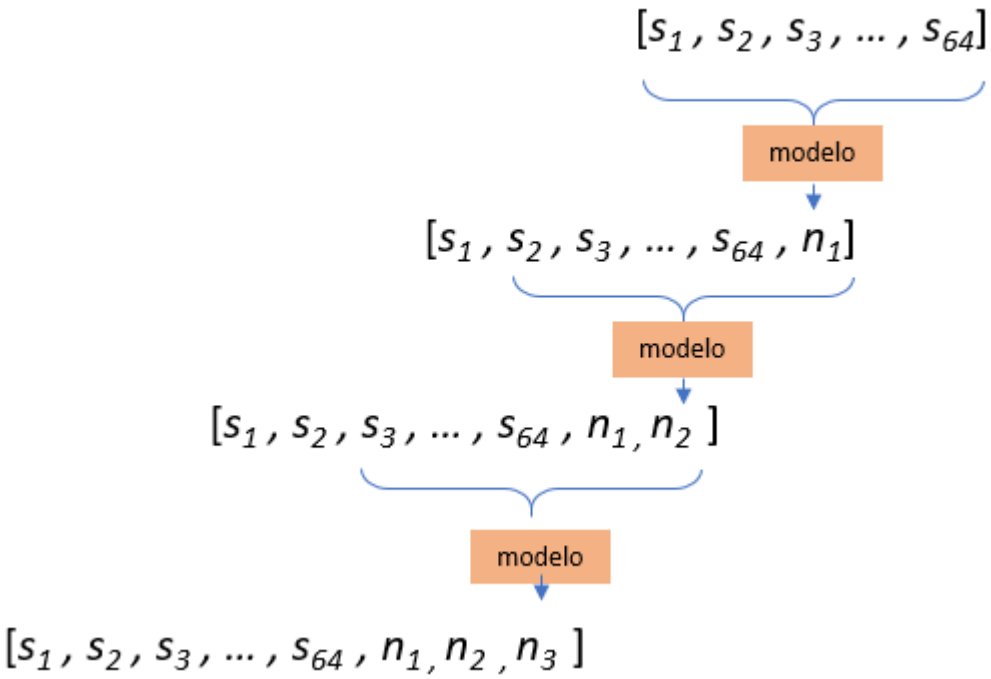


Figura 32: Gráfico representativo del proceso de generación a partir de una semilla (seed) con la arquitectura propuesta.

Y luego de 64 elementos creados el modelo ya está generando nuevos elementos tomando como inputs elementos sintéticos autogenerados.

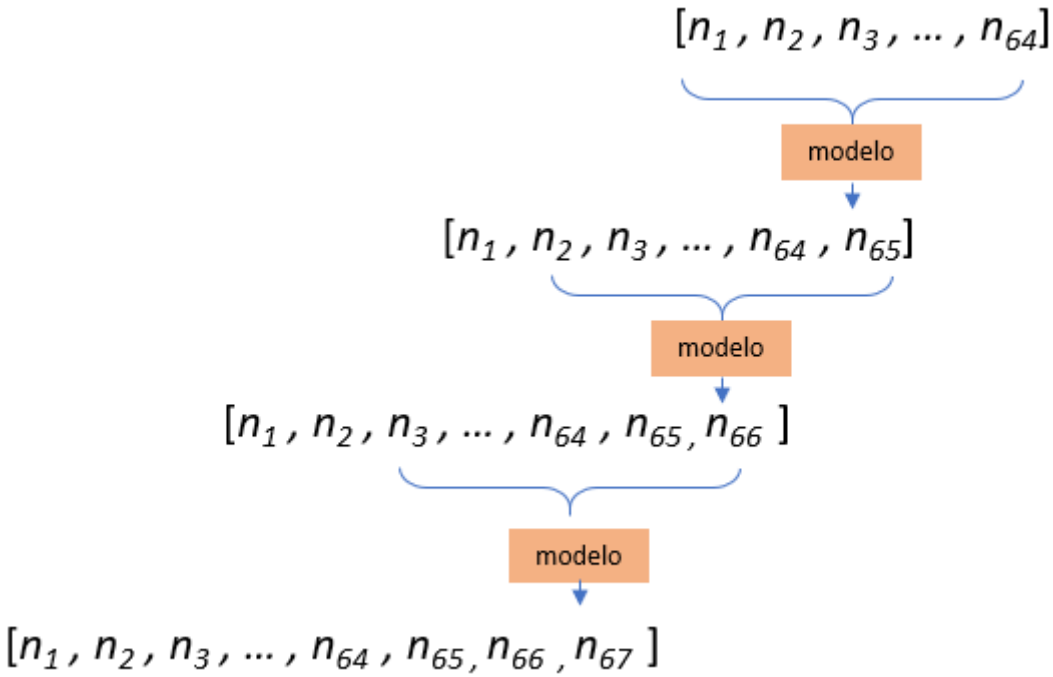


Figura 33: Gráfico representativo del proceso de generación a partir de tokens ya generados con la arquitectura propuesta

4.7 Creación de melodías a diferentes temperaturas

Como se ha explicado en el apartado del diseño experimental, con el objeto de calcular las métricas que posteriormente determinarán el valor óptimo de temperatura con el cual crear melodías sintéticas que posean las características requeridas, se debe generar una cantidad considerable de melodías.

Para ello se configuró la rutina generadora de melodías para que cree trechos musicales cuyo final posea las características apropiadas aprendidas a partir del corpus de entrenamiento. A tal efecto el modelo toma el símbolo de barra (“/”), que se ha utilizado para rellenar las ubicaciones en la secuencia de 64 notas tanto al final de cada canción del corpus de entrenamiento, también para rellenar el comienzo de las semillas con las que se inicia el proceso de generación.

Para cada una de las 15 temperaturas de la lista mencionada en el apartado de diseño experimental (de 0.1 a 1.5, con paso de 0.1) se genera un corpus de 200 canciones . El final de una canción queda determinado con la aparición de un carácter “/”, que es el utilizado justamente para delimitar las canciones entre sí en el corpus. Si el modelo genera este carácter , es que aprendió del entrenamiento que es en ese token en el que debe finalizar la canción. De esta manera se crearon las 200 canciones para cada temperatura, arrojando la siguiente cantidad de compases en total:

Tabla 2

Temperatura	Cant. Compases generados
Temp 0,1	37711
Temp 0,2	41941
Temp 0,3	32796
Temp 0,4	31395
Temp 0,5	23326
Temp 0,6	27236
Temp 0,7	28029
Temp 0,8	32113
Temp 0,9	31646
Temp 1,0	37931
Temp 1,1	27032
Temp 1,2	31172
Temp 1,3	36556
Temp 1,4	25669
Temp 1,5	32463

Todos los análisis sobre métricas objetivas se realizarán a nivel compás, o sea que se tomará como unidad de análisis para el relevamiento del valor de la métrica a analizar esta unidad de contenido musical. A tal efecto, posteriormente a la generación de las canciones se escribieron diferentes módulos en Python con el fin de generar listas que contengan como elementos los miles de compases generados en las 200

canciones de cada corpus (recordar que se genera un corpus por cada una de las 15 temperaturas)

4.8 Evaluación del Contenido Musical generado

4.8.1 Introducción

Partiendo de las premisas y de los lineamientos del diseño experimental establecido, a continuación se presentan los resultados obtenidos a partir de los 15 corpus generados, uno para cada temperatura.

Considerando como unidad de análisis al compás musical (en nuestro caso de 4/4, o sea 4 negras por compás) se calcularán las distribuciones empíricas de masa de probabilidad de las siguientes métricas:

- Cantidad Alteraciones (variable cuantitativa discreta)
- Salto Interválico Máximo (variable cuantitativa discreta)
- Ratio de Cantidad de Duraciones Diferentes sobre Cantidad de Notas (variable cuantitativa continua, del tipo proporción)

4.8.2 Evaluación Objetiva

4.8.2.1 Cantidad de alteraciones por compás

Unidad de análisis: compás

Variable a analizar: cantidad de alteraciones.

Tamaño de muestra : corpus de entrenamiento , 30512 compases. Corpus generados, aprox. 35000 por temperatura

Descripción del proceso de cálculo de la métrica

Hemos tomado como subdivisión y base de la grilla para representar simbólicamente las notas musicales del corpus a la semicorchea . A su vez hemos filtrado las canciones dejando sólo las que están en compas de 4/4 por lo que un compás se compone de 16 subdivisiones. El sistema de representación simbólica elegido exige que cada una de las subdivisiones posea un valor: si es el ataque de una nota será su valor en notación MIDI, si es el inicio de un silencio una “r”, y si es la continuación de la duración de una nota o un silencio (pues esta tiene una duración mayor a una semicorchea) será un signo “_” .No se contempla el signo “/” pues justamente indica el fin de una canción y no formara, por definición, parte de ningún compas analizado.

El espacio muestral de la variable a modelar será (0,16).

Se determinará la función empírica de probabilidad de esta variable a partir del análisis de los corpus generados para cada temperatura.

En el siguiente gráfico se indican las alteraciones para los compases de un trecho de un Tango.

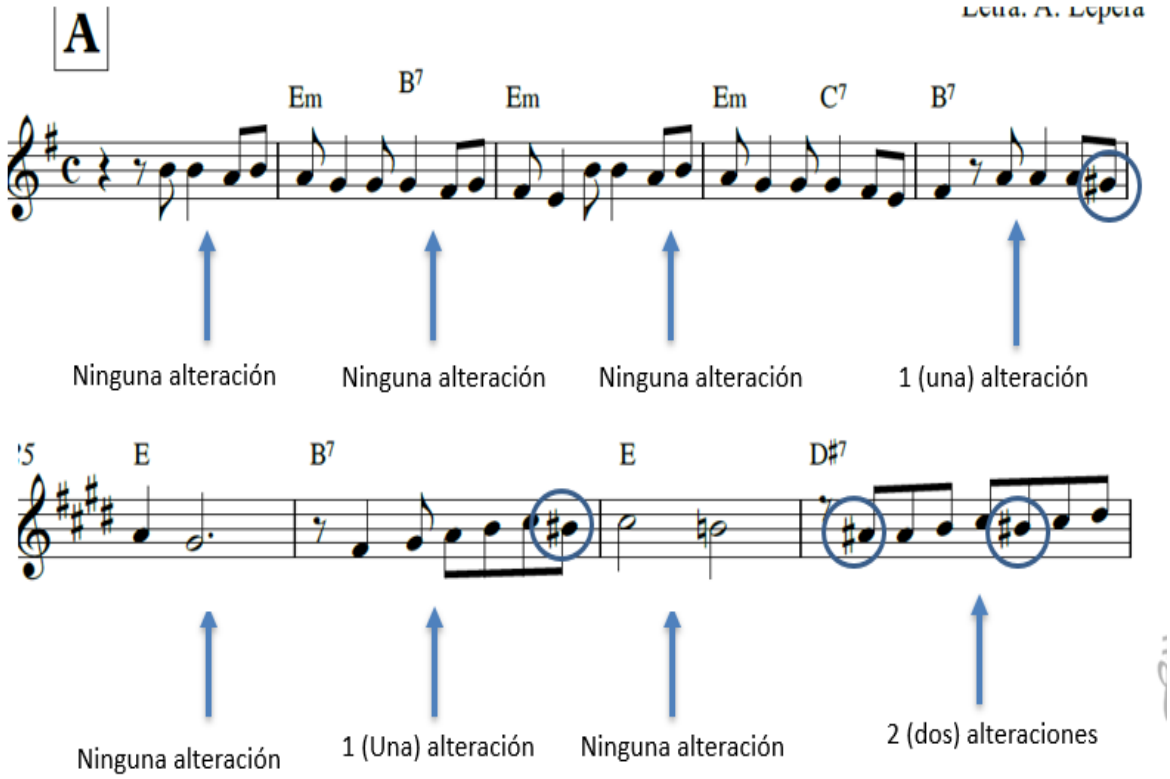


Figura 34: Extracto de partitura de un Tango en el que se explicitan las cantidades de alteraciones por compás.

4.8.2.1.1 Distribuciones empíricas de Masa Probabilidad de la métrica cantidad de alteraciones en el corpus y para frases musicales generadas con distintas temperaturas

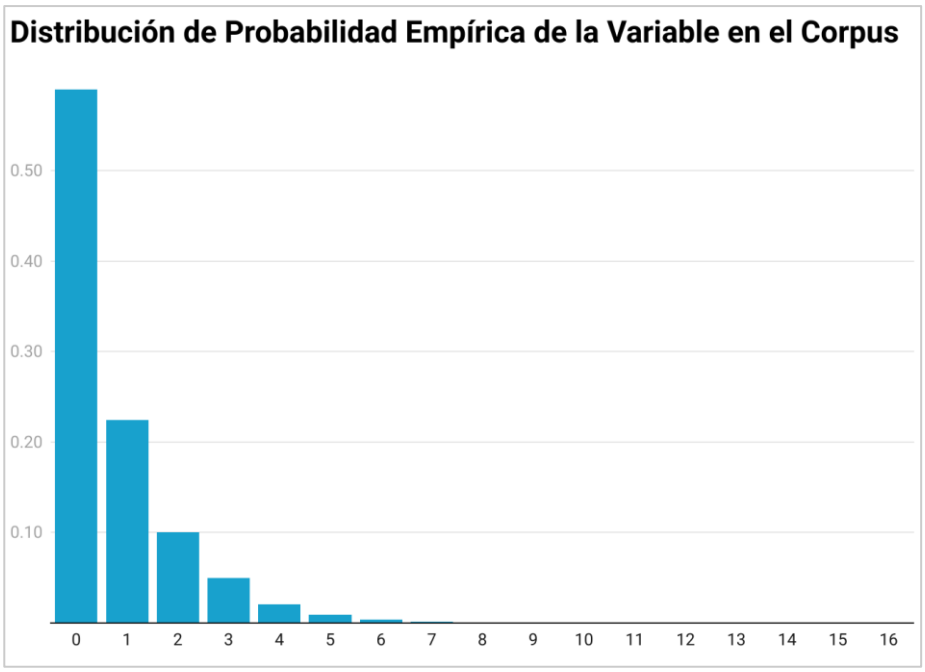


Figura 35. Distribución empírica de la variable cantidad de alteraciones en el corpus de entrenamiento

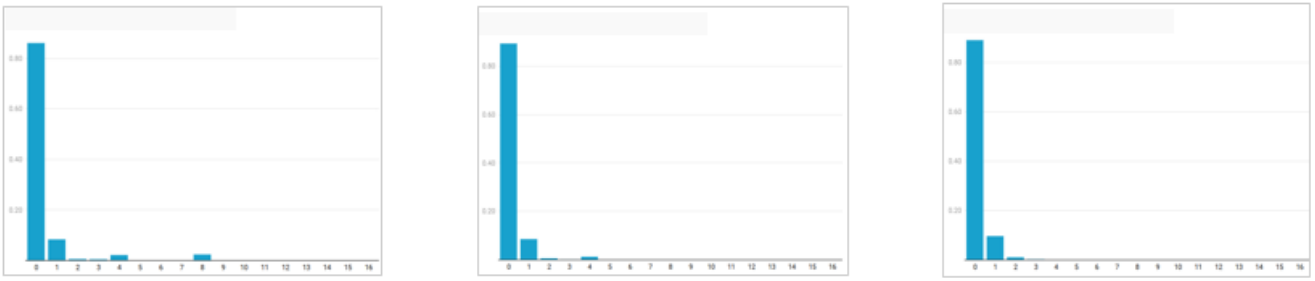


Figura 35 . Distribución empírica de la variable cantidad de alteraciones para temperaturas 0,1 ; 0,2 y 0,3

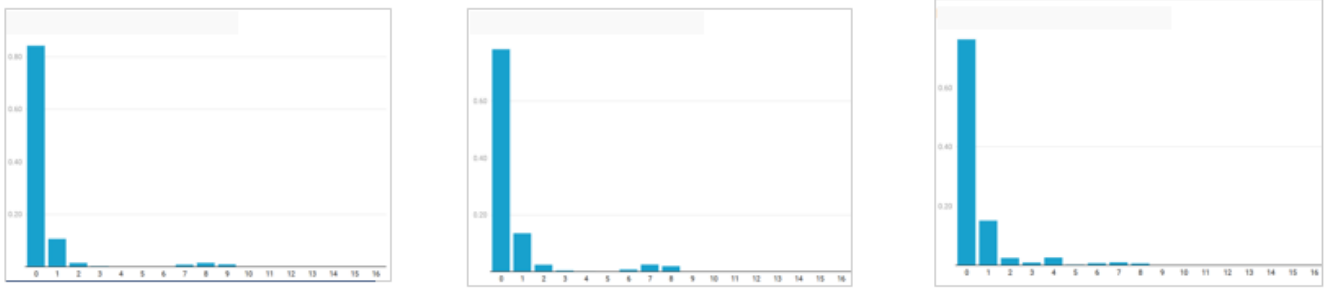


Figura 36 . Distribución empírica de la variable cantidad de alteraciones para temperaturas 0,4 ; 0,5 y 0,6

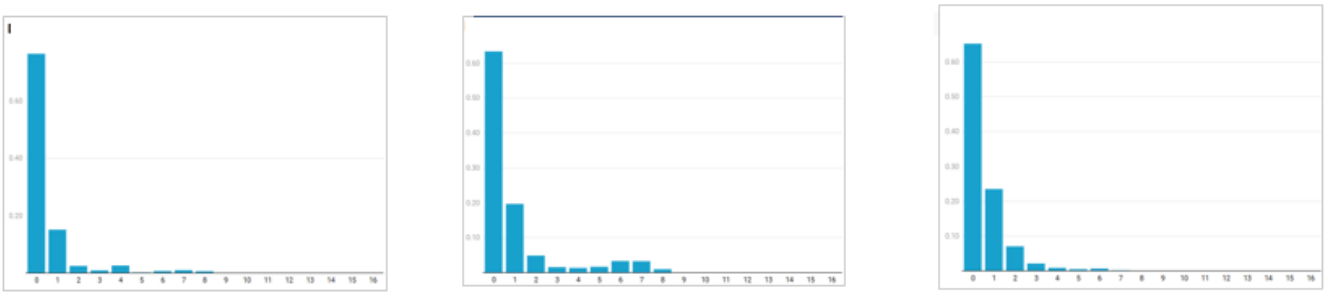


Figura 37 . Distribución empírica de la variable cantidad de alteraciones para temperaturas 0,7 ; 0,8 y 0,9

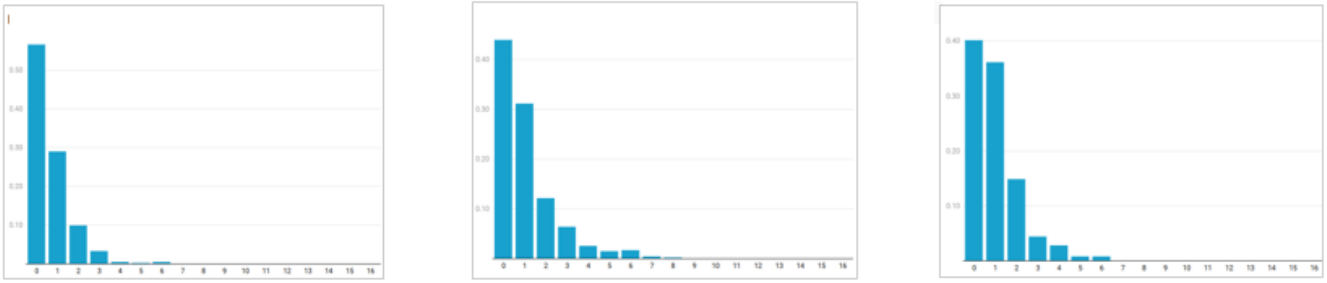


Figura 38 . Distribución empírica de la variable cantidad de alteraciones para temperaturas 1,0 ; 1,1 y 1,2

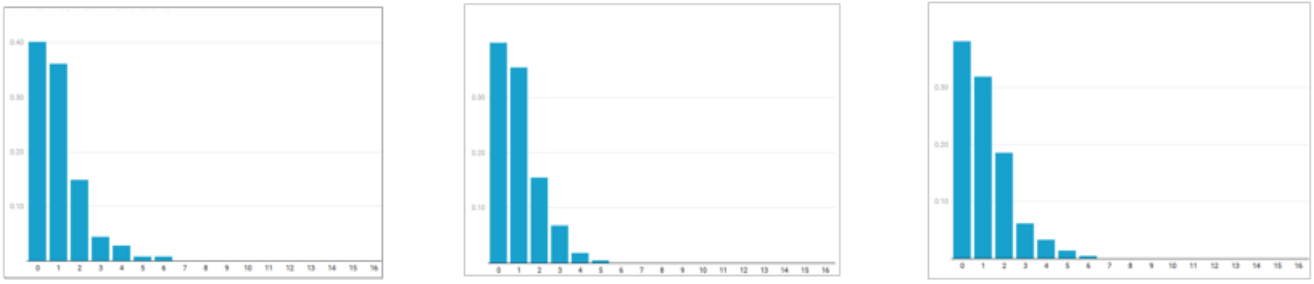


Figura 39 . Distribución empírica de la variable cantidad de alteraciones para temperaturas 1,3 ; 1,4 y 1,5

```
def distrib_prob_binomial(df, var_bernoulli, exito, cant_clases):
    mappings_1 = {}

    for i in range(len(cant_clases)+1):
        mappings_1[i] = 0


    #recorre las canciones
    for i in range(len(df['Song_ID'].unique())):   # i=0
        print("Analizando canción: " + df.loc[df['Song_ID']==i, 'Tema'].iloc[0])


        #recorres los compases de esa canción i     i=0  j=1
        for j in range(len(df.loc[df['Song_ID'] == i]['Compas'].unique())) :
            #inicializa la variable binomial que va acontar los exitos en la variable bernoulkii
            x_binomial = 0

            #recorre las notas de ese compas j de la canción i   i=0 j=1  k=1
            for k in range(len(df.loc[(df['Song_ID'] == i) & (df['Compas'] == j+1)])):

                #recorro el compás y actualizao el diccionario de esa variable
                 variable = df.loc[(df['Song_ID'] == i) & (df['Compas'] == j+1) , var_bernoulli].iloc[k]
                 if variable == exito:
                    x_binomial += 1


            #actualiza el mapping sumandole una frecuencia al valor de la variable binoliam observadop
            mappings_1[x_binomial] = mappings_1[x_binomial] + 1


    total_compases = 0
    distr_prob = []

    for i in sorted(mappings_1):
        total_compases += mappings_1[i]

    for i in sorted(mappings_1):
```

Figura 40 – Extracto del módulo de Python utilizado para calcular las métricas para cada temperatura

4.8.2.2 Salto Interválico Máximo por Compás

Unidad de análisis: compás

Variable a analizar: salto máximo medido en semitonos

Tamaño de muestra : corpus de entrenamiento , 30512 compases. Corpus generados, aprox 35000 por temperatura

Descripción del proceso de cálculo de la métrica

Para determinar el salto interválico medido en semitonos calcularemos la diferencia medida en valores MIDI de las notas más alta y más baja.

El espacio muestral determinado será de 0 a 25 semitonos.

Se determinará la función empírica de probabilidad de esta variable a partir del análisis de los corpus generadas para cada temperatura.

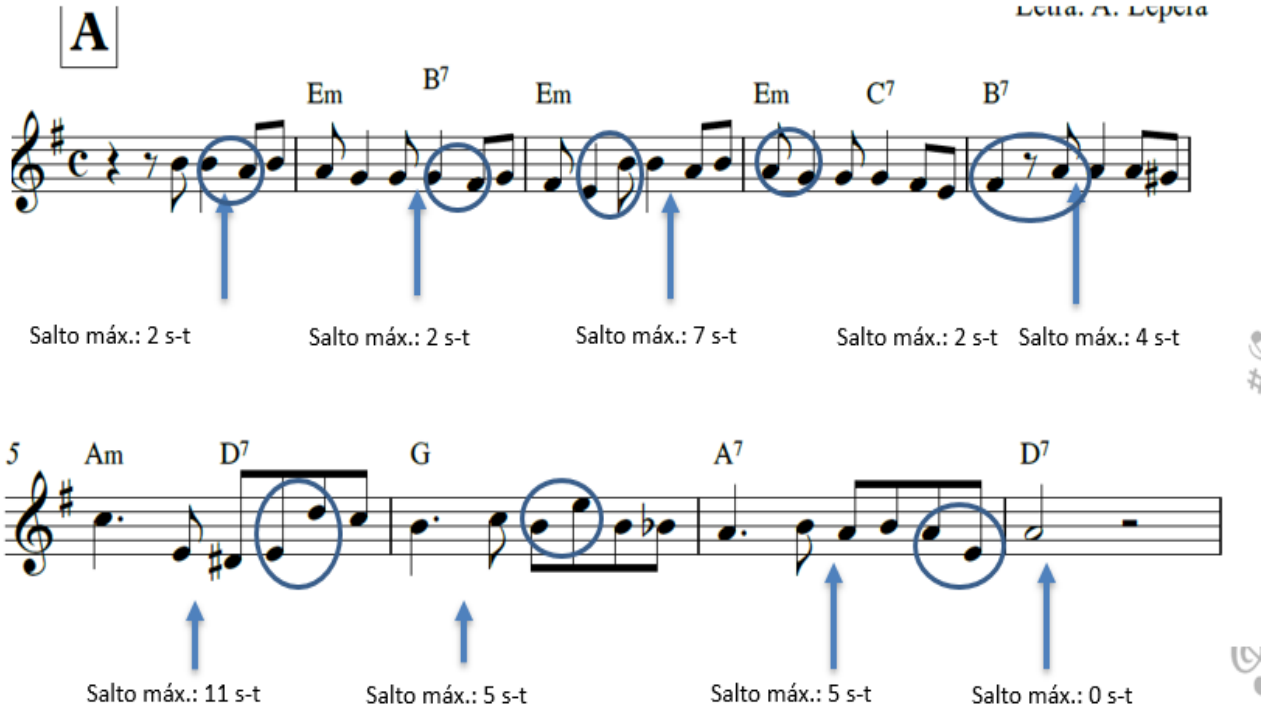


Figura 41: Extracto de partitura de un Tango en el que se explicitan Salto Interválico Máximo por Compás

4.8.2.2.1 Distribuciones de Masa Probabilidad empírica de la métrica en el corpus y para frases musicales generadas con distintas temperaturas

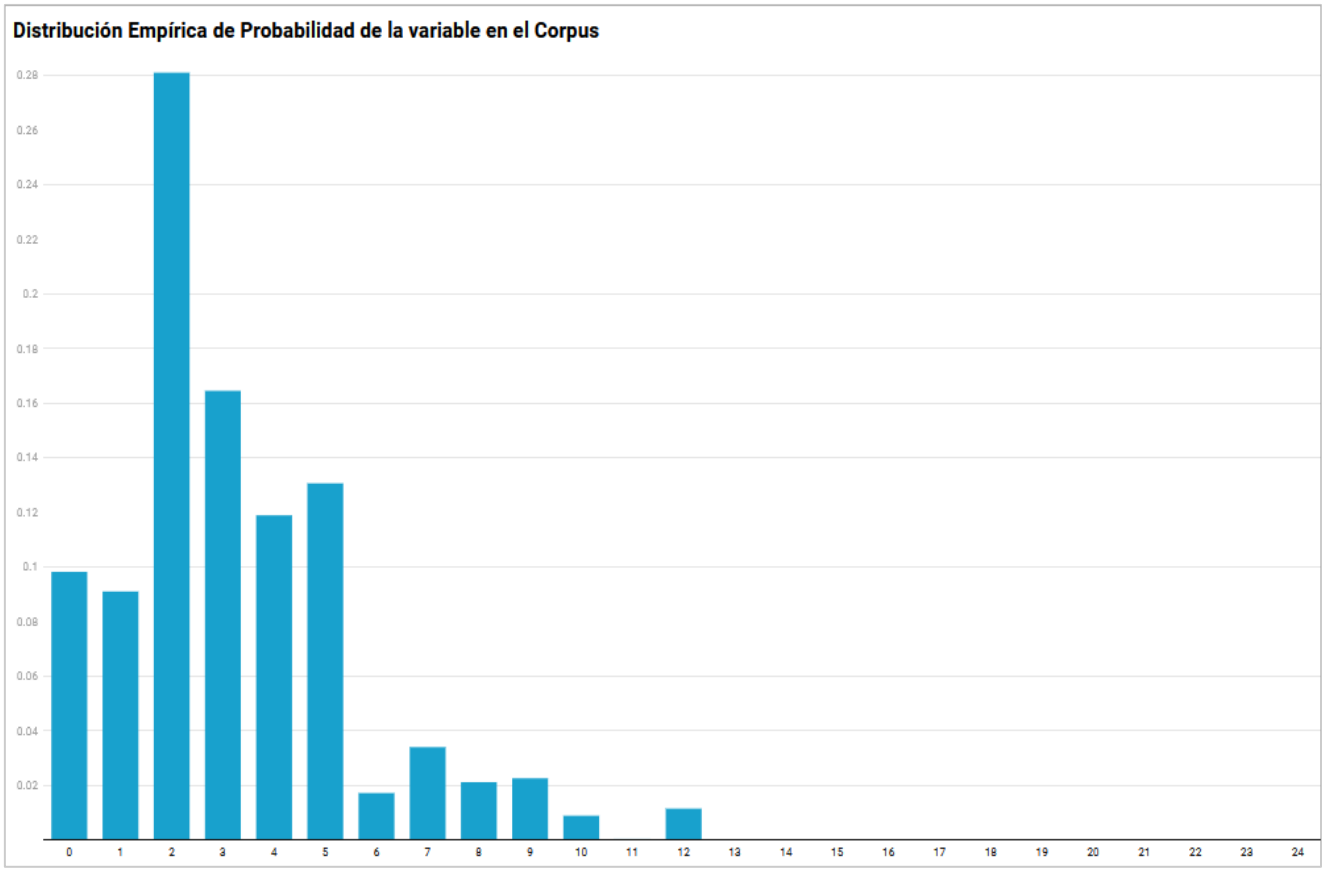
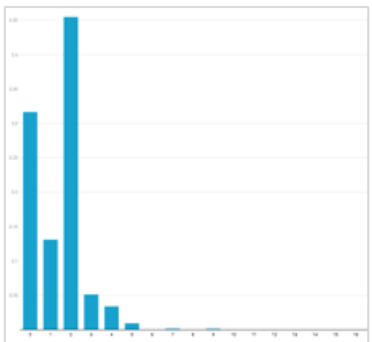


Figura 44 . Distribución de Probabilidad Empírica de la Métrica en el Corpus de Entrenamiento



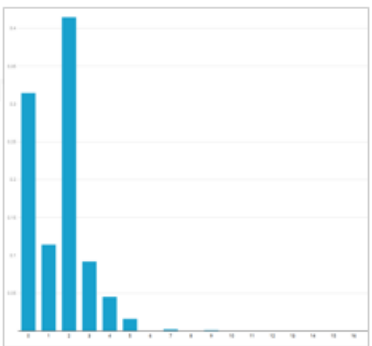


Figura 43 . Distribución de Probabilidad Empírica de la Métrica en los corpus para temperaturas 0,1 ; 0,2 y 0,3

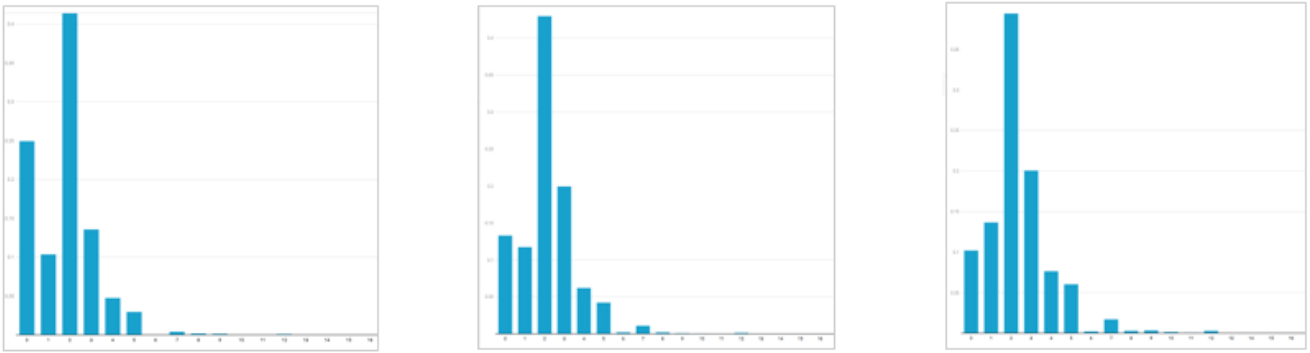


Figura 44 . Distribución de Probabilidad Empírica de la Métrica en los corpus para temperaturas 0,4 ; 0,5 y 0,6

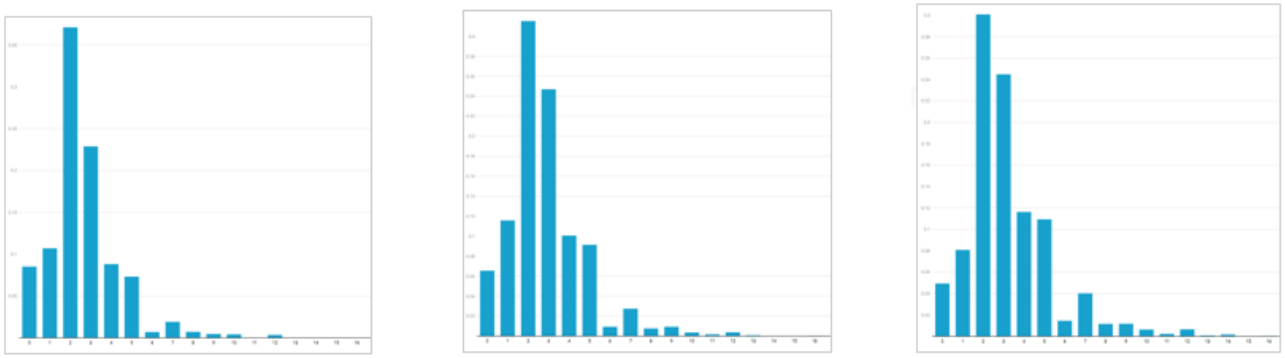


Figura 45 . Distribución de Probabilidad Empírica de la Métrica en los corpus para temperaturas 0,7 ; 0,8 y 0,9

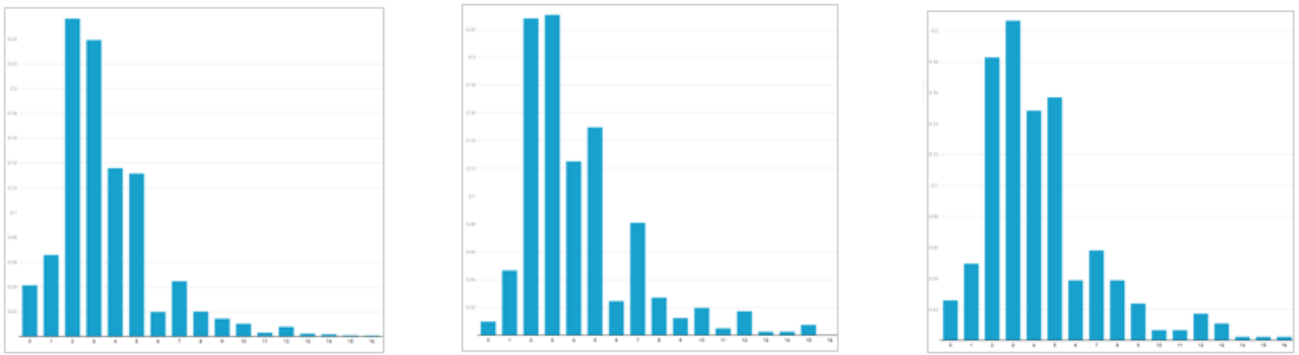


Figura 46 . Distribución de Probabilidad Empírica de la Métrica en los corpus para temperaturas 1,0 ; 1,1 y 1.2

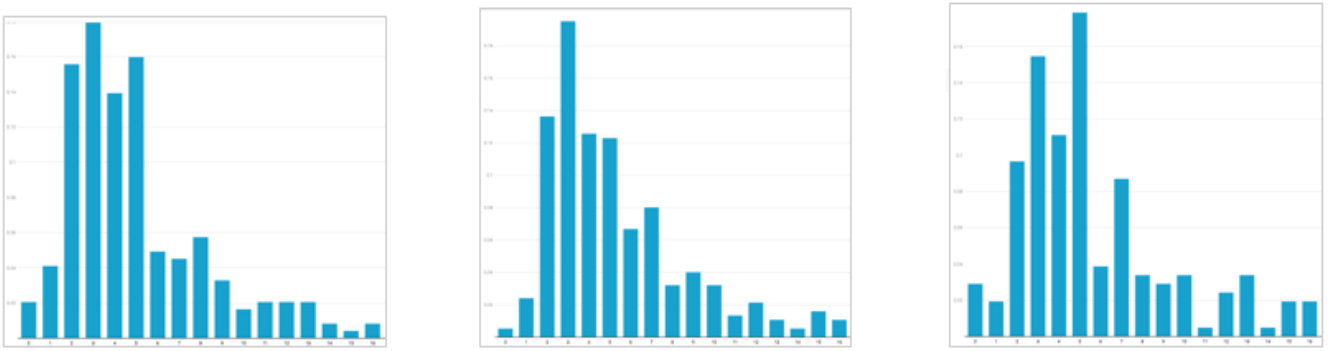


Figura 47 . Distribución de Probabilidad Empírica de la Métrica en los corpus para temperaturas 1,3 ; 1,4 y 1,5

```python
def distr_Salto_Intervalico_Maximo_por_Compas(df, var_analizar, salto_max):    #df = df_2
    #Cantidad devariable  por compás df
    salto_max = np.arange(0,25,1)


    #inicializa el mapping length de intervlos maximos
    mappings_salto_int = {}
    for m in range(len(salto_max)):
        mappings_salto_int[salto_max[m]] = 0


    #define la variable a relevar
    #ratio_difRitmos_notasCompas = []


    #recorre las canciones
    for i in range(len(df['Song_ID'].unique())):  # i=0
        print("Analizando canción: " + df.loc[df['Song_ID']==i, 'Tema'].iloc[0])

        #recorres los compases de esa canción i     i=0  j=9
        for j in range(len(df.loc[df['Song_ID'] == i]['Compas'].unique())) :

            #inicializa intervalo_max
            intervalo_max = 0

            #recorre las notas de ese compas j de la canción i   i=2 j=0  k=0
            cant_notas_en_compas = len(df.loc[(df['Song_ID'] == i) & (df['Compas'] == j+1)])
            if cant_notas_en_compas >= 3: #para que no me tome notas largas o compases de final con mucho silencio

                for k in range(len(df.loc[(df['Song_ID'] == i) & (df['Compas'] == j+1)])):

                    #mefijo que no sea la ultima nota
                    if not k == len(df.loc[(df['Song_ID'] == i) & (df['Compas'] == j+1)])-1:
                    #recorro el compás nota actual y nota siguinte y actualizao el máximo
                        nota_act = df.loc[(df['Song_ID'] == i) & (df['Compas'] == j+1),var_analizar].iloc[k]
                        nota_sig = df.loc[(df['Song_ID'] == i) & (df['Compas'] == j+1),var_analizar].iloc[k+1]

                         #se fija que no sea un rest
                        if not(nota_act == 'r' or nota_sig == 'r'):
                            intervalo = abs(int(nota_sig) - int(nota_act))

                            #evalúa si es mayor al max
                            if intervalo > intervalo_max:
                                intervalo_max = intervalo

                                #evalua si es ayor al máximo del rango de saltos maximos
                                if intervalo_max > max(salto_max):
                                    intervalo max = max(salto max)
```

Figura 48 – Extracto del módulo de Python utilizado para calcular las métricas para cada temperatura

4.8.2.3 Ratio cantidad de figuras diferentes por compas

Unidad de análisis: compás

Variable a analizar: ratio_cant_figuras_diferentes_por_compas

Tamaño de muestra : corpus de entrenamiento 250 canciones, 30512 compases. Corpus generados, aprox 200 melodías generadas, 35000 por temperatura

Descripción del proceso de cálculo de la métrica

Esta variable se calcula como la división entre la cantidad de figuras diferentes y la cantidad la cantidad de notas que componen el compás. En el caso de que todas las notas de compás sean iguales, este ratio dependerá de la cantidad de notas. No es lo mismo dos blancas en un compás, que 8 corcheas (la métrica tomará valores 1/2 y 1/8, respectivamente)

Se determina la función empírica de probabilidad de esta variable a partir del análisis de los corpus generadas para cada temperatura.

Al ser esta una variable aleatoria continua que toma valores entre 0 y 1, hubo que determinar el ancho de los intervalos o *bins*. Se tomaron anchos de 0,05, por lo que nos quedaron 20 bins.

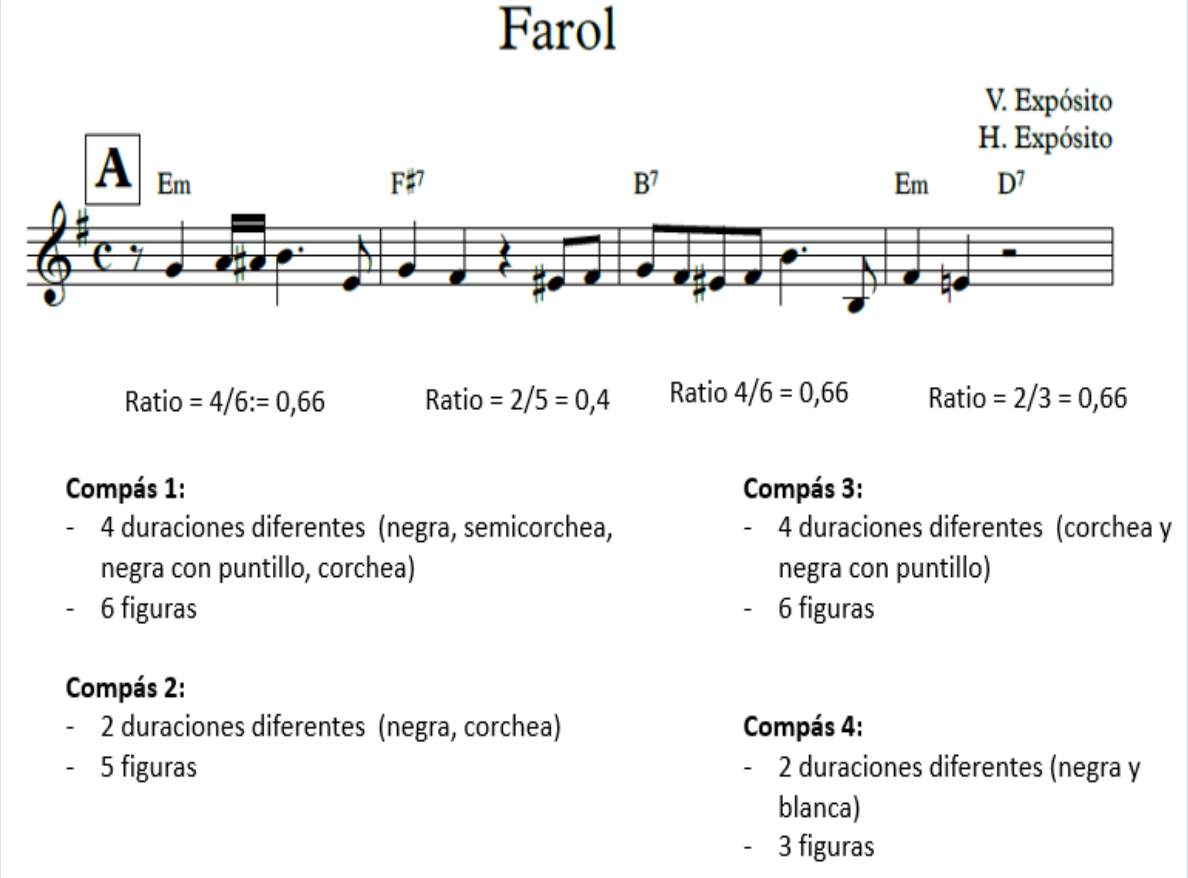


Figura 49: Extracto de partitura de un Tango en el que se explicitan los ratios de cantidad de figuras diferentes por compas

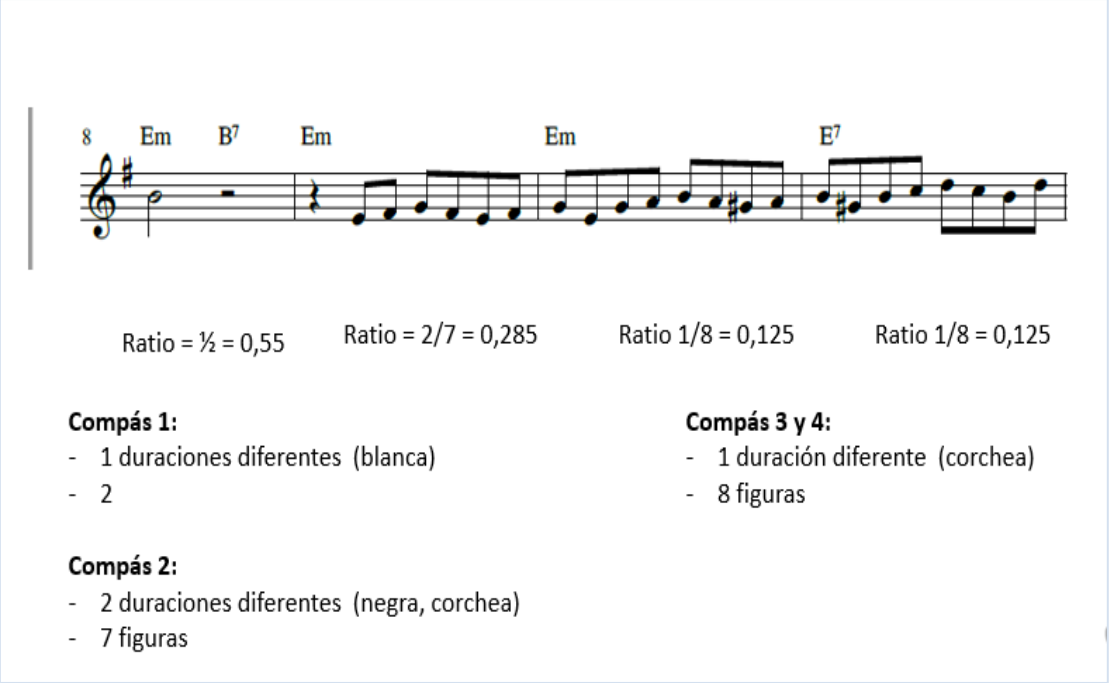


Figura 50: Extracto de partitura de un Tango en el que se explicitan los ratios de cantidad de figuras diferentes por compas. Parte 2

4.8.2.3.1 Distribuciones de Probabilidad empírica de la métrica en el corpus y para frases musicales generadas con distintas temperaturas

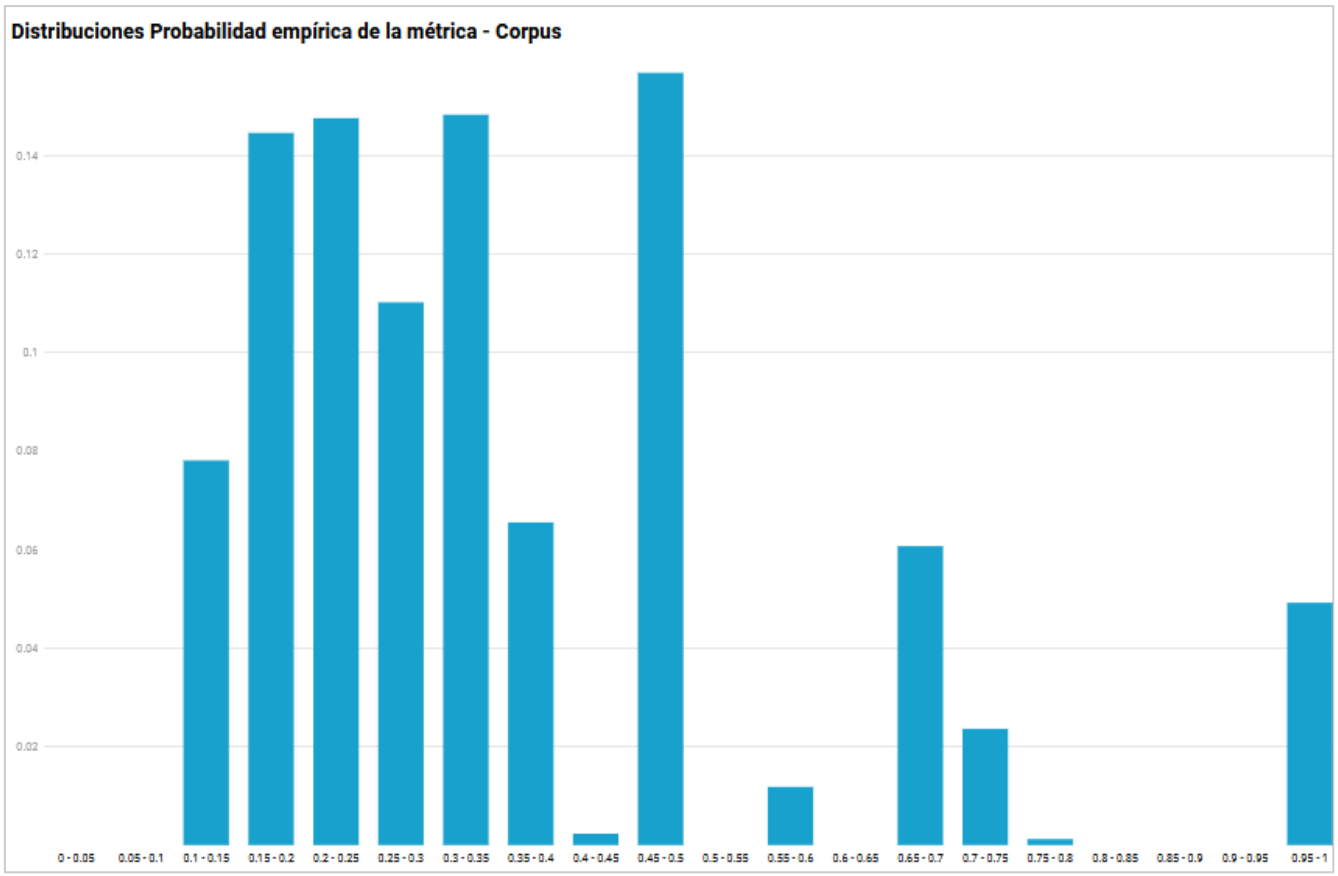


Figura 51 . Distribución de Probabilidad Empírica de la Métrica en el Corpus de Entrenamiento

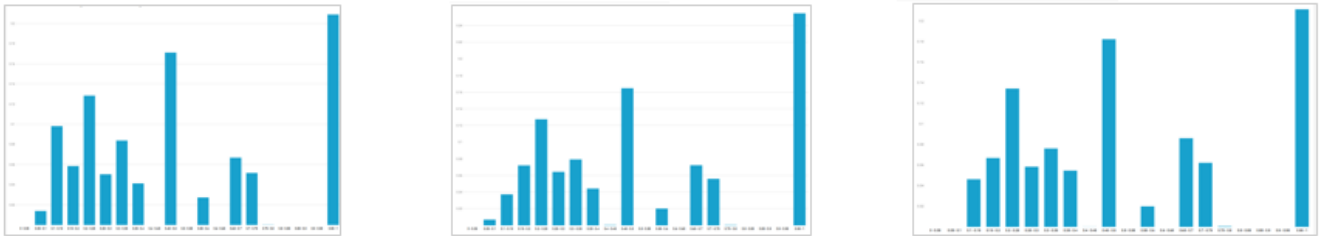


Figura 52 . Distribución de Probabilidad Empírica de la Métrica en los corpus para temperaturas 0,1 ; 0,2 y 0,3

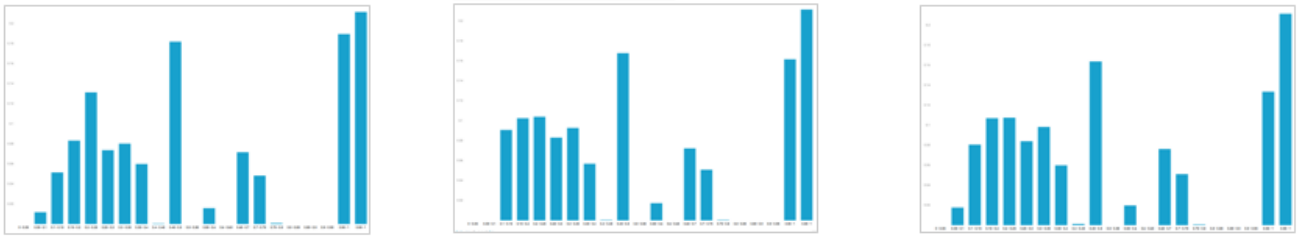


Figura 53 . Distribución de Probabilidad Empírica de la Métrica en los corpus para temperaturas 0,4 ; 0,5 y 0,6

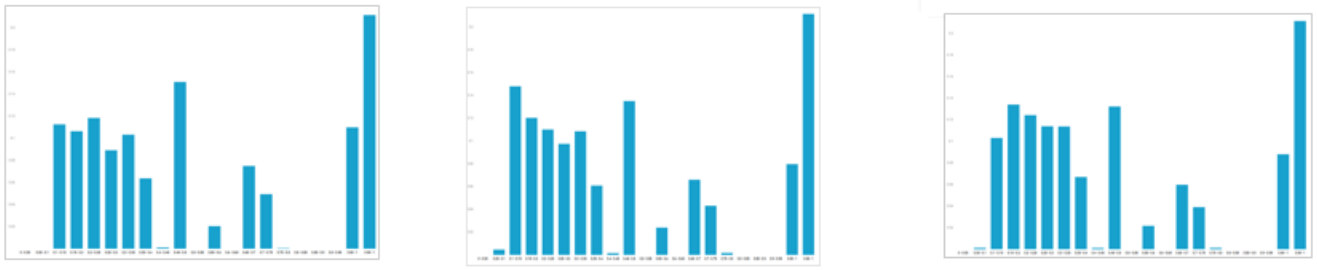


Figura 54 . Distribución de Probabilidad Empírica de la Métrica en los corpus para temperaturas 0,7 ; 0,8 y 0,9

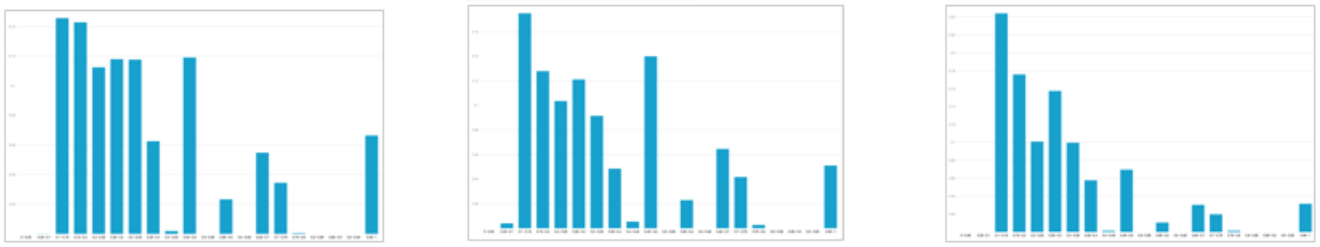


Figura 55 . Distribución de Probabilidad Empírica de la Métrica en los corpus para temperaturas 1,0 ; 1,1 y 1.2

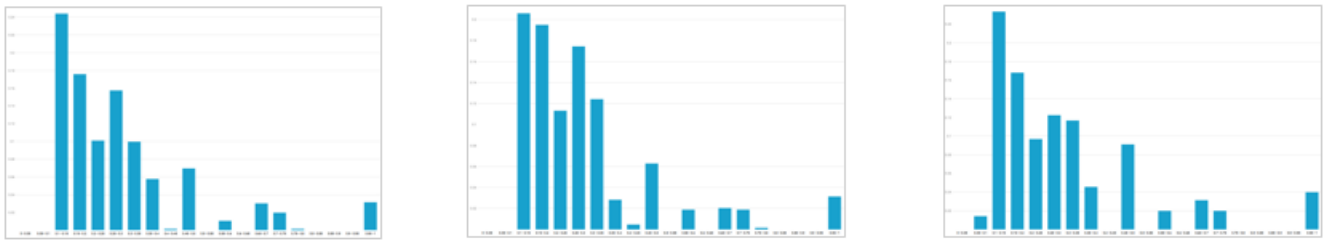
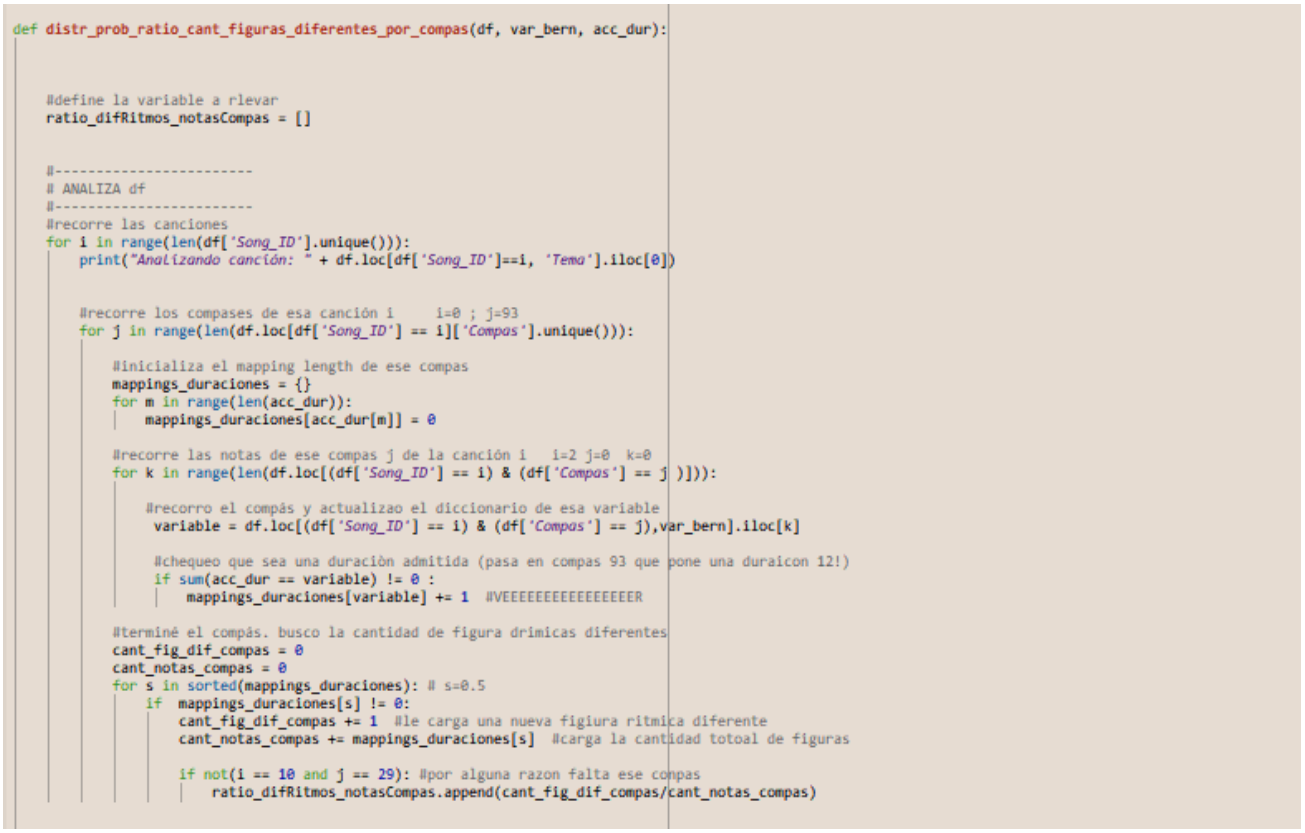


Figura 56 . Distribución de Probabilidad Empírica de la Métrica en los corpus para temperaturas 1,3 ; 1,4 y 1,5



```python
def distr_prob_ratio_cant_figuras_diferentes_por_compas(df, var_bern, acc_dur):



    #define la variable a rlevar
    ratio_difRitmos_notasCompas = []


    #------------------------
    # ANALIZA df
    #------------------------
    #recorre las canciones
    for i in range(len(df['Song_ID'].unique())):
        print("Analizando canción: " + df.loc[df['Song_ID']==i, 'Tema'].iloc[0])


        #recorre los compases de esa canción i     i=0 ; j=93
        for j in range(len(df.loc[df['Song_ID'] == i]['Compas'].unique())):

            #inicializa el mapping length de ese compas
            mappings_duraciones = {}
            for m in range(len(acc_dur)):
                mappings_duraciones[acc_dur[m]] = 0

            #recorre las notas de ese compas j de la canción i   i=2 j=0  k=0
            for k in range(len(df.loc[(df['Song_ID'] == i) & (df['Compas'] == j )])):

                #recorro el compás y actualizao el diccionario de esa variable
                 variable = df.loc[(df['Song_ID'] == i) & (df['Compas'] == j),var_bern].iloc[k]

                 #chequeo que sea una duraciòn admitida (pasa en compas 93 que pone una duraicon 12!)
                 if sum(acc_dur == variable) != 0 :
                    mappings_duraciones[variable] += 1  #VEEEEEEEEEEEEEEER

            #terminé el compás. busco la cantidad de figura drimicas diferentes
            cant_fig_dif_compas = 0
            cant_notas_compas = 0
            for s in sorted(mappings_duraciones): # s=0.5
                if  mappings_duraciones[s] != 0:
                    cant_fig_dif_compas += 1  #le carga una nueva figiura ritmica diferente
                    cant_notas_compas += mappings_duraciones[s]  #carga la cantidad totoal de figuras

                    if not(i == 10 and j == 29): #por alguna razon falta ese compas
                        ratio_difRitmos_notasCompas.append(cant_fig_dif_compas/cant_notas_compas)
```

Figura 57 – Extracto del módulo de Python utilizado para calcular las métricas para cada temperatura

4.8.3 - Evaluación Subjetiva

4.8.3.1 Test de Turing – Diseño

Para la realización de las pruebas asociadas al método de evaluación subjetivo del tipo test de Turing en el que se deberá diferenciar el contenido musical generado (del tipo A en la figura 65) del original creado por un compositor humano (del tipo B en la figura 65) , es que se deberán confeccionar corpus de análisis que contengan una cierta cantidad especifica de contenido musical de cada tipo.

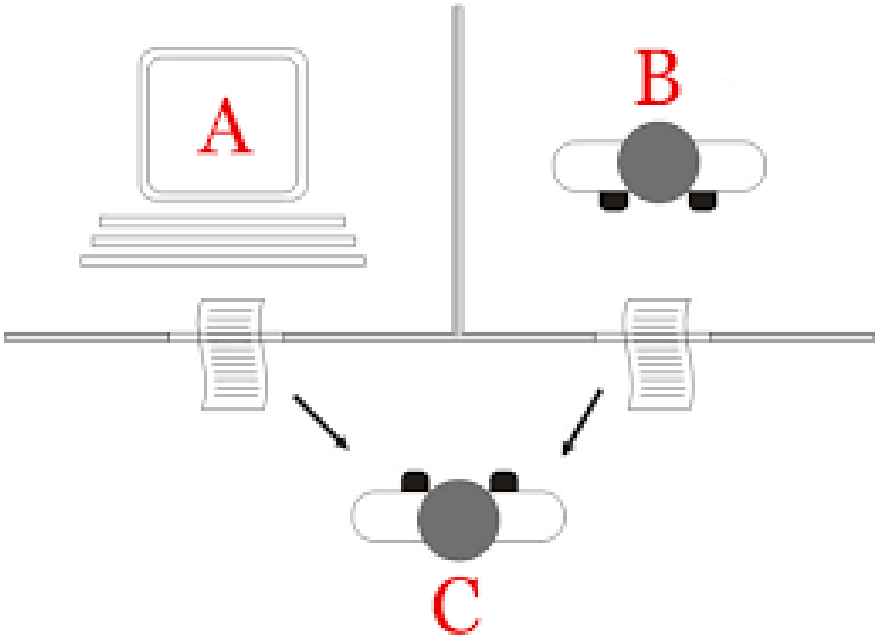


Figura 65 – Concepto Básico del Test de Turing

,

Como este test será estrictamente subjetivo y de escucha de un experto, es necesario que la unidad de análisis no sea más el compás, pues en un alto porcentaje de los casos las ideas musicales con sentido estético no se inician exactamente al principio de un compás. Es por eso que fue necesario desarrollar un algoritmo que reconociera

el inicio de una frase musical y que de allí en adelante generara melodías, o frases como se denominan en la jerga musical, de una determinada longitud.

4.8.3.2 Determinación de una frase musical

El criterio que se utilizó para la extracción de frases musicales a partir de las partes de las canciones del corpus fue el siguiente

- Se asume que una canción o parte de una canción se inicia con una frase musical.

- Se asume que una frase musical termina en una nota de larga duración, tomando como referencia para este estudio la negra con puntillo. Como consecuencia de lo anterior se asume que a continuación de esta referencia se iniciará otra frase musical, llegando así a poder presentar a toda la canción como una sucesión de frases musicales.

Para poder tomar como unidad de análisis a la frase se escribió un módulo en Python que recorre cada canción del corpus siguiendo las indicaciones de inicio de frase explicitadas en las reglas recién planteadas. Se tomará como criterio adicional que las frases tendrán una longitud de 64 subdivisiones a partir de la duración de referencia (negra con puntillo o mayor) Siguiendo este procedimiento se generará una lista de frases.

En la siguiente figura se muestra el procedimiento empleado para la extracción de frases musicales de 4 compases de las canciones del corpus.

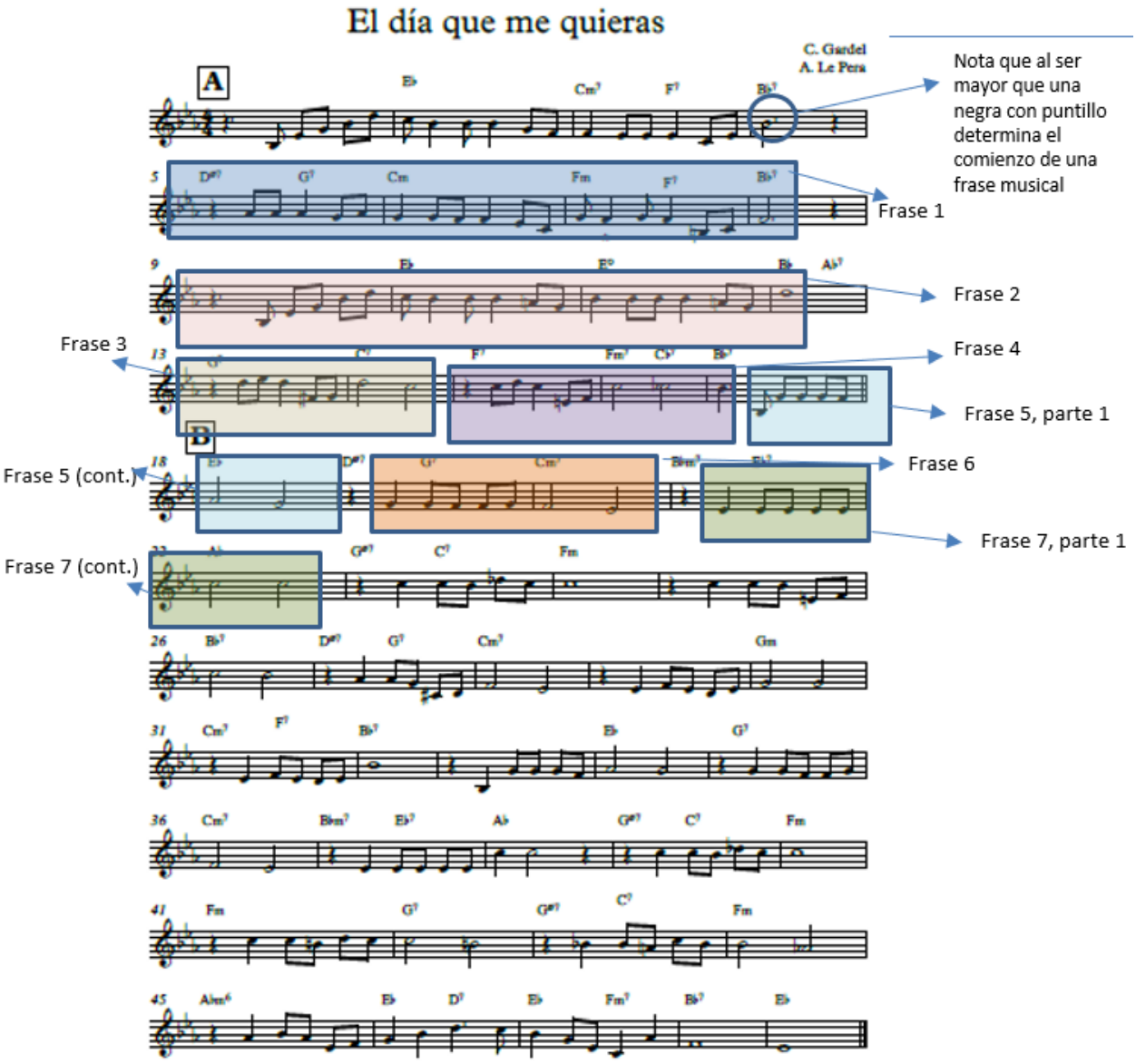
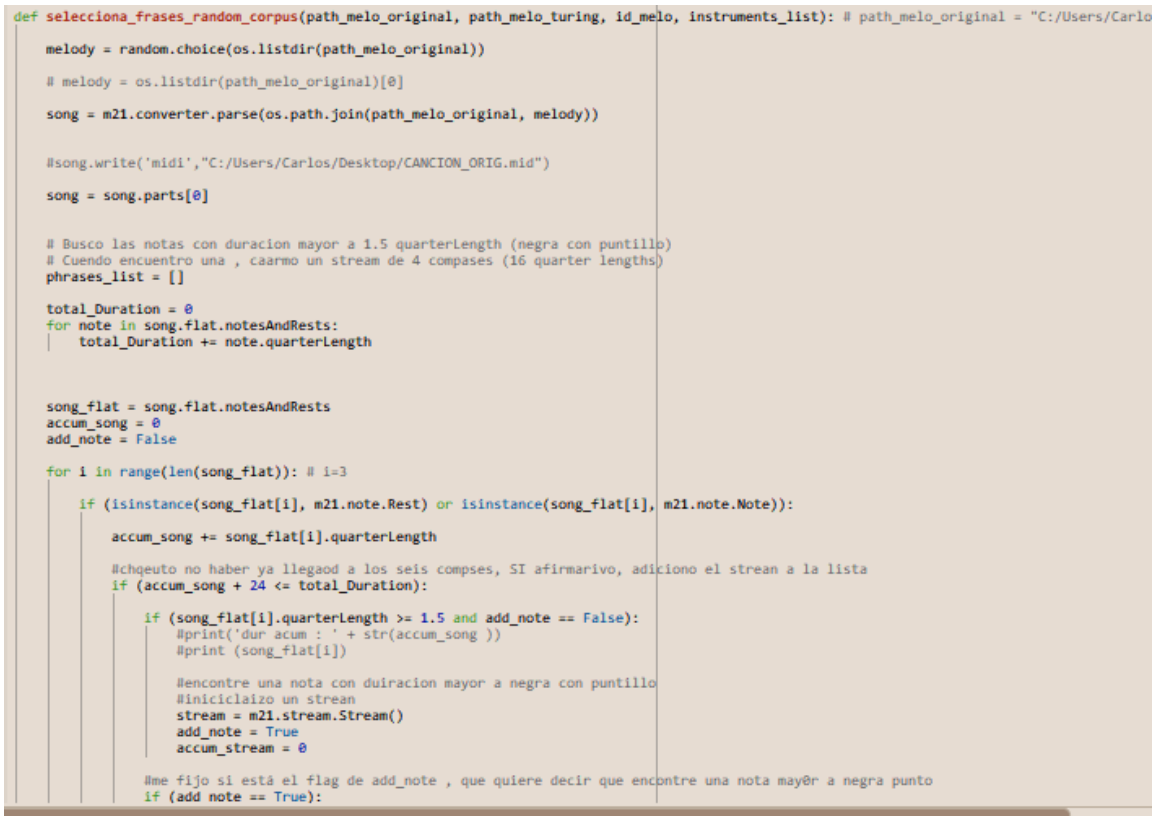


Figura 66 – Ejemplo de extracción de frases en el emblemático Tango "El día que me quieras"



```
def selecciona_frases_random_corpus(path_melo_original, path_melo_turing, id_melo, instruments_list): # path_melo_original = "C:/Users/Carlo

    melody = random.choice(os.listdir(path_melo_original))

    # melody = os.listdir(path_melo_original)[0]

    song = m21.converter.parse(os.path.join(path_melo_original, melody))


    #song.write('midi',"C:/Users/Carlos/Desktop/CANCION_ORIG.mid")

    song = song.parts[0]


    # Busco las notas con duracion mayor a 1.5 quarterLength (negra con puntillo)
    # Cuendo encuentro una , caarmo un stream de 4 compases (16 quarter lengths)
    phrases_list = []

    total_Duration = 0
    for note in song.flat.notesAndRests:
        total_Duration += note.quarterLength



    song_flat = song.flat.notesAndRests
    accum_song = 0
    add_note = False

    for i in range(len(song_flat)): # i=3

        if (isinstance(song_flat[i], m21.note.Rest) or isinstance(song_flat[i], m21.note.Note)):

            accum_song += song_flat[i].quarterLength

            #chqeuto no haber ya llegaod a los seis compses, SI afirmarivo, adiciono el strean a la lista
            if (accum_song + 24 <= total_Duration):

                if (song_flat[i].quarterLength >= 1.5 and add_note == False):
                    #print('dur acum : ' + str(accum_song ))
                    #print (song_flat[i])

                    #encontre una nota con duiracion mayor a negra con puntillo
                    #iniciclaizo un strean
                    stream = m21.stream.Stream()
                    add_note = True
                    accum_stream = 0

                #me fijo si está el flag de add_note , que quiere decir que encontre una nota mayor a negra punto
                if (add_note == True):
```

Figura 67- Extracto del código del módulo en Python para extracción de frases.

4.8.3.3 Diseño Experimental del Test de Turing

Para la realización de las pruebas asociadas al método de evaluación subjetivo del tipo test de Turing, se generan diferentes corpus de análisis , uno por cada temperatura, conteniendo cada uno de ellos 10 frases musicales junto a 10 frases musicales extraídas del corpus de entrenamiento original, totalizando así 20 frases musicales por corpus. Como se tendrá un corpus de análisis por temperatura, en total cada evaluador deberá clasificar como “generadas” o “corpus original” a 300 frases musicales (20 frases -10 originales , 10 diez generadas- por 15 temperaturas)

Se seleccionaron 3 evaluadores, músicos profesionales todos, provenientes de distintas especialidades; Música Clásica, Música Popular Latinoamérica y Jazz.

Cada evaluador analizó 20 melodías para cada una de las 15 temperaturas.

Para asegurarse que las frases posean la necesaria variedad que le dé esa impronta de novedad que el oyente precisa para que la monotonía no termine abrumándolo, y por consiguiente malogrando el test , es que se decidió implementar lo siguiente:

1) Se transpuso cada frase a una tonalidad elegida al azar de manera ascendente o descendente dependiendo de un intervalo de transposición elegido al azar entre 16 semitonos más grave y 16 semitonos más agudo.

2) Se crearon los archivos MIDI que fueron reproducidos por el software de reproducción de contenido multimedial especificado asignando un cierto instrumento al azar para que lo intérprete. Los instrumentos utilizados fueron seleccionados aleatoriamente a partir de esta lista:

- Violín
- Piano
- Acordeón (intentando emular de la manera más aproximada a un bandoneón)
- Viola
- Violoncello.

```
                    if (isinstance(song_flat[i+1], m21.note.Rest) or isinstance(song_flat[i+1], m21.note.Note)):

                        stream.append(song_flat[i + 1])
                        accum_stream += song_flat[i + 1].quarterLength

                        if (accum_stream >= 24):
                            #Traspone a una tonalidad al azar
                            transp_semitones = random.randint(-11, 11) #si es del corpus lo nuevo entre una octava arriba y una abajo
                            transpose_interval = interval.Interval(transp_semitones)
                            stream_transp = stream.transpose(transpose_interval)

                            phrases_list.append(stream)
                            add_note = False



#un stream de cuatro compases para analizar
#    phrase = phrases_list[1]

#elijo un frase al azar

if (len(phrases_list) >= 1) :
    phrase = phrases_list[random.randint(0, len(phrases_list)-1)]

    instrument_random =random.randint(0, 4)
    if (instrument_random == 0) : phrase.insert(0, instrument.Violin())
    elif (instrument_random ==  1):  phrase.insert(0, instrument.Piano())
    elif (instrument_random ==  2):  phrase.insert(0, instrument.Accordion())
    elif (instrument_random ==  3):  phrase.insert(0, instrument.Violoncello())
    else:  phrase.insert(0, instrument.Viola())

    phrase.write('midi', path_melo_turing )
    print('Listo melodia ' + path_melo_turing)
    return True
else:
    return False
```

Figura 68- Extracto del código del módulo en Python para procesamiento de las frases para ser utilizado en los test de Turing.

4.8.3.4 Diseño del Test

El evaluador reproduce cada uno de los archivos MIDI del conjunto específico para una temperatura. A partir de esta escucha el evaluador determina si esta frase proviene del corpus generado o del corpus de entrenamiento.

Se realizarán un total de 20 por escuchas por cada una de las 15 temperaturas por evaluador (900 escuchas anotadas)

Al final del experimento se relevaron la siguientes métricas:

Accuracy: Aciertos/Total_de_Frases

Recall_Corpus = Aciertos/Total de Frases del corpus original

Recall_generados= Aciertos/Total de Frases del corpus generado

Si la accuracy es cercana a 0,5 indica que el evaluador no tiene capacidad de mejorar una selección al azar, por lo que se asume que la calidad de la generación es buena. Cuando más alta sea la diferencia con 0,5 mayor será la capacidad del evaluador de diferenciar las frases reales compuestas por un ser humando de las generadas artificialmente por el modelo, por lo que se tomará como que se asumirá que el modelo de generación no tiene una buena performance al momento de crear contenido musical sintético de calidad.

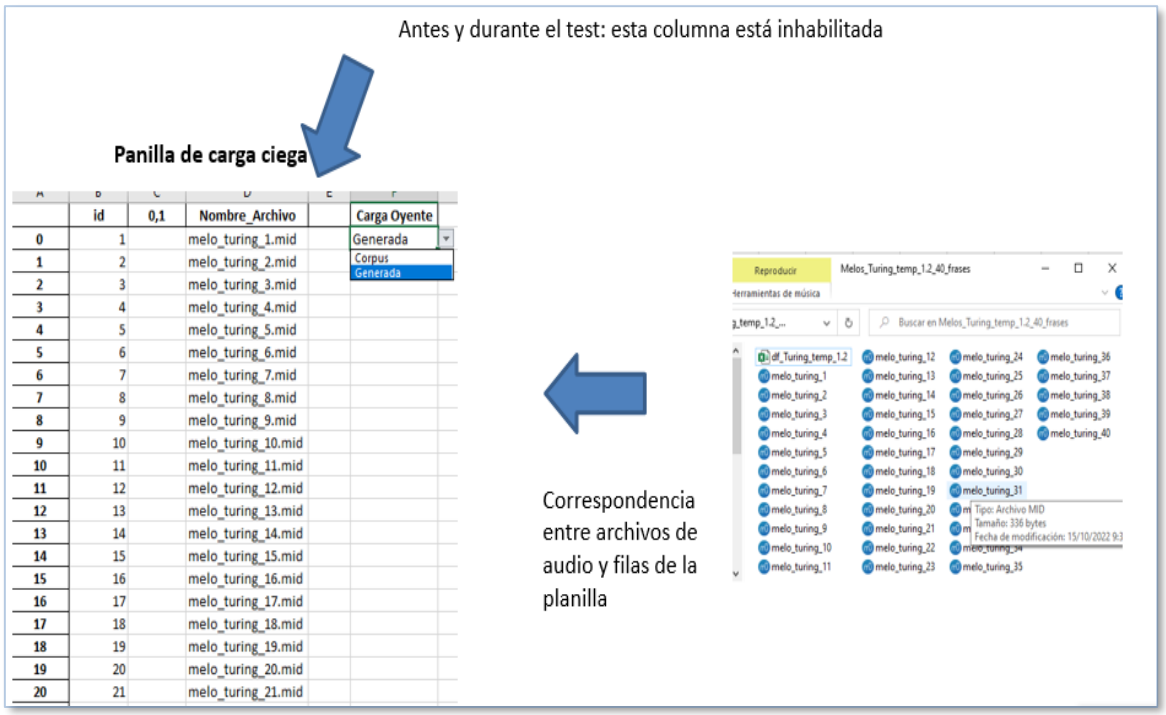


Figura 69- Procedimiento básico para la ejecución del Test

4.8.3.5 Procedimiento

Paso 1: El evaluador reproduce le archivo correspondiente hasta su finalización. Si es necesario lo puede volver a reproducir.

En función de su percepción decide si es original o generado, y pasa a la planilla de Excel

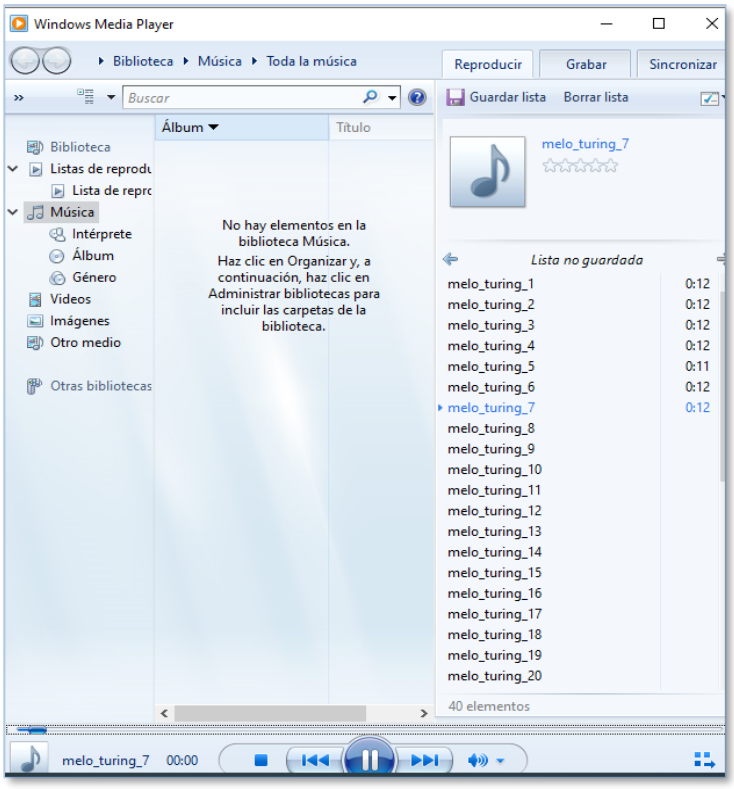
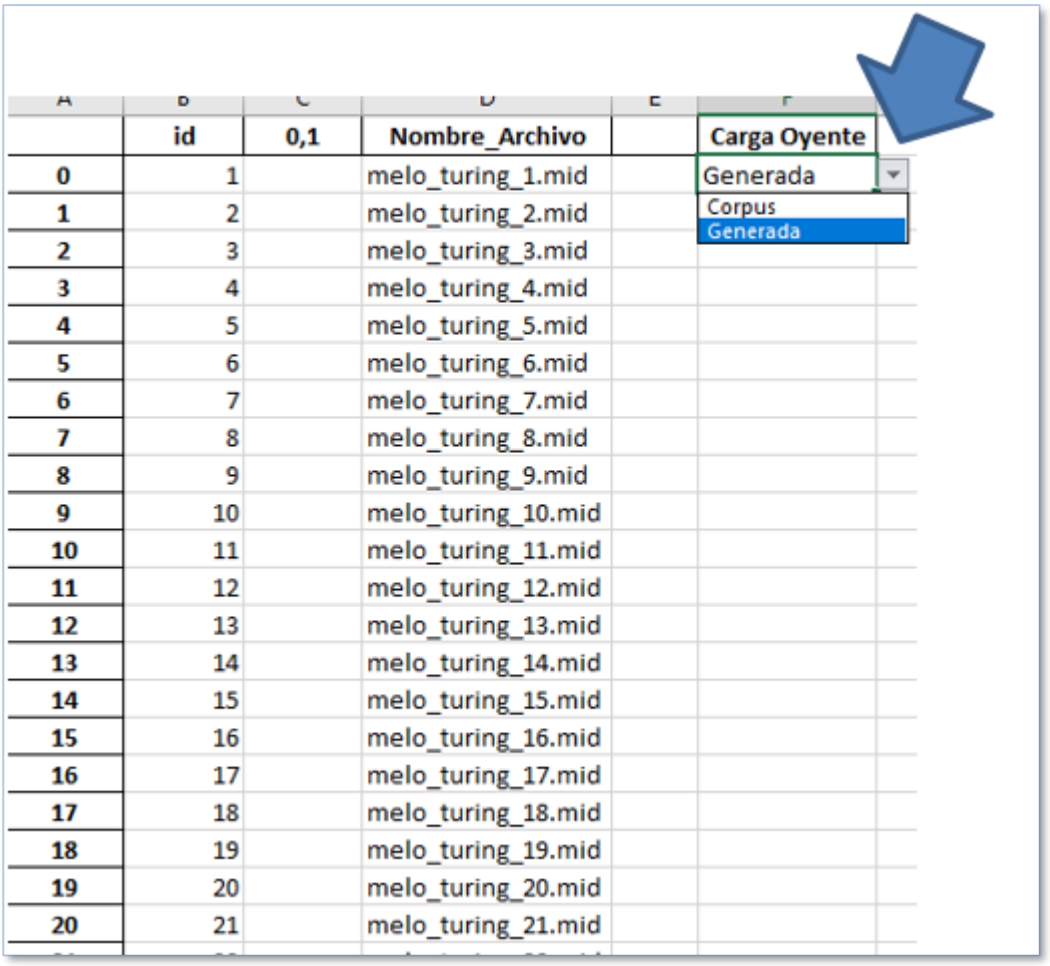


Figura 70- Aplicación de reproductor multimedia que el oyente deberá disponer para la escucha de las frases seleccionadas al zar (del corpus y generadas a diferentes temperaturas)

Paso 2: El oyente carga aquí su respuesta mediante la utilización de una check-list que sólo permite cargar ‘Corpus’ si el oyente considera que esa frase proviene del corpus original de Tangos compuesto por un compositor humano, o ‘Generada’ si en cambio considera que esa frase fue generada por el modelo de Inteligencia Artificial desarrollado

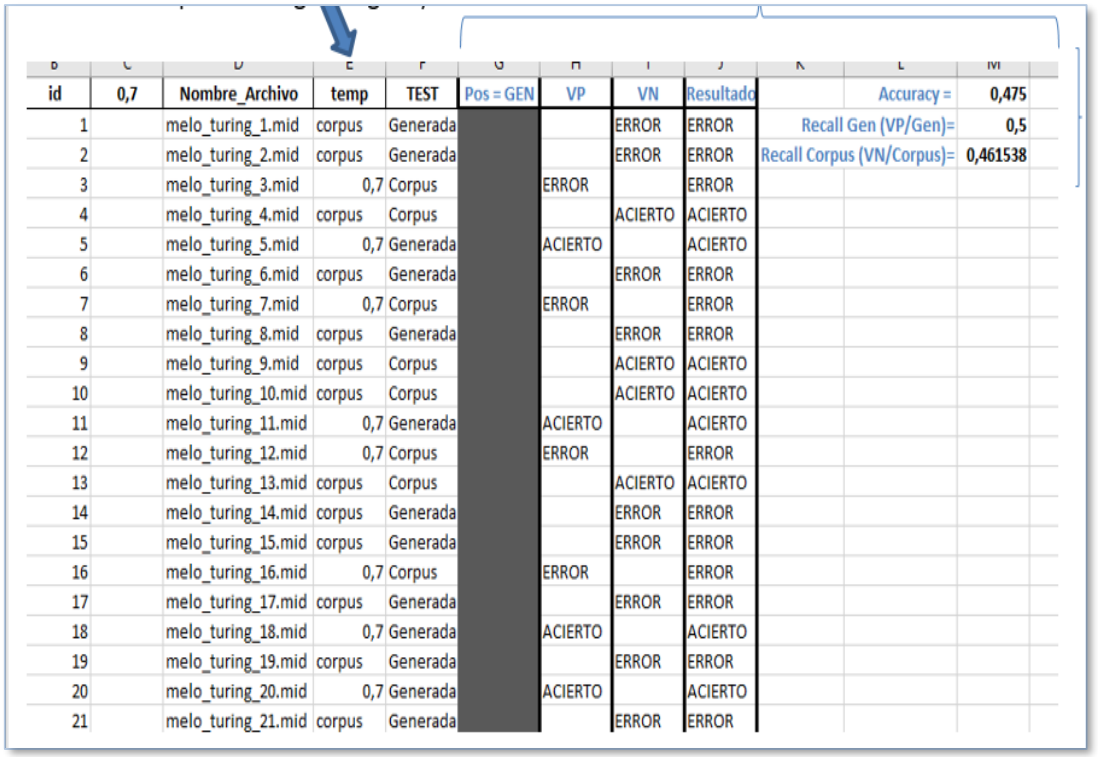


	id	0,1	Nombre_Archivo		Carga Oyente
0	1		melo_turing_1.mid		Generada
1	2		melo_turing_2.mid		
2	3		melo_turing_3.mid		
3	4		melo_turing_4.mid		
4	5		melo_turing_5.mid		
5	6		melo_turing_6.mid		
6	7		melo_turing_7.mid		
7	8		melo_turing_8.mid		
8	9		melo_turing_9.mid		
9	10		melo_turing_10.mid		
10	11		melo_turing_11.mid		
11	12		melo_turing_12.mid		
12	13		melo_turing_13.mid		
13	14		melo_turing_14.mid		
14	15		melo_turing_15.mid		
15	16		melo_turing_16.mid		
16	17		melo_turing_17.mid		
17	18		melo_turing_18.mid		
18	19		melo_turing_19.mid		
19	20		melo_turing_20.mid		
20	21		melo_turing_21.mid		

Figura 71. Planilla de carga en procedimiento ciego: el oyente no sabe si la frase proviene del corpus de entrenamiento o si es generada

Paso 3: Luego del test se habilita esta columna que indica la condición real del archivo (si se corresponde a una frase generada artificialmente por el modelo a una cierta temperatura o si es una frase extraída del corpus de tangos original)

En el extremo superior derecho se encuentran los resultados finales del test para esta temperatura (ver figura 72)



id	0,7	Nombre_Archivo	temp	TEST	Pos = GEN	VP	VN	Resultado		Accuracy =	0,475
1		melo_turing_1.mid	corpus	Generada			ERROR	ERROR		Recall Gen (VP/Gen)=	0,5
2		melo_turing_2.mid	corpus	Generada			ERROR	ERROR		Recall Corpus (VN/Corpus)=	0,461538
3		melo_turing_3.mid	0,7	Corpus		ERROR		ERROR			
4		melo_turing_4.mid	corpus	Corpus			ACIERTO	ACIERTO			
5		melo_turing_5.mid	0,7	Generada		ACIERTO		ACIERTO			
6		melo_turing_6.mid	corpus	Generada			ERROR	ERROR			
7		melo_turing_7.mid	0,7	Corpus		ERROR		ERROR			
8		melo_turing_8.mid	corpus	Generada			ERROR	ERROR			
9		melo_turing_9.mid	corpus	Corpus			ACIERTO	ACIERTO			
10		melo_turing_10.mid	corpus	Corpus			ACIERTO	ACIERTO			
11		melo_turing_11.mid	0,7	Generada		ACIERTO		ACIERTO			
12		melo_turing_12.mid	0,7	Corpus		ERROR		ERROR			
13		melo_turing_13.mid	corpus	Corpus			ACIERTO	ACIERTO			
14		melo_turing_14.mid	corpus	Generada			ERROR	ERROR			
15		melo_turing_15.mid	corpus	Generada			ERROR	ERROR			
16		melo_turing_16.mid	0,7	Corpus		ERROR		ERROR			
17		melo_turing_17.mid	corpus	Generada			ERROR	ERROR			
18		melo_turing_18.mid	0,7	Generada		ACIERTO		ACIERTO			
19		melo_turing_19.mid	corpus	Generada			ERROR	ERROR			
20		melo_turing_20.mid	0,7	Generada		ACIERTO		ACIERTO			
21		melo_turing_21.mid	corpus	Generada			ERROR	ERROR			

Figura 72. Planilla de carga actualizada con los datos reales (deja de ser ciego al indicar el origen de la frase) y con las fórmulas para el cálculo de las métricas de performance

Capítulo 5. Resultados Experimentales

5.1 Divergencias Kullback-Leibler y Jensen-Shannon

5.1.1 Cantidad de alteraciones por compás

p = distribución empírica de probabilidad de la métrica en frases musicales extraídas el corpus de Tangos originales

q = distribución empírica de probabilidad de la métrica en frases musicales extraídas de los corpus generados por el modelo para cada temperatura

Tabla 3. Métrica Cantidad de Alteraciones por Compás. Divergencias Kullback-Leibler y Jensen-Shannon para diferentes temperaturas

Divergencias

Temp	KL_pq	Dist_JS_pq
0.1	0.085	0.292
0.2	0.093	0.305
0.3	0.078	0.279
0.4	0.061	0.247
0.5	0.040	0.200
0.6	0.038	0.195
0.7	0.023	0.151
0.8	0.023	0.150
0.9	0.012	0.109
1.0	0.016	0.126
1.1	0.029	0.171
1.2	0.061	0.247
1.3	0.063	0.252
1.4	0.062	0.250
1.5	0.057	0.238

Created with Datawrapper

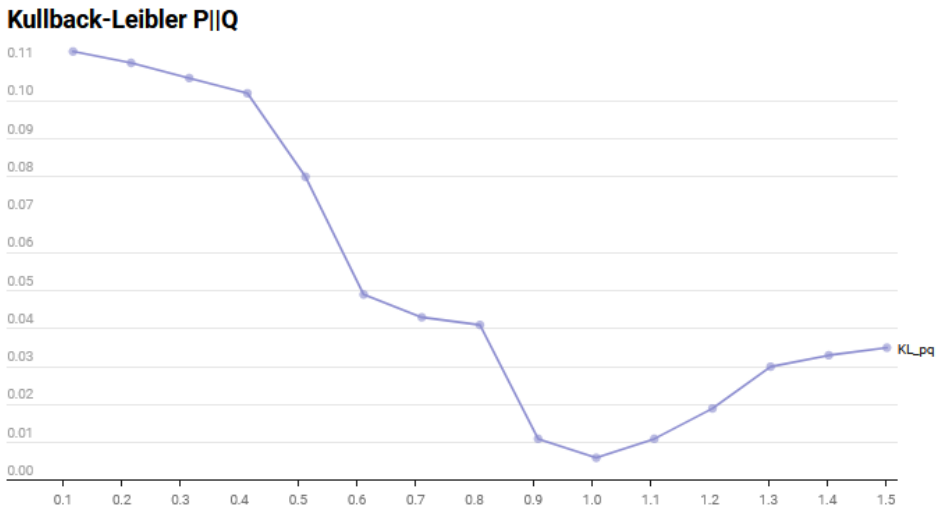


Figura 58 - Métrica Cantidad de Alteraciones por Compás. Divergencias Kullback Leibler

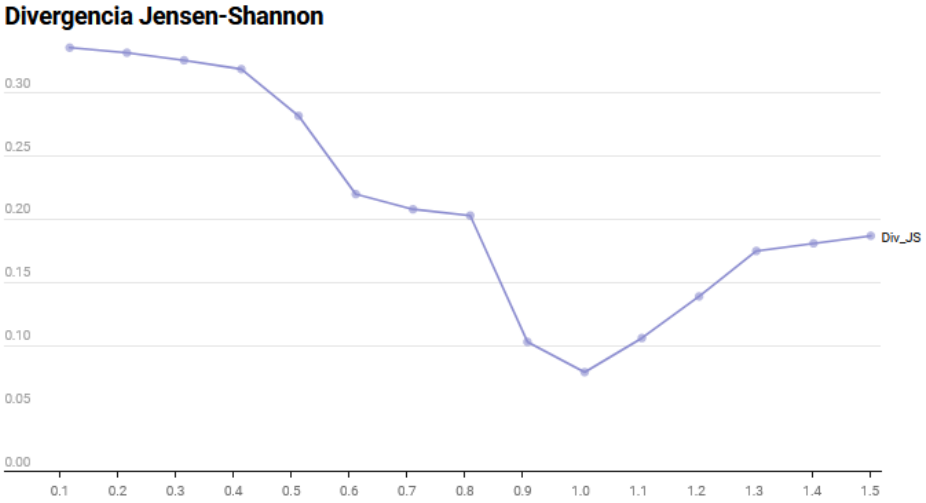


Figura 59 - Métrica Cantidad de Alteraciones por Compás. Distancias Jensen-Shannon

La temperatura en la que se registra el menor valor de ambas divergencias, es al que llamaremos Temperatura Óptima, cuyo valor es:

Temp_opt = argmin (JS(Cant_alteraciones por compás)) = **1**
temp

5.1.2 Salto_Intervalico_Maximo_por_Compas

p = distribución empírica de probabilidad de la métrica en frases musicales extraídas el corpus de Tangos originales

q = distribución empírica de probabilidad de la métrica en frases musicales extraídas de los corpus generados por el modelo para cada temperatura

Tabla 3. Métrica Cantidad de Alteraciones por Compás. Divergencias Kullback-Leibler y Jensen-Shannon para diferentes temperaturas

Divergencias

Temp	KL_pq	Dist_JS_pq
0.1	0.197	0.443
0.2	0.158	0.398
0.3	0.125	0.354
0.4	0.113	0.336
0.5	0.072	0.268
0.6	0.026	0.236
0.7	0.030	0.173
0.8	0.034	0.184
0.9	0.045	0.212
1.0	0.058	0.242
1.1	0.1...	0.345
1.2	0.110	0.332
1.3	0.110	0.332
1.4	0.216	0.465
1.5	0.223	0.472

Get the data · Created with Datawrapper

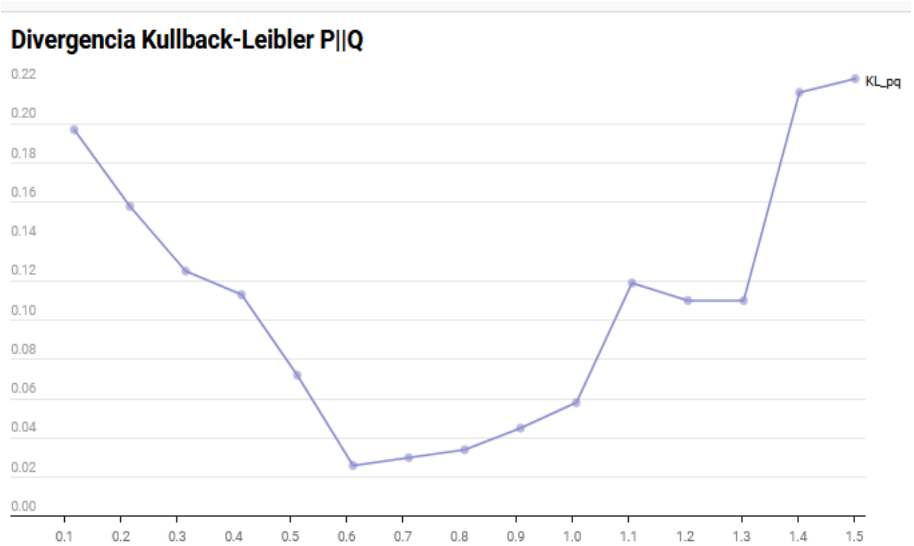


Figura 60 - Métrica Salto Interválico Máximo por Compás. Divergencia Kullback Leibler

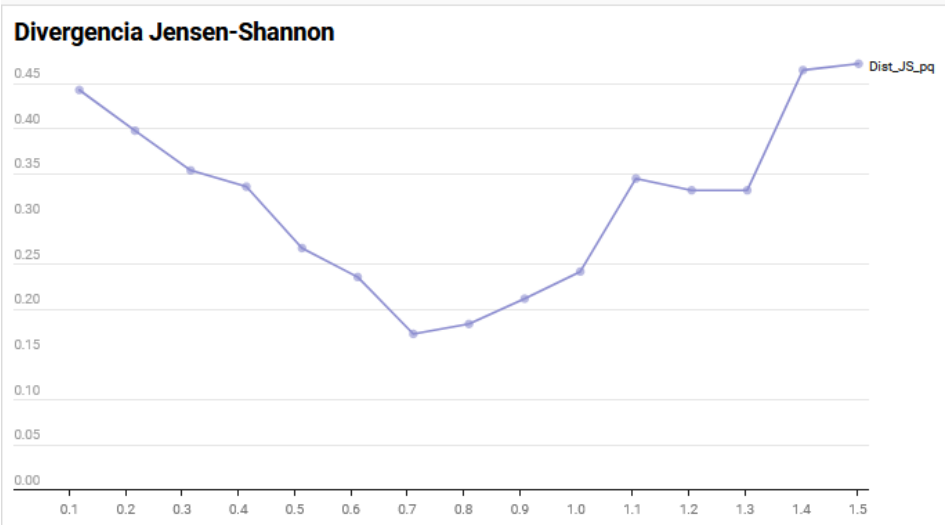


Figura 61 - Métrica Salto Interválico Máximo por Compás. Distancia Jensen-Shannon

La temperatura óptima para esta métrica es:

Temp_opt = argmin (JS(Salto_Intervalico_Maximo_por_Compas) = **0.6**
temp

5.1.3 Ratio_cant_figuras_diferentes_por_compas

p = distribución empírica de probabilidad de la métrica en frases musicales extraídas del corpus de Tangos originales

q = distribución empírica de probabilidad de la métrica en frases musicales extraídas de los corpus generados por el modelo para cada temperatura

Tabla 5- Métrica Ratios de Cantidad de Figuras Diferentes por Compas. Divergencias Kullback Leibler y Jensen-Shannon para Diferentes temperaturas

Divergencias

Temp	KL_pq	Dist_JS_pq
0.1	0.085	0.292
0.2	0.093	0.305
0.3	0.078	0.279
0.4	0.061	0.247
0.5	0.040	0.200
0.6	0.038	0.195
0.7	0.023	0.151
0.8	0.023	0.150
0.9	0.012	0.109
1.0	0.016	0.126
1.1	0.029	0.171
1.2	0.061	0.247
1.3	0.063	0.252
1.4	0.062	0.250
1.5	0.057	0.238

Get the data · Created with Datawrapper

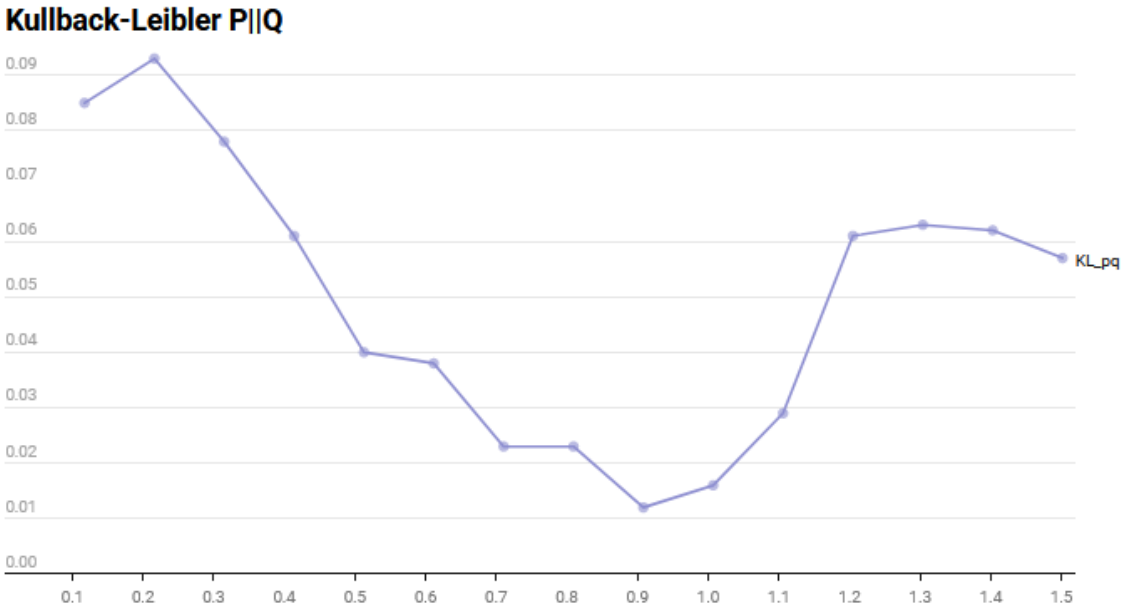


Figura 62 - Métrica Ratio de Cantidad de Figuras Diferentes por Compas.. Divergencia Kullback Leibler

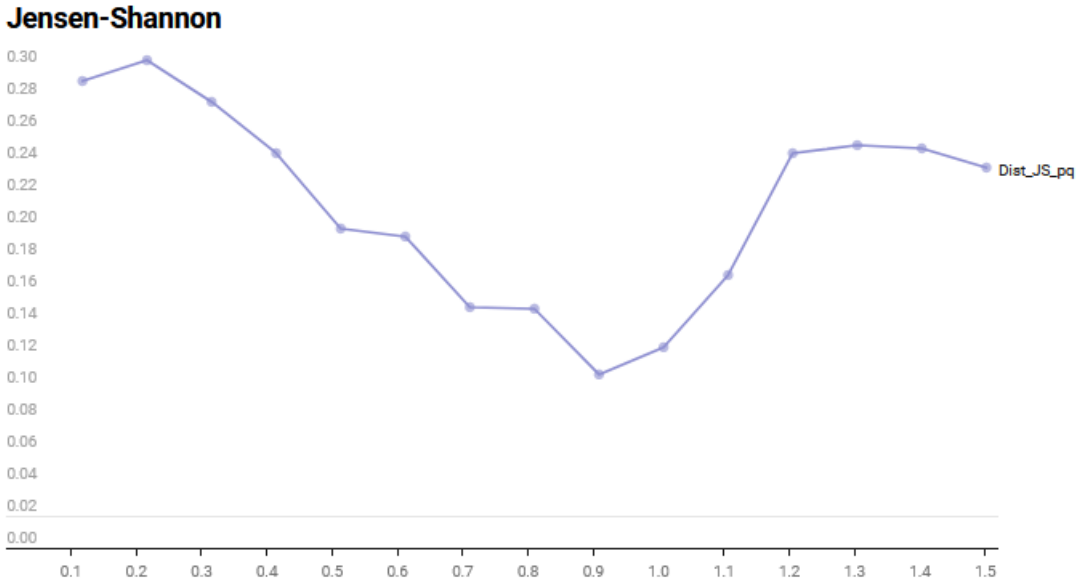


Figura 63 - Métrica Ratio de Cantidad de Figuras Diferentes por Compas. Distancia Jensen-Shannon

La temperatura óptima para esta métrica es:

Temp_opt = argmin (JS(Ratio_cant_figuras_diferentes_por_compas))) = 0.9
temp

```python
def kl_divergence(p, q):

    KL = 0
    for i in range(len(p)):
        if not (p[i] == 0 or q[i] == 0):
            KL +=  p[i] * math.log(p[i]/q[i],2.0)

    return KL

# calculate the js divergence
def js_divergence(p, q):
    m = 0.5 * (p + q)
    return 0.5 * kl_divergence(p, m) + 0.5 * kl_divergence(q, m)


def calcula_JS(p, q):


    # define distributions
    p = np.asarray(p)
    q = np.asarray(q)

    # calculate JS(P || Q)
    JS_pq = js_divergence(p, q)
    distance_JS_pq = math.sqrt(JS_pq)
```

Figura 64 – Extracto del módulo de Python utilizado para calcular las divergencias de Kullback Leibler y Jensen-Shannon para Diferentes temperaturas

5.2 – Determinación de la Temperatura Óptima Conjunta

Para la determinación de Temperatura Óptima Conjunta se procede a promediar los valores de temperaturas óptimas a partir de la análisis de las divergencias para cada métrica. Su valor es:

Temp_opt = promedio (Temp_opt$_{SAC}$;Temp_opt $_{SIMC}$,Temp_opt$_{RCFDC}$) = Promedio(1 ; 0,6 ; 09) = **0,833**

Donde:

CAC = Cant_alteraciones por compás

SIMC = Salto_Intervalico_Maximo_por_Compas

RCFDC = Ratio_cant_figuras_diferentes_por_compas

5.3 Análisis de los Resultados Experimentales en Métricas Subjetivas

5.3.1 *Accuracy* y *Recall* para diferentes temperaturas

Tabla 6 . Tabla de accuracy para diferentes temperaturas

Temperature	Accuracy
0.10	0.55
0.20	0.51
0.30	0.44
0.40	0.50
0.50	0.51
0.60	0.44
0.70	0.45
0.80	0.52
0.90	0.54
1.00	0.63
1.10	0.64
1.20	0.71
1.30	0.66
1.40	0.73
1.50	0.78

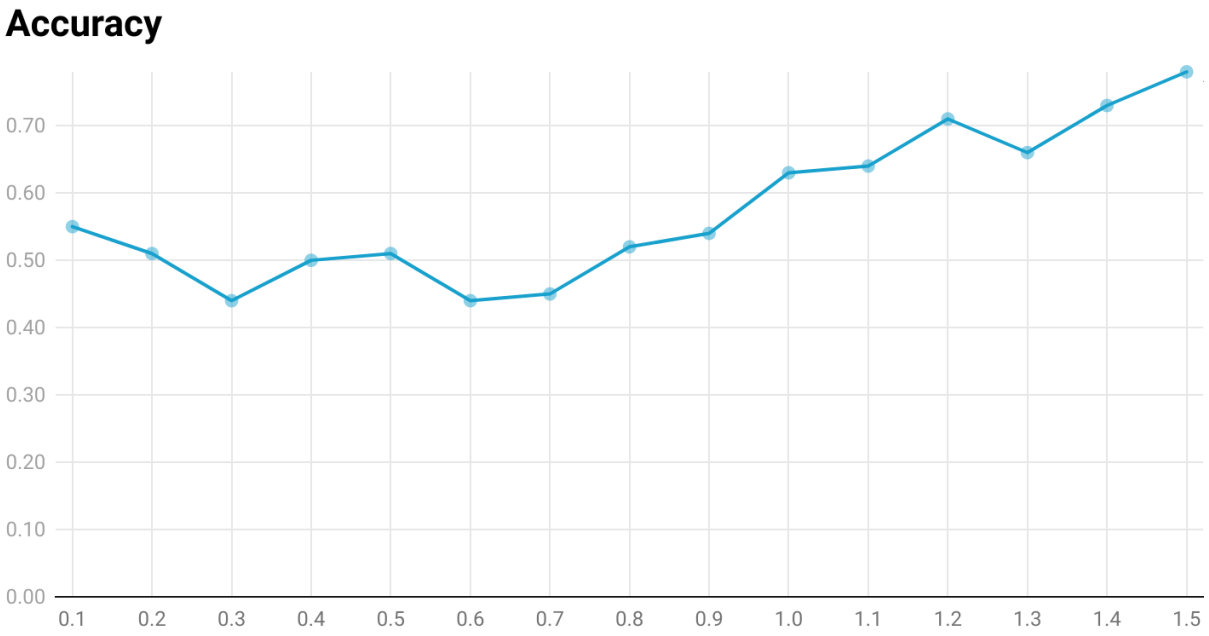


Figura 73. Accuracy para diferentes temperaturas

Tabla 7 . Tabla de recall de positivos (generados) para diferentes temperaturas

Temperature	Recall Gen (VP/Gen)
0.10	0.56
0.20	0.57
0.30	0.50
0.40	0.48
0.50	0.47
0.60	0.32
0.70	0.50
0.80	0.43
0.90	0.31
1.00	0.65
1.10	0.64
1.20	0.64
1.30	0.45
1.40	0.57
1.50	0.82

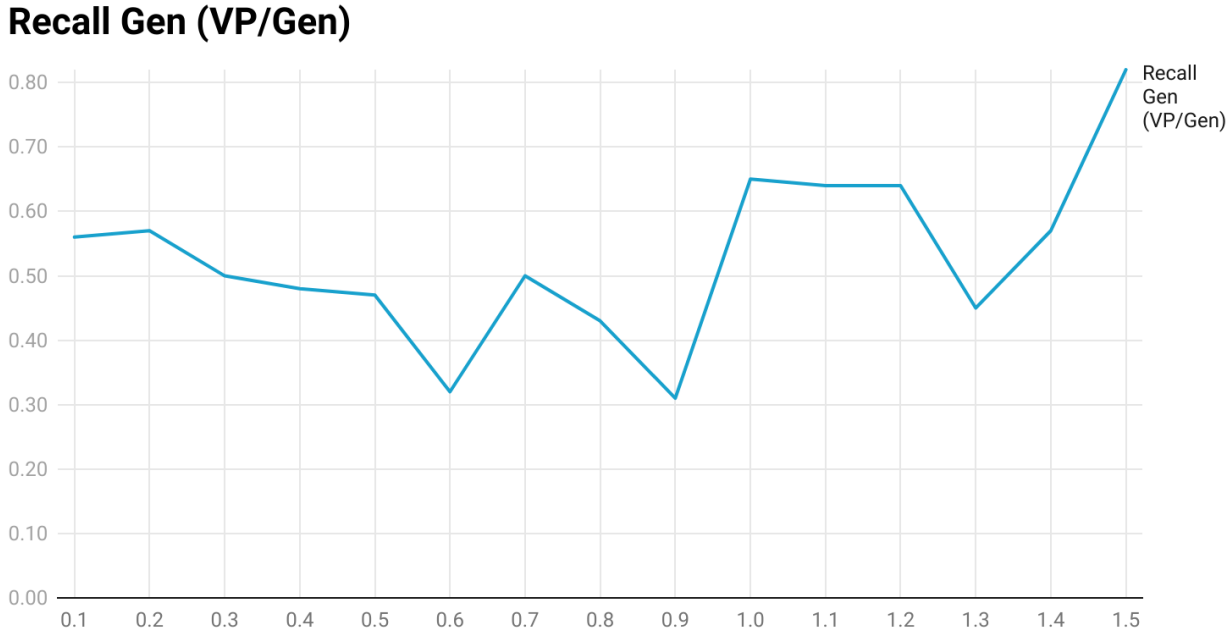


Figura 74 Recall de positivos (generados) para diferentes temperaturas

Tabla 8 . Tabla de recall de negativos (corpus) para diferentes temperaturas

Temperature	Recall Corpus (VN/Corpus)
0.10	0.64
0.20	0.53
0.30	0.28
0.40	0.42
0.50	0.52
0.60	0.40
0.70	0.46
0.80	0.50
0.90	0.59
1.00	0.50
1.10	0.67
1.20	0.73
1.30	0.70
1.40	0.74
1.50	0.83

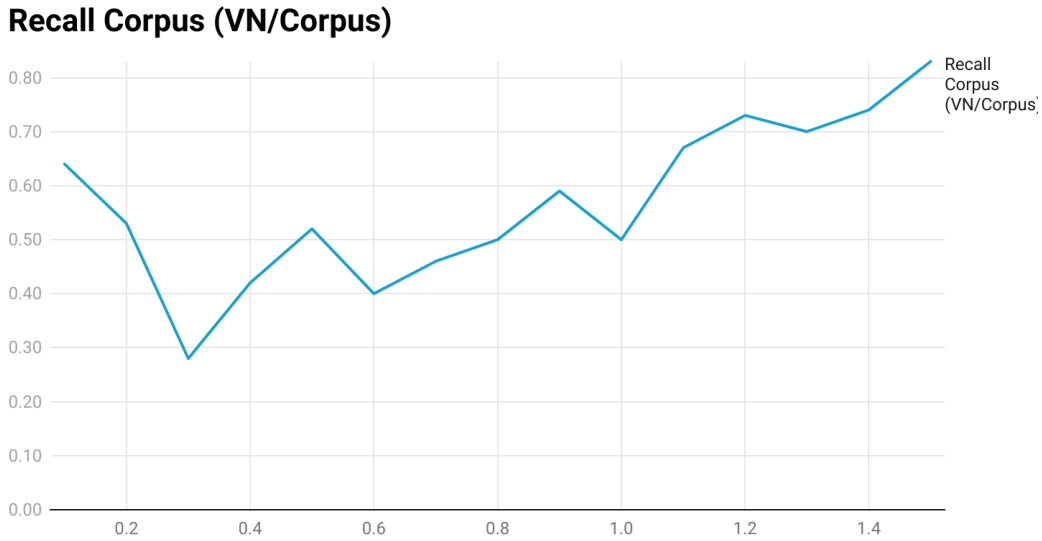


Figura 75 Recall de negativos (corpus) para diferentes temperaturas

5.3.2 Determinación de la Temperatura óptima

Para uniformizar en un solo valor de temperatura a utilizar para la generación , primero se definió una zona de temperaturas optimas, que es la que contiene valores de accuracy en el rango 0.5 +/- 0.05. Este rango (como se aprecia en la figura 76) está compuesto por las temperaturas 0.7; 0.8 y 0.9. Al ser estos estos valores asociados a proporciones se les aplica una función de normalización del tipo softmax para así obtener una función empírica de probabilidad de esta variable , y luego calcular su esperanza matemática.

Temperature	Accuracy
0.10	0.55
0.20	0.51
0.30	0.44
0.40	0.50
0.50	0.51
0.60	0.44
0.70	0.45
0.80	0.52
0.90	0.54
1.00	0.63
1.10	0.64
1.20	0.71
1.30	0.66
1.40	0.73
1.50	0.78

Figura 76 – Accuracy del Test de Turing para distintas temperaturas

A partir de esta operación del tipo softmax se obtiene el valor de temperatura óptima de evaluación subjetiva mediante test de Turing de **0,802**

5.3.3 Resumen

Luego de los experimentos realizados basados en los diferentes métodos de evaluación, tanto objetivos como subjetivos, y el análisis de sus fortalezas y sus debilidades, hemos arribado a un resultado que viabiliza la implementación de un modelo de generación de contenido musical educativo de ejercitación en audioperceptiva y lectura musical a una temperatura específica, que cumpla con los requisitos enunciados de variabilidad, musicalidad y extensión virtualmente ilimitada.

El trabajo de entrenamiento de la red neuronal generativa se llevó a cabo utilizando un corpus de tamaño razonable y la red neuronal recurrente utilizada fue entrenada en servidores con un hardware de accesibilidad en algunos casos abierta (Google Colab) o a costos muy razonables para un estudio de investigación de esta envergadura (Amazon SageMaker). Haber arribado a estos resultados utilizando estos recursos abre caminos de investigación e implementación sumamente auspiciosos, y sobre los cuales se ahondará en la sección siguiente

Capítulo 6 - Conclusiones

Al analizar en detalle el estado del arte en materia de oferta de material de entrenamiento musical de ejercitación variada y de alta musicalidad nos hemos encontrado con que lo desarrollado a la fecha posee limitaciones, tanto en lo físico (material tradicional en papel, CD o VHS/DVD) como en lo funcional (el sistema de reglas de los softwares de enseñanza).

A partir de allí se ha abordado un trabajo de investigación tendiente a evaluar la viabilidad del desarrollo e implementación de un modelo basado en Redes Neuronales Profundas, arquitectura paradigmáticas de la Inteligencia Artificial , en particular las redes generativas, que subsane estas falencias. Este tipo de modelos per-se tiene la capacidad de generar cantidades virtualmente ilimitadas de contenido, por lo cual demostraríamos que una de los requerimientos había sido cumplimentado, restando analizar la musicalidad de estas creaciones sintéticas. Para ello abordamos en exhaustivo estudio del estado del arte tanto de la generación de contenido musical , relevando los diferentes desarrollos, analizando sus alcances, y en especial los diferentes métodos de evaluación.

A partir de allí se pudo tomar conocimiento de las limitaciones de estos desarrollos. Este minucioso e informativo estudio nos permitió comprobar que la aplicación de los modelos de Inteligencia Artificial para la generación de contenido educativo se encontraba entre los tópicos clasificados como Trabajo a futuro de los artículos que condensan la actividad académica y de investigación de una cierta disciplina (los denominados *surveys*). Este conocimiento hizo que fuera necesario analizar la viabilidad de implementar un modelo utilizando modelos de Deep Learning capaces de ser entrenados sin tener que recurrir a una inversión prohibitiva en recursos de hardware y software.

Teniendo el aval y el incentivo que le imprimió a este trabajo de investigación la posibilidad de tener un desarrollo necesario y a la vez novedoso en ciernes, es que se amplió el espectro de la investigación, incluyendo ahora la prueba de musicalidad. En la literatura de investigación especializada en evaluación subjetiva de modelos generativos se deja expresa constancia de la complejidad asociada a su desarrollo e implementación. Esto se debe a que al basarse estos métodos en apreciaciones subjetivas -y ya así lo indica su nombre-, su replicación es de difícil concreción, complicando así la capacidad de extrapolar los resultados a partir de ellos obtenidos. No obstante lo anterior se encaró un trabajo de evaluación subjetiva del tipo Test de Turing acorde a las especificaciones del diseño experimental determinado. Este estudio de evaluación ratificó el rango de valores óptimo del hiperparámetro de temperatura hallado en el análisis de las métricas objetivas, ratificando el éxito de los resultados obtenidos en la evaluación objetiva.

Las características esenciales de la música como lenguaje simbólico que permite compartir significado, y su particular sofisticación hacen que para el cometido fundamental de esta tesis, que es la generación de contenido de ejercitación musical, sea menester considerar la posibilidad de crear contenido de mayor longitud . Se ha relevado en la literatura especifica que esta es una de las limitaciones fundamentales de los modelos desarrollados hasta la fecha. Es por ello que una línea de trabajo a futuro será desarrollar un modelo que genere contenido musical de ejercitación de mayor longitud, para lo cual será necesario seguir el estado del arte en la generación de contenido musical.

El trabajo se enfocó en la generación de melodías con fines específicos de creación sintética de ejercicios en las subdisciplinas musicales de lectura y audioperceptiva. Los modelos de generativos de Deep Learning son aptos para la generación de contrapuntos, acordes, orquestaciones, etc. La limitante que esta poseen está íntimamente asociadas a las restricciones que hemos analizado, por lo que su

aplicación en educación estará supeditada a los avances en el desarrollo de modelos generativos que vayan subsanando estos escollos asociados a la creación de contenido musical en general.

Para evaluar la implementación efectiva de este modelo en el aula ser debería analizar el diseño pedagógico y los lineamientos establecidos en los programas de la asignatura o del curso en particular en el que se aplicarían estos modelos generativos. Los modelos basados en redes neuronales son susceptibles de ser condicionados de diferentes maneras, por lo que se podría reorientar su diseño y entrenamiento para que cuenten con la posibilidad de mensurar la complejidad del contenido generado, y así adaptarlo a las diferentes instancias progresivas del diseño curricular.

Respecto a las posibles arquitecturas de Deep Learning a utilizar, se debería realizar pruebas con el otro tipo de modelos especializados en el tratamiento de datos secuenciales, que son los modelos basados en mecanismos atención, como Transformers. En muchos desarrollos de procesamiento y generación de lenguaje natural, en especial los que es necesario representar dependencias de mayor alcance hay una tendencia a experimentar con este tipo de arquitectura. El entrenamiento y la validación de la performance de modelos cuya arquitectura sea de mayor complejidad traerán aparejados naturalmente la necesidad de contar con recursos tanto de hardware y de software de mayor envergadura, con el consiguiente incremento de los costos asociados a su uso y explotación.

Una aspiración de mayor proyección y alcance sería la de utilizar este tipo de desarrollos y los métodos de evaluación aquí implementados y desarrollados, para la generación de contenido educativo en áreas asociadas a la enseñanza de lenguas, como son el análisis sintáctico, la redacción, la creación poética, etc. Al compartir los mismos lineamientos generales por ser ambos lenguajes simbólicos esta posibilidad abre horizontes de aplicación de gran proyección.

Bibliografía

David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for Boltzmann machines. Cognitive science, 9(1) 147–169, 1985.

Adrian, J.A., Páez, D. y Álvarez, J. (1996). Art, Emotion and cognition: Vygotskian and current approaches to musical induction and changes in mood, and cognitive complexization. Psicothema, 8(1),107-118

Agrawal, R., Gollapudi, S., Kannan, A., & Kenthapadi, K. (2014). "Study navigator: An algorithmically generated aid for learning from electronic textbooks". JEDM-Journal of Educational Data Mining, 6(1), 53–75.

Akombo, D., & Lewis, A.J. (2019). The Benefits of Music Software in the Music Classroom: Expropriating Technology. In Pérez-Aldeguer, S., & Akombo, D. (Eds.), Research, technology and best practices in Education. (pp. 1-17). Eindhoven, NL: Adaya Press

Al Emran M., Shaalan K. (2014). "A Survey of Intelligent Language Tutoring Systems". Proceedings of the 2014 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2014.

Charles Ames . Automated composition in retrospect: 1956-1986. Leonardo, 20(2):169–185, 1987.

Arana, Carlos. Bossa Nova Guitar - Book/Audio. Hal Leonard Publishing, Milwaukee, USA (2008) ISBN 10: 1423425197 ISBN 13: 9781423425199

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate.arXiv preprint arXiv:1409.0473, 2014

Leonard E Baum,, Ted Petrie, George Soules, and Norman Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. The annals of mathematical statistics, 41(1):164–171, 1970

Christopher M. Bishop - Pattern Recognition and *Machine Learning* , ISBN-10 : 0387310738, Springer 2006

Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In ICML, pages 1881–1888, 2012

Brusilovsky, P. and Peylo, C. (2003)" Adaptive and intelligent Web-based educational systems". International Journal of Artificial Intelligence in Education 13 (2-4), 159-172

Brusilovsky, P. and Peylo, C. (2003)" Adaptive and intelligent Web-based educational systems". International Journal of Artificial Intelligence in Education 13 (2-4), 159-172

Massimo Caccia, Lucas Caccia, William Fedus, Hugo Larochelle, Joelle Pineau, and Laurent Charlin. Language gans falling short. In Critiquing and Correcting Trends in *Machine Learning*: NeurIPS 2018 Workshop, 2018

Danhauser, Rodolphe, H. Lemoine, G. Carulli, and Schneitzhoeffer. Solfège des solfèges 1910, G. Schirmer - New York OL25506838M

Charniak, E. 2018. Introduction to deep learning - ISBN-13 : 978-0262039512 – MIT press 2018

M. Chassignol, A. Khoroshavin, A. Klimova, and A. Bilyatdinova (2018). "Artificial intelligence trends in education: A narrative overview,'' Procedia Comput. Sci., vol. 136, pp. 16_24.

Gong Chen, Yan Liu, Sheng-hua Zhong, and Xiang Zhang. Musicality-novelty generative adversarial nets for algorithmic composition. In Proceedings of the 26th ACM international conference on Multimedia, pages 1607–1615, 2018.

Yunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, FethiBougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation.arXiv preprint ar-Xiv:1406.1078, 2014

David Cope. The algorithmic composer, volume 16. AR Editions, Inc., 2000

Shuqi Dai, Zheng Zhang, and Gus Guangyu Xia. Music style transfer issues: A position paper. arXiv preprint arXiv:1803.06841, 2018.

Deloitte Res., Deloitte China, Deloitte Company, (2019). "Global Development of AI-Based Education

Deza M.M., Deza E. Encyclopedia of Distances. Springer; Berlin/Heidelberg, Germany: 2009. (Google Scholar) (Ref list)

Deza M.M., Deza E. Encyclopedia of Distances. Springer; Berlin/Heidelberg, Germany: 2009. (Google Scholar) (Ref list)

Douglas Eck and Juergen Schmidhuber. Finding temporal structure in music: Blues improvisation with lstm recurrent networks. In Proceedings of the 12th IEEE workshop on neural networks for signal processing, pages 747–756. IEEE, 2002.

Jeff Ens and Philippe Pasquier. Mmm: Exploring conditional multi-track music generation with the transformer. arXiv preprint arXiv:2008.06048, 2020.

Frize M., Frasson C. (2000). “Decision-support and intelligent tutoring systems in medical education” Clin. Invest. Med.

García Salas Horacio Alberto , Alexander Gelbukh, Hiram Calvo, and Fernando Galindo Soria. Automatic music composition with simple probabilistic generative grammars. Polibits, (44):59–65, 2011

Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press. http://www.deeplearningbook.org (2016)).

Graves, A. (2013), 'Generating Sequences With Recurrent Neural Networks.', CoRR abs/1308.0850

H Paul Grice. Logic and conversation. In P Cole and J L Morgan (eds.), Speech Acts, volume 3 of Syntax and Semantics, pp. 41–58. Academic Press, 1975.

Gaëtan Hadjeres and Frank Nielsen. Interactive music generation with positional constraints using anticipation-rnns.arXiv preprint arXiv:1709.06404, 2017

Harshvardhan - A comprehensive survey and analysis of generative models in machine learning - Computer Science Review Volume 38, 100285 , 2020

Shunit Haviv Hakimi, Nadav Bhonker, and Ran El-Yaniv. Bebopnet: Deep neural models for personalized jazz improvisations. In ISMIR, pages 828–836, 2020.

Antonio Hernández-Blanco, Boris Herrera-Flores, David Tomás, Borja Navarro-Colorado (2019). "A Systematic Review of Deep Learning Approaches to Educational Data Mining", Complexity, vol. 2019, Article ID 1306039, 22 pages, https://doi.org/10.1155/2019/1306039

Antonio Hernández-Blanco, Boris Herrera-Flores, David Tomás, Borja Navarro-Colorado (2019). "A Systematic Review of Deep Learning Approaches to Educational Data Mining", Complexity, vol. 2019, Article ID 1306039, 22 pages, https://doi.org/10.1155/2019/1306039

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural computation, 9(8):1735–1780, 1997

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, Yejin Choi. The Curious Case of Neural Text Degeneration. Published in ICLR 2020 Subjects: Computation and Language (cs.CL)

Harsh Jhamtani and Taylor Berg-Kirkpatrick. Modeling self-repetition in music generation using generative adversarial networks. In *Machine Learning* for Music Discovery Workshop, ICML, 2019.

Daniel D Johnson. Generating polyphonic music using tied parallel networks. In International conference on evolutionary and biologically inspired music and art, pages 128–143. Springer, 2017

Koedinger K.R. et al. (2013). “New potentials for data-driven inteligent tutoring system development and optimization “. AI Mag. 2013.

S. Kullback. - Information Theory and Statistics. John Wiley & Sons, USA, 1959

William Leavitt. Publisher: Boston : Berklee Press ; Winona, Mn : Distributed by Hal Leonard, 1966. Series: Berklee series. ISBN 0876390114, 9780876390115

Lin J. Divergence measures based on the Shannon entropy. IEEE Trans. Inf. Theory. 1991;37:145–151. doi: 10.1109/18.61115.

Ma W. et al. (2014). “ Intelligent Tutoring Systems and Learning Outcomes: A Meta-Analysis”. J. Educ. Psychol. 2014.

Mahmoud M.H., Abo El-Hamayed S.H. (2016). “An intelligent tutoring system for teaching the grammar of the Arabic language”. J. Electr. Syst. Inf. Technol. 2016.

Huanru Henry Mao, Taylor Shin, and Garrison Cottrell. Deepj: Style-specific music generation. In 2018 IEEE 12th International Conference on Semantic Computing (ICSC), pages 377–382. IEEE, 2018.

Sageev Oore, Ian Simon, Sander Dieleman, Douglas Eck, and Karen Simonyan. This time with feeling: Learning expressive musical performance. Neural Computing and Applications, 32(4):955–967, 2020

Georgia Pike-Rowney (2016) - The past, present and future of music in education - A thesis submitted for the degree of Doctor of Philosophy at The Australian National University.2016

Rallo, R., Gisbert, M., & Salinas, J. (1999). “Using data mining and social networks to analyze the structure and content of educative on-line communities”. Analysis, 468(472), 473.

C. Romero, S. Ventura,M. Pechenizkiy, and R. Baker (2010). “Handbook of educational data mining”. CRC Press.

Roselli, N. D. El lenguaje musical y el lenguaje verbal : Un enfoque histórico-cultural (en línea) En: Fernández, B. Paula Ortiz Ruiz, F. de. López de la Llave, A. (Coord.) (2017) CONΨMUSICA 2017

Sabo K.E. et al. (2013). “ Searching for the two sigma advantage: Evaluating algebra intelligent tutors”. Comput. Human Behav. 2013.

Sason I. Tight bounds for symmetric divergence measures and a new inequality relating f-divergences; Proceedings of the 2015 IEEE Information Theory Workshop (ITW); Jerusalem, Israel. 26 April–1 May 2015; pp. 1–5. (Google Scholar)

C. E. SHANNON, A Mathematical Theory of Communication The Bell System Technical Journal, Vol. 27, pp. 379–423, 623–656, July, October, 1948.

Connor Shorten and Taghi M. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning"" - Journal of Big Data - Springer 2016

Connor Shorten and Taghi M. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning"" - Journal of Big Data - Springer 2016

Sung Y.T. et al. (2016). The effect of online summary assessment and feedback system on the summary writing on 6th graders: The LSA-based technique". Comput. Educ. 2016.

Ilya Sutskever, Oriol Vinyals, Quoc V. Le, Sequence to Sequence Learning with Neural Networks

Computation and Language (cs.CL); Machine Learning (cs.LG), 2014

Peter M Todd. A connectionist approach to algorithmic composition. Computer Music Journal, 13(4):27–43, 1989

Ben Ubovich - Utilization and Effectiveness of Technology in Music Education - DO - 10.13140/RG.2.2.17694.87365 - 2015

VanLehn K. et al. (2002). "The Architecture of Why2-Atlas: A Coach for Qualitative Physics Essay Writing". 6th Int. Conf. ITS 2002 Biarritz, Fr. San Sebastian, Spain, June 2–7, 2002 Proc. 2002.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan NGomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. InAdvancesin neural information processing systems, pages 5998–6008, 2017

Elliot Waite et al. Generating long-term structure in songs and stories. Magenta Bolg: https: //magenta.tensorflow.org/blog/2016/ 07/15/ lookback-rnn-attention-rnn/, 2016

Wang D. et al. (2015). "A problem solving oriented intelligent tutoring system to improve students' acquisition of basic computer skills". Comput. Educ. 2015.

Ronald J Williams and David Zipser. Gradient-based learning algorithms for recurrent.Backpropagation: Theory, architectures, and applications, 433, 1995.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural imagecaption generation with visual attention. InInternational conference on machinelearning, pages 2048–2057, 2015

Yamshchikov and Alexey Tikhonov. Music generation with variational recurrent autoencoder supported by history. arXiv preprint arXiv:1705.05458, 2017."

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy.Hierarchical attention networks for document classification. InProceedings of the2016 conference of the North American chapter of the association for computationallinguistics: human language technologies, pages 1480–1489, 2016

Yang Li-Chia, CHOU Szu-Yu, and YANG Yi-Hsuan. Midinet: A convolutional generative adversarial network forsymbolic-domain music generation. In ISMIR, pages 324–331, 2017."

Ning Zhang. Learning adversarial transformer for symbolic music generation. IEEE Transactions on Neural Networksand Learning Systems, 2020"

Apéndice A: Librerías utilizadas para el proeprocesameinto del corpus, entrenamiento del modelo y generación de contenido sintético

Pandas:

Pandas es una herramienta de análisis y manipulación de datos de código abierto, rápida, potente, flexible y fácil de usar, construida sobre el lenguaje de programación Python. (https://pandas.pydata.org/)

Características fundamentales

- Objeto DataFrame para la manipulación de datos con indexación integrada.

- Herramientas para leer y escribir datos entre estructuras de datos en memoria y diferentes formatos de archivo.

- Alineación de datos y manejo integrado de datos faltantes.

- Remodelación y pivoteo de conjuntos de datos.

- Repartos basados en etiquetas, indexación de fantasía y subconjuntos de grandes conjuntos de datos.

- Inserción y eliminación de columnas en la estructura de datos.

- Motor de agrupación que permite realizar operaciones de división-aplicación-combinación en conjuntos de datos.

- Fusión y unión de conjuntos de datos.

- Indexación jerárquica de ejes para trabajar con datos de alta dimensión en una estructura de datos de baja dimensión.

- Funcionalidad de series temporales: Generación de rangos de fechas y conversiones de frecuencias, estadísticas de ventana móvil, regresiones lineales de ventana móvil, desplazamiento de fechas y desfase.

- Proporciona filtrado de datos.)

Numpy

NumPy es una biblioteca para el lenguaje de programación Python, que añade soporte para matrices y arrays multidimensionales de gran tamaño, junto con una gran colección de funciones matemáticas de alto nivel para operar con estos arrays.(https://en.wikipedia.org/wiki/NumPy)

Rápido y versátil, los conceptos de vectorización, indexación y difusión de NumPy son los estándares de hecho de la computación de matrices hoy en día. NumPy ofrece funciones matemáticas completas, generadores de números aleatorios, rutinas de álgebra lineal, transformadas de Fourier y mucho más. NumPy es compatible con una amplia gama de plataformas de hardware y computación, y funciona bien con bibliotecas distribuidas, de GPU y de matrices dispersas.(https://numpy.org/)

music21:

Music21 es un conjunto de herramientas basado en Python para la musicología asistida por ordenador. La gente utiliza music21 para responder a preguntas de musicología utilizando ordenadores, para estudiar grandes conjuntos de datos de música, para generar ejemplos musicales, para enseñar los fundamentos de la teoría musical, para editar la notación musical, para estudiar la música y el cerebro, y para componer música (tanto algorítmica como directamente).

Tensorflow:

TensorFlow es una plataforma integral de código abierto para el aprendizaje automático. Cuenta con un ecosistema completo y flexible de herramientas, bibliotecas y recursos de la comunidad que permite a los investigadores impulsar el estado del arte en ML y a los desarrolladores construir y desplegar fácilmente aplicaciones potenciadas por ML. Su arquitectura flexible permite el fácil despliegue de los cálculos a través de una variedad de plataformas (CPUs, GPUs, TPUs), y desde ordenadores de sobremesa hasta clusters de servidores y dispositivos móviles y de borde.

Los cálculos de TensorFlow se expresan como gráficos de flujo de datos con estado. El nombre TensorFlow deriva de las operaciones que estas redes neuronales realizan sobre matrices de datos multidimensionales, que se denominan tensores

(https://www.tensorflow.org/)

json:

Librería especializada en el tratamiento de archivos JSON. **JSON** (JavaScript Object Notation) es un formato ligero de intercambio de datos. Es fácil de leer y escribir para los humanos. Es fácil de analizar y generar para las máquinas. Se basa en un subconjunto del estándar de lenguaje de programación JavaScript ECMA-262 3ª edición - diciembre de 1999. JSON es un formato de texto completamente independiente del lenguaje, pero utiliza convenciones que resultan familiares a los programadores de la familia de lenguajes C, como C, C++, C#, Java, JavaScript, Perl, Python y muchos otros. Estas propiedades hacen de JSON un lenguaje de intercambio de datos ideal. (https://www.json.org/json-en.html)

JSON se basa en dos estructuras:

- Una colección de pares nombre/valor. En varios lenguajes, esto se realiza como un objeto, un registro, una estructura, un diccionario, una tabla hash, una lista con claves o una matriz asociativa.

- Una lista ordenada de valores. En la mayoría de los lenguajes, se realiza como una matriz, un vector, una lista o una secuencia.

os: Interfaces misceláneas del sistema operativo.

Este módulo provee una manera versátil de usar funcionalidades dependientes del sistema operativo. Si quieres leer o escribir un archivo mira open(), si quieres manipular rutas, mira el módulo os.path, y si quieres leer todas las líneas en todos los archivos en la línea de comandos mira el módulo fileinput. Para crear archivos temporales y directorios mira el módulo tempfile, y para el manejo de alto nivel de archivos y directorios puedes ver el módulo shutil.(https://docs.python.org/es/3.10/library/os.html)

Apéndice B Tablas de Frecuencias Absolutas utilizada para la determinación de las Distribuciones de Probabilidad Empíricas de las Métricas utilizadas para la Evaluación Objetiva

1) Cant. de Alteraciones

Distribuciones Probabilidad empírica de la métrica: Cant Alteraciones por Compas

Corpus

Cant. Alt.	Prob.
0	0.5896
1	0.224
2	0.1006
3	0.0498
4	0.0206
5	0.0091
6	0.0035
7	0.0017
8	0.0007
9	0.0
10	0.0
11	0.0
12	0.0
13	0.0
14	0.0
15	0.0
16	0.0

Temp: 0,1

Cant. Alt.	Prob.
0	0.8614
1	0.0834
2	0.0053
3	0.0044
4	0.0214
5	0.0002
6	0.0001
7	0.0001
8	0.0237
9	0.0
10	0.0
11	0.0
12	0.0
13	0.0
14	0.0
15	0.0
16	0.0

Temp: 0,2

Cant. Alt.	Prob.
0	0.8934
1	0.0857
2	0.0067
3	0.0015
4	0.0127
5	0.0
6	0.0
7	0.0
8	0.0
9	0.0
10	0.0
11	0.0
12	0.0
13	0.0
14	0.0
15	0.0
16	0.0

Temp: 0,3

Cant. Alt.	Prob.
0	0.8898
1	0.0964
2	0.0106
3	0.0022
4	0.0009
5	0.0
6	0.0
7	0.0
8	0.0
9	0.0
10	0.0
11	0.0
12	0.0
13	0.0
14	0.0
15	0.0
16	0.0

Temp: 0,4

Cant. Alt.	Prob.
0	0.8428
1	0.1064
2	0.0146
3	0.0023
4	0.0006
5	0.0002
6	0.0011
7	0.008
8	0.0149
9	0.0091
10	0.0
11	0.0

Temp: 0,5

Cant. Alt.	Prob.
0	0.7838
1	0.1349
2	0.024
3	0.004
4	0.0013
5	0.0011
6	0.0072
7	0.0246
8	0.0191
9	0.0
10	0.0
11	0.0

Temp: 0,6

Cant. Alt.	Prob.
0	0.7634
1	0.151
2	0.0247
3	0.0093
4	0.0261
5	0.0021
6	0.007
7	0.0101
8	0.0062
9	0.0
10	0.0
11	0.0

Temp: 0,7

Cant. Alt.	Prob.
0	0.7186
1	0.1767
2	0.0386
3	0.0104
4	0.0048
5	0.0068
6	0.0129
7	0.0212
8	0.0098
9	0.0
10	0.0
11	0.0

12	0.0
13	0.0
14	0.0
15	0.0
16	0.0

12	0.0
13	0.0
14	0.0
15	0.0
16	0.0

12	0.0
13	0.0
14	0.0
15	0.0
16	0.0

12	0.0
13	0.0
14	0.0
15	0.0
16	0.0

Temp: 0,8

Cant. Alt.	Prob.
0	0.6334
1	0.1968
2	0.0487
3	0.0157
4	0.0134
5	0.0167
6	0.0331
7	0.0326
8	0.0097
9	0.0
10	0.0
11	0.0
12	0.0
13	0.0
14	0.0
15	0.0
16	0.0

Temp: 0,9

Cant. Alt.	Prob.
0	0.6528
1	0.2349
2	0.0702
3	0.0209
4	0.0082
5	0.0044
6	0.0062
7	0.0016
8	0.0005
9	0.0
10	0.0002
11	0.0
12	0.0
13	0.0
14	0.0
15	0.0
16	0.0

Temp: 1,0

Cant. Alt.	Prob.
0	0.6076
1	0.262
2	0.0777
3	0.0266
4	0.0117
5	0.0059
6	0.0056
7	0.0024
8	0.0005
9	0.0
10	0.0
11	0.0
12	0.0
13	0.0
14	0.0
15	0.0
16	0.0

Temp: 1,1

Cant. Alt.	Prob.
0	0.566
1	0.2901
2	0.0991
3	0.033
4	0.0047
5	0.0024
6	0.0047
7	0.0
8	0.0
9	0.0
10	0.0
11	0.0
12	0.0
13	0.0
14	0.0
15	0.0
16	0.0

Temp: 1,2

Cant. Alt.	Prob.
0	0.4392
1	0.3113
2	0.1215
3	0.064
4	0.0256
5	0.0149
6	0.0171
7	0.0043
8	0.0021
9	0.0
10	0.0
11	0.0
12	0.0
13	0.0
14	0.0

Temp: 1,3

Cant. Alt.	Prob.
0	0.4016
1	0.3614
2	0.1486
3	0.0442
4	0.0281
5	0.008
6	0.008
7	0.0
8	0.0
9	0.0
10	0.0
11	0.0
12	0.0
13	0.0
14	0.0

Temp: 1,4

Cant. Alt.	Prob.
0	0.399
1	0.3543
2	0.1549
3	0.0682
4	0.0184
5	0.0052
6	0.0
7	0.0
8	0.0
9	0.0
10	0.0
11	0.0
12	0.0
13	0.0
14	0.0

Temp: 1,5

Cant. Alt.	Prob.
0	0.381
1	0.319
2	0.1857
3	0.0619
4	0.0333
5	0.0143
6	0.0048
7	0.0
8	0.0
9	0.0
10	0.0
11	0.0
12	0.0
13	0.0
14	0.0

15	0.0	15	0.0	15	0.0	15	0.0
16	0.0	16	0.0	16	0.0	16	0.0

Distribuciones Probabilidad empírica de la métrica: Salto Interválico Máximo

Distribución de Probabilidad Empírica de la Variable

Corpus

Salto Int. Máx.	Prob.
0	0.0982
1	0.091
2	0.281
3	0.1645
4	0.1189
5	0.1306
6	0.0172
7	0.034
8	0.0212
9	0.0226
10	0.0089
11	0.0004
12	0.0115
13	0.0
14	0.0
15	0.0
16	0.0
17	0.0
18	0.0
19	0.0
20	0.0
21	0.0
22	0.0
23	0.0
4	0.0

Temp: 0,1

Salto Int. Máx.	Prob.
0	0.3165
1	0.1309
2	0.4548
3	0.0513
4	0.0339
5	0.0093
6	0.0
7	0.0015
8	0.0001
9	0.0014
10	0.0001
11	0.0
12	0.0001
13	0.0
14	0.0
15	0.0
16	0.0
17	0.0
18	0.0
19	0.0
20	0.0
21	0.0
22	0.0
23	0.0
24	0.0

Temp: 0,2

Salto Int. Máx.	Prob.
0	0.3145
1	0.1141
2	0.4146
3	0.0918
4	0.0451
5	0.016
6	0.0001
7	0.0021
8	0.0003
9	0.0011
10	0.0001
11	0.0
12	0.0001
13	0.0
14	0.0
15	0.0
16	0.0
17	0.0
18	0.0
19	0.0
20	0.0
21	0.0
22	0.0
23	0.0
24	0.0

Temp: 0,3

Salto Int. Máx.	Prob.
0	0.2646
1	0.1043
2	0.4288
3	0.125
4	0.0445
5	0.025
6	0.0003
7	0.0053
8	0.0005
9	0.001
10	0.0002
11	0.0
12	0.0005
13	0.0
14	0.0
15	0.0
16	0.0
17	0.0
18	0.0
19	0.0
20	0.0
21	0.0
22	0.0
23	0.0
24	0.0

Temp: 0,4

Salto Int. Máx.	Prob.
0	0.2493
1	0.1036
2	0.4255
3	0.1357
4	0.0474

Temp: 0,5

Salto Int. Máx.	Prob.
0	0.1328
1	0.1172
2	0.4294
3	0.1992
4	0.0619

Temp: 0,6

Salto Int. Máx.	Prob.
0	0.1019
1	0.1365
2	0.3942
3	0.2005
4	0.0763

Temp: 0,7

Salto Int. Máx.	Prob.
0	0.0851
1	0.107
2	0.3711
3	0.2288
4	0.0881

5	0.0295	5	0.0421	5	0.0602	5	0.0731
6	0.0004	6	0.0019	6	0.0021	6	0.007
7	0.004	7	0.0108	7	0.017	7	0.0191
8	0.0018	8	0.0019	8	0.0028	8	0.0072
9	0.0014	9	0.0008	9	0.0033	9	0.0046
10	0.0004	10	0.0005	10	0.0014	10	0.0041
11	0.0	11	0.0003	11	0.0005	11	0.0002
12	0.001	12	0.0013	12	0.0028	12	0.0035
13	0.0	13	0.0	13	0.0002	13	0.0002
14	0.0	14	0.0	14	0.0	14	0.0
15	0.0	15	0.0	15	0.0	15	0.0
16	0.0	16	0.0	16	0.0002	16	0.0002
17	0.0	17	0.0	17	0.0	17	0.0
18	0.0	18	0.0	18	0.0	18	0.0002
19	0.0	19	0.0	19	0.0	19	0.0002
20	0.0	20	0.0	20	0.0	20	0.0
21	0.0	21	0.0	21	0.0	21	0.0002
22	0.0	22	0.0	22	0.0	22	0.0
23	0.0	23	0.0	23	0.0	23	0.0
24	0.0	24	0.0	24	0.0002	24	0.0

Distribuciones Probabilidad empírica de la métrica: Ratio Cant Fig. Diferentes

Distribución de Probabilidad Empírica de la Variable

Corpus

Ratio	Prob.
0 - 0.05	0.0
0.05 - 0.1	0.0
0.1 - 0.15	0.0781
0.15 - 0.2	0.1446
0.2 - 0.25	0.1476
0.25 - 0.3	0.1102
0.3 - 0.35	0.1483
0.35 - 0.4	0.0655
0.4 - 0.45	0.0023
0.45 - 0.5	0.1568
0.5 - 0.55	0.0
0.55 - 0.6	0.0118
0.6 - 0.65	0.0
0.65 - 0.7	0.0607
0.7 - 0.75	0.0236
0.75 - 0.8	0.0012
0.8 - 0.85	0.0
0.85 - 0.9	0.0
0.9 - 0.95	0.0
0.95 - 1	0.0492

Temp: 0,1

Ratio	Prob.
0 - 0.05	0.0
0.05 - 0.1	0.0139
0.1 - 0.15	0.0981
0.15 - 0.2	0.0585
0.2 - 0.25	0.1284
0.25 - 0.3	0.0503
0.3 - 0.35	0.0838
0.35 - 0.4	0.0413
0.4 - 0.45	0.0
0.45 - 0.5	0.1711
0.5 - 0.55	0.0
0.55 - 0.6	0.0273
0.6 - 0.65	0.0
0.65 - 0.7	0.0668
0.7 - 0.75	0.0516
0.75 - 0.8	0.0001
0.8 - 0.85	0.0
0.85 - 0.9	0.0
0.9 - 0.95	0.0
0.95 - 1	0.2089

Temp: 0,2

Ratio	Prob.
0 - 0.05	0.0
0.05 - 0.1	0.0069
0.1 - 0.15	0.0373
0.15 - 0.2	0.072
0.2 - 0.25	0.1274
0.25 - 0.3	0.0643
0.3 - 0.35	0.0792
0.35 - 0.4	0.0442
0.4 - 0.45	0.0005
0.45 - 0.5	0.1647
0.5 - 0.55	0.0
0.55 - 0.6	0.0202
0.6 - 0.65	0.0
0.65 - 0.7	0.0722
0.7 - 0.75	0.0559
0.75 - 0.8	0.0005
0.8 - 0.85	0.0
0.85 - 0.9	0.0
0.9 - 0.95	0.0
0.95 - 1	0.2546

Temp: 0,3

Ratio	Prob.
0 - 0.05	0.0
0.05 - 0.1	0.0
0.1 - 0.15	0.0463
0.15 - 0.2	0.067
0.2 - 0.25	0.1344
0.25 - 0.3	0.0585
0.3 - 0.35	0.0762
0.35 - 0.4	0.0547
0.4 - 0.45	0.0001
0.45 - 0.5	0.1823
0.5 - 0.55	0.0
0.55 - 0.6	0.02
0.6 - 0.65	0.0
0.65 - 0.7	0.0862
0.7 - 0.75	0.0623
0.75 - 0.8	0.0006
0.8 - 0.85	0.0
0.85 - 0.9	0.0
0.9 - 0.95	0.0
0.95 - 1	0.2114

Temp: 0,4

Ratio	Prob.
0 - 0.05	0.0
0.05 - 0.1	0.0118
0.1 - 0.15	0.0515
0.15 - 0.2	0.0832
0.2 - 0.25	0.1313
0.25 - 0.3	0.0737
0.3 - 0.35	0.0802
0.35 - 0.4	0.0599
0.4 - 0.45	0.0005
0.45 - 0.5	0.1818
0.5 - 0.55	0.0
0.55 - 0.6	0.0159
0.6 - 0.65	0.0
0.65 - 0.7	0.0716
0.7 - 0.75	0.0483

Temp: 0,5

Ratio	Prob.
0 - 0.05	0.0
0.05 - 0.1	0.0
0.1 - 0.15	0.0907
0.15 - 0.2	0.1024
0.2 - 0.25	0.1039
0.25 - 0.3	0.083
0.3 - 0.35	0.0927
0.35 - 0.4	0.0568
0.4 - 0.45	0.0005
0.45 - 0.5	0.1677
0.5 - 0.55	0.0
0.55 - 0.6	0.0173
0.6 - 0.65	0.0
0.65 - 0.7	0.0722
0.7 - 0.75	0.0508

Temp: 0,6

Ratio	Prob.
0 - 0.05	0.0
0.05 - 0.1	0.0176
0.1 - 0.15	0.0805
0.15 - 0.2	0.1069
0.2 - 0.25	0.1074
0.25 - 0.3	0.0839
0.3 - 0.35	0.0982
0.35 - 0.4	0.0598
0.4 - 0.45	0.0012
0.45 - 0.5	0.1637
0.5 - 0.55	0.0
0.55 - 0.6	0.0197
0.6 - 0.65	0.0
0.65 - 0.7	0.0761
0.7 - 0.75	0.051

Temp: 0,7

Ratio	Prob.
0 - 0.05	0.0
0.05 - 0.1	0.0
0.1 - 0.15	0.1125
0.15 - 0.2	0.1063
0.2 - 0.25	0.1183
0.25 - 0.3	0.0891
0.3 - 0.35	0.1032
0.35 - 0.4	0.0637
0.4 - 0.45	0.0011
0.45 - 0.5	0.1509
0.5 - 0.55	0.0
0.55 - 0.6	0.0205
0.6 - 0.65	0.0
0.65 - 0.7	0.0749
0.7 - 0.75	0.0494

0.75 - 0.8	0.0009	0.75 - 0.8	0.0005	0.75 - 0.8	0.0006	0.75 - 0.8	0.0005
0.8 - 0.85	0.0	0.8 - 0.85	0.0	0.8 - 0.85	0.0	0.8 - 0.85	0.0
0.85 - 0.9	0.0	0.85 - 0.9	0.0	0.85 - 0.9	0.0	0.85 - 0.9	0.0
0.9 - 0.95	0.0	0.9 - 0.95	0.0	0.9 - 0.95	0.0	0.9 - 0.95	0.0
0.95 - 1	0.1894	0.95 - 1	0.1615	0.95 - 1	0.1334	0.95 - 1	0.1098

Temp: 0,8

Ratio	Prob.
0 - 0.05	0.0
0.05 - 0.1	0.0046
0.1 - 0.15	0.1479
0.15 - 0.2	0.1202
0.2 - 0.25	0.11
0.25 - 0.3	0.0974
0.3 - 0.35	0.1084
0.35 - 0.4	0.0608
0.4 - 0.45	0.0015
0.45 - 0.5	0.135
0.5 - 0.55	0.0
0.55 - 0.6	0.0239
0.6 - 0.65	0.0
0.65 - 0.7	0.0659
0.7 - 0.75	0.0431
0.75 - 0.8	0.0017
0.8 - 0.85	0.0
0.85 - 0.9	0.0
0.9 - 0.95	0.0
0.95 - 1	0.0796

Temp: 0,9

Ratio	Prob.
0 - 0.05	0.0
0.05 - 0.1	0.0012
0.1 - 0.15	0.1031
0.15 - 0.2	0.134
0.2 - 0.25	0.1242
0.25 - 0.3	0.1139
0.3 - 0.35	0.1137
0.35 - 0.4	0.0669
0.4 - 0.45	0.0011
0.45 - 0.5	0.1322
0.5 - 0.55	0.0
0.55 - 0.6	0.0217
0.6 - 0.65	0.0
0.65 - 0.7	0.0597
0.7 - 0.75	0.039
0.75 - 0.8	0.0012
0.8 - 0.85	0.0
0.85 - 0.9	0.0
0.9 - 0.95	0.0
0.95 - 1	0.088

Temp: 1

Ratio	Prob.
0 - 0.05	0.0
0.05 - 0.1	0.0001
0.1 - 0.15	0.1457
0.15 - 0.2	0.1429
0.2 - 0.25	0.1125
0.25 - 0.3	0.118
0.3 - 0.35	0.1177
0.35 - 0.4	0.0626
0.4 - 0.45	0.0019
0.45 - 0.5	0.1191
0.5 - 0.55	0.0
0.55 - 0.6	0.0233
0.6 - 0.65	0.0
0.65 - 0.7	0.0548
0.7 - 0.75	0.0345
0.75 - 0.8	0.0005
0.8 - 0.85	0.0
0.85 - 0.9	0.0
0.9 - 0.95	0.0
0.95 - 1	0.0664

Temp: 1,1

Ratio	Prob.
0 - 0.05	0.0
0.05 - 0.1	0.004
0.1 - 0.15	0.175
0.15 - 0.2	0.1279
0.2 - 0.25	0.1036
0.25 - 0.3	0.1211
0.3 - 0.35	0.0915
0.35 - 0.4	0.0485
0.4 - 0.45	0.0054
0.45 - 0.5	0.14
0.5 - 0.55	0.0
0.55 - 0.6	0.0229
0.6 - 0.65	0.0
0.65 - 0.7	0.0646
0.7 - 0.75	0.0417
0.75 - 0.8	0.0027
0.8 - 0.85	0.0
0.85 - 0.9	0.0
0.9 - 0.95	0.0
0.95 - 1	0.0511

Temp: 1,2

Ratio	Prob.
0 - 0.05	0.0
0.05 - 0.1	0.0
0.1 - 0.15	0.2441
0.15 - 0.2	0.1759
0.2 - 0.25	0.101
0.25 - 0.3	0.1575
0.3 - 0.35	0.0997
0.35 - 0.4	0.0577
0.4 - 0.45	0.0013
0.45 - 0.5	0.0696

Temp: 1,3

Ratio	Prob.
0 - 0.05	0.0
0.05 - 0.1	0.0
0.1 - 0.15	0.2441
0.15 - 0.2	0.1759
0.2 - 0.25	0.101
0.25 - 0.3	0.1575
0.3 - 0.35	0.0997
0.35 - 0.4	0.0577
0.4 - 0.45	0.0013
0.45 - 0.5	0.0696

Temp: 1,4

Ratio	Prob.
0 - 0.05	0.0
0.05 - 0.1	0.0
0.1 - 0.15	0.206
0.15 - 0.2	0.195
0.2 - 0.25	0.1132
0.25 - 0.3	0.1745
0.3 - 0.35	0.1242
0.35 - 0.4	0.0283
0.4 - 0.45	0.0047
0.45 - 0.5	0.0629

Temp: 1,5

Ratio	Prob.
0 - 0.05	0.0
0.05 - 0.1	0.0142
0.1 - 0.15	0.2336
0.15 - 0.2	0.1681
0.2 - 0.25	0.0969
0.25 - 0.3	0.1225
0.3 - 0.35	0.1168
0.35 - 0.4	0.0456
0.4 - 0.45	0.0
0.45 - 0.5	0.0912

0.5 - 0.55	0.0	0.5 - 0.55	0.0	0.5 - 0.55	0.0	0.5 - 0.55	0.0
0.55 - 0.6	0.0105	0.55 - 0.6	0.0105	0.55 - 0.6	0.0189	0.55 - 0.6	0.0199
0.6 - 0.65	0.0	0.6 - 0.65	0.0	0.6 - 0.65	0.0	0.6 - 0.65	0.0
0.65 - 0.7	0.0302	0.65 - 0.7	0.0302	0.65 - 0.7	0.0204	0.65 - 0.7	0.0313
0.7 - 0.75	0.0197	0.7 - 0.75	0.0197	0.7 - 0.75	0.0189	0.7 - 0.75	0.0199
0.75 - 0.8	0.0013	0.75 - 0.8	0.0013	0.75 - 0.8	0.0016	0.75 - 0.8	0.0
0.8 - 0.85	0.0	0.8 - 0.85	0.0	0.8 - 0.85	0.0	0.8 - 0.85	0.0
0.85 - 0.9	0.0	0.85 - 0.9	0.0	0.85 - 0.9	0.0	0.85 - 0.9	0.0
0.9 - 0.95	0.0	0.9 - 0.95	0.0	0.9 - 0.95	0.0	0.9 - 0.95	0.0
0.95 - 1	0.0315	0.95 - 1	0.0315	0.95 - 1	0.0314	0.95 - 1	0.0399

Apéndice C: Muestras del Código Utilizado

Preprocesamiento, Entrenamiento y Creación de melodías

El preprocesado del corpus completo, el armado del modelo, su entrenamiento, las inferencias y la generación de melodías para el posterior análisis se llevaron a cabo en la plataforma colaborativa Google Colab (siempre utilizando la plataforma interactivas de programación Jupyter Notebook) y en Sage Maker, la plataforma en la nube de Amazon Web Services que permite crear entrenar y poner en producción modelos basados en machine Learning.

Aquí capturas de pantallas del código de preprocesamiento completo (incluyendo semillas, desarrollo de modelos, entrenamiento, inferencias y creación de melodías.

```
#================================
# Librerías y paquetes a importar
#================================
import os
import json
import music21 as m21
from music21 import *
import numpy as np
import tensorflow.keras as keras
import pandas as pd


#======================
# Parámetros básicos
#======================


os.chdir('/content/')
wd = os.getcwd()


XML_DATASET_PATH = wd + "/Cancionero_XML"
SAVE_DIR =  wd + "Dataset"
FILE_DATASET_PATH = wd + "/Guarda_toda_cancion"
MAPPING_PATH = wd + "/mapping.json"
SEQUENCE_LENGTH = 64
GUARDA_INPUTS_TARGETS_DIR = wd + "/Guarda inputs y targets/"
ACCEPTABLE_DURATIONS = [0, 0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0, 2.25, 2.5, 2.75, 3.0, 3.25, 3.5, 3.75, 4.0 ]
```

```
#======================
# Funciones
#======================

# 1) Carga de canciones utilizando music21

def load_songs_in_xml(dataset_path):

    songs = []
    nombres = []

    for path, subdirs, files in os.walk(dataset_path):
        for file in files:
            if file[-3:] == "xml":
                song = m21.converter.parse(os.path.join(path, file))
                songs.append(song)
                nombres.append(os.path.splitext(file)[0])

    return songs, nombres


```

```python
# 2) Armado de partes de canciones utilizando music21

def arma_songs_partes(songs, nombres):

    partes = []
    nombres_partes = []


    for i in range(len(songs)):  #i=15

        song = songs[i]
        song_flat = song.flat

        parte = stream.Stream()
        nombre_parte = nombres[i] + '_parte_A'

        for j in range(len(song_flat)):

            if isinstance(song_flat[j] , expressions.RehearsalMark) and song_flat[j].content != 'A':

                partes.append(parte)
                nombres_partes.append(nombre_parte)

                parte = stream.Stream()
                nombre_parte = nombres[i] + '_parte_' + song_flat[j].content

            else:
                parte.append(song_flat[j])


        partes.append(parte)
        nombres_partes.append(nombre_parte)

    return partes, nombres_partes
```

```python
# 3) Chequea si tiene duraciones aceptables
def has_acceptable_durations(song, acceptable_durations):
    for note in song.flat.notesAndRests:
        if note.duration.quarterLength not in acceptable_durations:
            return False
    return True


# 4) Transpone la canción
def transpose(song):

    key = song.analyze("key")

    if key.mode == "major":
        interval = m21.interval.Interval(key.tonic, m21.pitch.Pitch("C"))
    elif key.mode == "minor":
        interval = m21.interval.Interval(key.tonic, m21.pitch.Pitch("A"))

    tranposed_song = song.transpose(interval)

    return tranposed_song

```

```python
# 5) Codifica la canción en representación simbólica elegida
def encode_song(song, time_step=0.25):

    encoded_song = []

    for event in song.flat.notesAndRests:


        if isinstance(event, m21.note.Note):
            symbol = event.pitch.midi # 60

        elif isinstance(event, m21.note.Rest):
            symbol = "r"


        steps = int(event.duration.quarterLength / time_step)
        for step in range(steps):

            if step == 0:
                encoded_song.append(symbol)
            else:
                encoded_song.append("_")

    # convierte la lista a un string
    encoded_song = " ".join(map(str, encoded_song))

    return encoded_song
```

```python
#======================================================================
# Preprocesamiesnto completo y prreparación de dataset de entrenamiento
#======================================================================

def preprocess(dataset_path , sequence_length, file_dataset_path):   # dataset_path= XML_DATASET_PATH  , file_dataset_path = FILE_DAT

    # Carga todas las canciones del corpus en formato XML utilizando la función load_songs_in_xml (punto 1)
    # y devuelve una lista de objetos de stream de music21
    print("Cargando canciones...")
    songs, nombres = load_songs_in_xml(dataset_path)

    # arma las partes de canciones que recibe en formato streams de music21, y
    # devuelve una lista de objetos de partes de canciones de music21
    songs_streams, nombres = arma_songs_partes(songs, nombres)

    #arma el delimitador de partes. Este será un string compuesto por SEQUENCE_LENGTH cantidad de caracteres "/", interponiendo entre
    # de esta manera simulamos la codificación utilizada para las notas de las canciones. En esta caso estaremos indicando una secuen
    #indica el fin de una parte.
    new_song_delimiter = "/ " * sequence_length

    #Inicializa el string que va a contener todads las partes, separadas por el grupo de caracteres new_song_delimiter
    songs_string = ""

    for i, song in enumerate(songs_streams):  #recorre todas las canciones de la lista

        # firltra las canciones que no tienen duraicones aceptables
        if not has_acceptable_durations(song, ACCEPTABLE_DURATIONS):
            print(f"Song '{nombres[i]}' no tiene duraciones aceptables")
            continue

        # si parte posee todas duraicones incluidas en la lista de duraicones aceptable entonces la transpone
        song = transpose(song)
        #print(f"Song '{nombres[i]}' transpuesta")

        # codifica la canción a la represnetación simbòlica elegida
        encoded_song = encode_song(song)

```

```python
# genera las secuencias de entrenamiento
def generate_training_sequences(sequence_length, songs_integers):

    inputs = []
    targets = []

    #se calcala la cantidad de secuencias . Como la secuncia tiene longitud sequence_length, se lo resto
    num_sequences = len(songs_integers) - sequence_length

    for i in range(num_sequences):
        inputs.append(songs_integers[i:i+sequence_length])
        targets.append(songs_integers[i+sequence_length])

    # one-hot encode the sequences
    vocabulary_size = len(set(songs_integers))

    # inputs size: (# of sequences, sequence length, vocabulary size)
    inputs = keras.utils.to_categorical(inputs, num_classes=vocabulary_size)
    targets = np.array(targets)

    print(f"There are {len(inputs)} sequences.")

    return inputs, targets
```

+ Code + Text

```python
#========================
# Entrenamiento del Modelo
#========================

OUTPUT_UNITS = len(set(songs_integers)) #son todos los posibles símbolos que se resultaron de la codificación
NUM_UNITS = [256 , 256]
LOSS = "sparse_categorical_crossentropy"
LEARNING_RATE = 0.001
EPOCHS = 90
BATCH_SIZE = 64
DROPOUT_RATE = 0.2

# model name
model_path = ""
for cell_size in NUM_UNITS:
  model_path += "_" + str(cell_size)


MODEL_NAME =  "model" + model_path + "_SEQ_" + str(SEQUENCE_LENGTH) + "_BTCH_" + str(BATCH_SIZE) + "_DROP_" +
MODEL_PATH =   MODEL_NAME + ".h5"
SAVE_MODEL_PATH = "/content/" + MODEL_PATH


```

```python
#armo el modelo
def build_model(output_units, num_units, loss, learning_rate, dropout_rate):
    # create the model architecture
    input = keras.layers.Input(shape=(None, output_units))

    if len(num_units) == 1:
      x = keras.layers.LSTM(num_units[0])(input)
    else:
      x = keras.layers.LSTM(num_units[0] , return_sequences=True)(input)
      for i in range(1,len(num_units)-1):
        x = keras.layers.LSTM(num_units[0], return_sequences=True)(x)

    #ultimo layer
    x = keras.layers.LSTM(num_units[len(num_units)-1])(x)

    #droput rate layer
    x = keras.layers.Dropout(dropout_rate)(x)

    #output layer
    output = keras.layers.Dense(output_units, activation="softmax")(x)

    #arma el modelo
    model = keras.Model(input, output)

    # compile model
    model.compile(loss=loss,
                  optimizer=keras.optimizers.Adam(lr=learning_rate),
                  metrics=["accuracy"])

    model.summary()

    return model
```

```python
from keras.utils.vis_utils import plot_model
plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)

#Configuro checkpoint
from keras.callbacks import ModelCheckpoint

#GUARDO EL MDOELO PARA CADA EPOCA
filepath = SAVE_MODEL_PATH + "/checkpoint"

checkpoint = ModelCheckpoint(filepath,
                             monitor='val_accuracy',
                             verbose=1,
                             save_best_only = True,
                             save_weights_only = False,
                             mode =  'max',
                             save_freq = "epoch")

callbacks_list = [checkpoint]
```

```python
#Entreno el modelo
def train(output_units=OUTPUT_UNITS, num_units=NUM_UNITS, loss=LOSS, learning_rate=LEARNING_RATE, droput_rate = DROPOUT_RATE):


    model = build_model(output_units, num_units, loss, learning_rate, droput_rate)

    historia = model.fit(inputs, targets, validation_split=0.2, epochs=EPOCHS, batch_size=BATCH_SIZE, verbose=1,
                         callbacks=callbacks_list)

    model.save(SAVE_MODEL_PATH)
```

```python
#Grafico la Historia
print(historia.history.keys())
import matplotlib.pyplot as plt

# historia  accuracy
plt.plot(historia.history['accuracy'])
plt.plot(historia.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# historia  loss
plt.plot(historia.history['loss'])
plt.plot(historia.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```python
def generate_melody(seed, mappings, num_steps, max_sequence_length, modelo, temperature):

    # seed
    seed_list = seed.split()
    melody = seed_list
    seed_list = ["/"] * SEQUENCE_LENGTH + seed_list

    # seed a int
    seed_int = [mappings[symbol] for symbol in seed_list]

    for _ in range(num_steps):
        seed_int = seed_int[-max_sequence_length:]

        onehot_seed = keras.utils.to_categorical(seed_int, num_classes=len(mappings))
        # (1, max_sequence_length, num of symbols in the vocabulary)
        onehot_seed = onehot_seed[np.newaxis, ...]

        # predicccciones
        probabilities = modelo.predict(onehot_seed)[0]
        # [0.1, 0.2, 0.1, 0.6] -> 1
        output_int = sample_with_temperature(probabilities, temperature)

        # actualiza seed
        seed_list.append(output_int)

        output_symbol = [k for k, v in mappings.items() if v == output_int][0]

        # chequea si es final de melodia
        if output_symbol == "/":
            break

        # actualiza
        melody.append(output_symbol)

    return melody
```

```python
def sample_with_temperature(probabilites, temperature):

    predictions = np.log(probabilites) / temperature
    probabilites = np.exp(predictions) / np.sum(np.exp(predictions))

    choices = range(len(probabilites)) # [0, 1, 2, 3]
    index = np.random.choice(choices, p=probabilites)

    return index
```

+ Code +

```python
# guarda la melodia
def save_melody(self, melody, step_duration=0.25, format="midi", file_name=GUARDA_MELO):

        stream = m21.stream.Stream()

        start_symbol = None
        step_counter = 1

        for i, symbol in enumerate(melody):

            # handle case in which we have a note/rest
            if symbol != "_" or i + 1 == len(melody):

                # ensure we're dealing with note/rest beyond the first one
                if start_symbol is not None:

                    quarter_length_duration = step_duration * step_counter # 0.25 * 4 = 1

                    # handle rest
                    if start_symbol == "r":
                        m21_event = m21.note.Rest(quarterLength=quarter_length_duration)

                    # handle note
                    else:
                        m21_event = m21.note.Note(int(start_symbol), quarterLength=quarter_length_duration)

                    stream.append(m21_event)

                    # reset the step counter
                    step_counter = 1

                start_symbol = symbol

            else:
                step_counter += 1

```

Preparación de planillas para creación de métricas

```python
# coding: utf-8

###* ARMA COMPASES
#%pip install music21
import music21 as m21
from music21 import *
import pandas as pd
import numpy as np
import os
import json


ACCEPTABLE_DURATIONS  = np.arange(0.25,4.25, 0.25).tolist()
```

```
def      load_songs_in_xml(dataset_path):      #      dataset_path      =
XML_MELOS_GENERADAS_PATH
    songs = ()
    nombres = ()

    for path, subdirs, files in os.walk(dataset_path):
        for file in files:

            # se fija si son XML, y carga la lista con los temas
            if file(-3:) == "xml":
                song = m21.converter.parse(os.path.join(path, file))
                songs.append(song)
                nombres.append(os.path.splitext(file)(0))

    #return songs, nombres
    return songs, nombres


def load_songs_in_midi(dataset_path):
    """Loads all kern pieces in dataset using music21.
    :param dataset_path (str): Path to dataset
    :return songs (list of m21 streams): List containing all pieces
    """
    songs = ()
    nombres = ()

    for path, subdirs, files in os.walk(dataset_path):
        for file in files:

            # se fija si son MIDI, y carga la lista con los temas
            if file(-3:) == "mid":
                song = m21.converter.parse(os.path.join(path, file))
                songs.append(song)
                nombres.append(os.path.splitext(file)(0))
                print('cargando ' + os.path.splitext(file)(0))

    return songs, nombres



def genera_df_compases(songs, nombres):

     df_compases  =  pd.DataFrame(columns  =  ('Song_ID',  'Tema',  'Compas',
'Nota','Duracion', 'Ubicacion','tonicaAcorde', 'tipoAcorde'))

    for i, song in enumerate(songs): #song = songs(1)
        song = song.parts(0)
        for j in range(1,len(song.getElementsByClass('Measure'))):
            print("Tema: " + str(nombres(i)) + " compas " + str(j))
```

```
            #analiza el compás_i = 1
                                                        compas          =
song.getElementsByClass('Measure')(j)                  #compas.show('text')
nota_silencio="f"

            ubicacion = 0

            for k in range(0,len(compas)):   #compas(8) j=1

                #si es harmony.ChordSymbol cargo el acorde
                if isinstance(compas(k),harmony.ChordSymbol):
                    acorde = compas(k)
                    tonicaAcorde = acorde.root().name
                    tipoAcorde = acorde.chordKind

                else:
                    #recorre el compas y arma el dataframe si es Nota o rest
                                if  isinstance(compas(k),  note.Note)  |
isinstance(compas(k), note.Rest):
                        if isinstance(compas(k), note.Note):
                                nota_midi = compas(k).pitch.midi
                        else:
                                nota_midi = 'r'
                        df_compases = df_compases.append({
                                        'Song_ID': i,
                                        'Tema':nombres(i),
                                        'Compas':j,
                                        'Nota':compas(k).name,
                                        'Midi': nota_midi ,
                                    'Duracion': compas(k).quarterLength,
                                                        'Ubicacion':
ubicacion,
                                        'tonicaAcorde':tonicaAcorde,
                                        'tipoAcorde':tipoAcorde},
                                        ignore_index = True)


                        ubicacion = ubicacion + compas(k).quarterLength

    return df_compases


def genera_df_compases_generadas(songs, nombres):  # songs = SONGS_GEN(0:2)

     df_compases  =  pd.DataFrame(columns  =  ('Song_ID',  'Tema',  'Compas',
'Nota',    'Alter',    'Midi',    'Duracion',    'Ubicacion','tonicaAcorde',
'tipoAcorde'))

    for i, song in enumerate(songs): # song = songs(0)
```

```
        song = song.parts(0)
        for j in range(1,len(song.getElementsByClass('Measure'))):
            print("Tema: " + str(nombres(i)) + " compas " + str(j))
            #analiza el compás_j = 1  j = 1
                                                        compas          =
song.getElementsByClass('Measure')(j)                  #compas.show('text')
nota_silencio="f"

            ubicacion = 0

            for k in range(0,len(compas)):   #compas(8) j=1  k=1

                    #recorre el compas y arma el dataframe si es Nota o rest
                                if  isinstance(compas(k),  note.Note)  |
isinstance(compas(k), note.Rest):
                        if isinstance(compas(k), note.Note):
                                nota_midi = compas(k).pitch.midi
                        else:
                                nota_midi = 'r'

                        #chequea si ed diatónica o alteracion
                         if (compas(k).name(-1:) == '-' or compas(k).name(-
1:) == '#'):
                            Alter = 'alt'
                        else:
                            Alter = 'diat'

                        #print('nota ' + compas(k).name + ' Alter ' + Alter)

                        #arma fila con esa nota
                        df_compases = df_compases.append({
                                        'Song_ID': i,
                                        'Tema':nombres(i),
                                        'Compas':j,
                                        'Nota':compas(k).name,
                                        'Alter': Alter,
                                        'Midi': nota_midi,
                                    'Duracion': compas(k).quarterLength,
                                                        'Ubicacion':
ubicacion,
                                        'tonicaAcorde':'',
                                        'tipoAcorde':''},
                                        ignore_index = True)


                        ubicacion = ubicacion + compas(k).quarterLength

    return df_compases
```

```
# ============================================
# 1) Genera df compases de canciones del corpus
# ============================================

XML_DATASET_PATH = wd + "/Cancionero XML"

SONGS, NOMBRES = load_songs_in_xml(XML_DATASET_PATH) #IMPORTAMNTE: en verdad
hay que transponerlo y chequear subdivisiones

df_compases = genera_df_compases(SONGS , NOMBRES)

df_compases.head()
df_compases.columns.values


#save it to Excel
df_compases.to_excel(wd)



# ============================================
# 2) Genera df compases de melodias generadas
# ============================================
XML_MELOS_GENERADAS_PATH = wd + "/NUEVO - Procesado en Funciones/PRUEBAS
GENERACION MELOS"

#recorre todos los directorios de melodias generadas para diferentes
temperaturas

for i in (("0.1" , "0.2", "0.3", "0.4", "0.5", "0.6", "0.7", "0.8", "0.9",
"1.0")):


    XML_MELOS_GENERADAS_PATH_TEMP_ESPECIFICA =  XML_MELOS_GENERADAS_PATH  +
"/2022-10-09_temp_" + i

                            SONGS_GEN,             NOMBRES_GEN            =
load_songs_in_midi(XML_MELOS_GENERADAS_PATH_TEMP_ESPECIFICA)

       df_compases_generadas   =   genera_df_compases_generadas(SONGS_GEN   ,
NOMBRES_GEN)

                              PATH_EXCEL_COMPASES_GENERADOS_TEMP_ESPECIFICA
=     df_compases_generadas.to_excel(PATH_EXCEL_COMPASES_GENERADOS_TEMP_ESPE
CIFICA)
```

```
#for i in (("2.0" , "3.0", "4.0", "5.0", "6.0", "7.0", "8.0", "9.0", "10")):
for i in np.arange(1.1, 1.6, 0.1):

    XML_MELOS_GENERADAS_PATH_TEMP_ESPECIFICA =  XML_MELOS_GENERADAS_PATH  +
"/2022-10-09_temp_" + str(i)(:3)

                            SONGS_GEN,             NOMBRES_GEN             =
load_songs_in_midi(XML_MELOS_GENERADAS_PATH_TEMP_ESPECIFICA)

       df_compases_generadas   =   genera_df_compases_generadas(SONGS_GEN   ,
NOMBRES_GEN)

                    PATH_EXCEL_COMPASES_GENERADOS_TEMP_ESPECIFICA          =
"C:/Users/Carlos/Desktop/DF_COMPASES/df_compases_generados_TEMP_"          +
str(i)(:3)  + ".xlsx"
    df_compases_generadas.to_excel(PATH_EXCEL_COMPASES_GENERADOS_TEMP_ESPEC
IFICA)
```

Apéndice D: Muestras de Planilla de Notas por Acorde Generada

Del Corpus

	Song_ID	Tema	Compas	Nota	Alter	Midi	Duracion	Ubicacion	TonicaAcorde	TipoAcorde
0	0	A media luz	1	E	diat	76	1	0	A	minor
17430	92	Mi buenos aires querido	2	C	diat	72	0,5	1,5	F	minor
17431	92	Mi buenos aires querido	2	B-	alt	70	0,5	2	B-	dominant
17432	92	Mi buenos aires querido	2	A-	alt	68	1	2,5	B-	dominant
17433	92	Mi buenos aires querido	2	C	diat	72	0,5	3,5	B-	dominant
17434	92	Mi buenos aires querido	3	G	diat	67	4	0	E-	major
17435	92	Mi buenos aires querido	4	rest	diat	r	0,5	0	F	minor
17436	92	Mi buenos aires querido	4	F	diat	65	0,5	0,5	F	minor
17437	92	Mi buenos aires querido	4	A-	alt	68	0,5	1	F	minor
17438	92	Mi buenos aires querido	4	C	diat	72	0,5	1,5	F	minor
17439	92	Mi buenos aires querido	4	G	diat	67	1	2	C	minor
17440	92	Mi buenos aires querido	4	C	diat	72	0,5	3	C	minor
17441	92	Mi buenos aires querido	4	E-	alt	75	0,5	3,5	C	minor
17442	92	Mi buenos aires querido	5	D	diat	74	4	0	G	dominant
17443	92	Mi buenos aires querido	6	C	diat	72	2	0	C	major
17444	92	Mi buenos aires querido	6	rest	diat	r	0,5	2	C	major
17445	92	Mi buenos aires querido	6	E	diat	76	1	2,5	C	major
17446	92	Mi buenos aires querido	6	E	diat	76	0,25	3,5	C	major
17447	92	Mi buenos aires querido	6	D	diat	74	0,25	3,75	C	major
17448	92	Mi buenos aires querido	7	D	diat	74	0,5	0	C	major
17449	92	Mi buenos aires querido	7	C	diat	72	0,5	0,5	C	major
17450	92	Mi buenos aires querido	7	C	diat	72	0,5	1	C	major
17451	92	Mi buenos aires querido	7	B	diat	71	0,5	1,5	C	major
17452	92	Mi buenos aires querido	7	B	diat	71	0,5	2	C#	diminished
17453	92	Mi buenos aires querido	7	A	diat	69	0,5	2,5	C#	diminished
17454	92	Mi buenos aires querido	7	A	diat	69	0,5	3	C#	diminished
17455	92	Mi buenos aires querido	7	G	diat	67	0,5	3,5	C#	diminished
17456	92	Mi buenos aires querido	8	F	diat	65	1	0	D	minor
17457	92	Mi buenos aires querido	8	rest	diat	r	1	1	D	minor
17458	92	Mi buenos aires querido	8	rest	diat	r	0,5	2	D	minor
17459	92	Mi buenos aires querido	8	D	diat	74	0,5	2,5	D	minor
17460	92	Mi buenos aires querido	8	D	diat	74	0,5	3	D	minor
17461	92	Mi buenos aires querido	8	C	diat	72	0,5	3,5	D	minor
17462	92	Mi buenos aires querido	9	C	diat	72	0,5	0	G	dominant
17463	92	Mi buenos aires querido	9	B	diat	71	0,5	0,5	G	dominant
17464	92	Mi buenos aires querido	9	B	diat	71	0,5	1	G	dominant

Generada para una temperatura especifica

	A	B	C	D	E	F	G	H	I	J	K
1		Song_ID	Tema	Compas	Nota	Alter	Midi	Duracion	Ubicacion	tonicaAcorde	tipoAcorde
752	750	2	melody__temp_1.2	40	E	diat	76	0,5	1		
753	751	2	melody__temp_1.2	40	D	diat	74	0,5	1,5		
754	752	2	melody__temp_1.2	40	C	diat	72	0,5	2		
755	753	2	melody__temp_1.2	40	D	diat	74	0,5	2,5		
756	754	2	melody__temp_1.2	40	C	diat	72	0,5	3		
757	755	2	melody__temp_1.2	40	C	diat	72	0,5	3,5		
758	756	2	melody__temp_1.2	41	E	diat	76	0,5	0		
759	757	2	melody__temp_1.2	41	C	diat	72	0,5	0,5		
760	758	2	melody__temp_1.2	41	A	diat	69	0,5	1		
761	759	2	melody__temp_1.2	41	B	diat	71	0,5	1,5		
762	760	2	melody__temp_1.2	41	B	diat	71	0,5	2		
763	761	2	melody__temp_1.2	41	C	diat	72	0,5	2,5		
764	762	2	melody__temp_1.2	41	D	diat	74	1	3		
765	763	2	melody__temp_1.2	42	F	diat	77	0,5	0		
766	764	2	melody__temp_1.2	42	E	diat	76	0,5	0,5		
767	765	2	melody__temp_1.2	42	D	diat	74	0,5	1		
768	766	2	melody__temp_1.2	42	C	diat	72	0,5	1,5		
769	767	2	melody__temp_1.2	42	D	diat	74	0,5	2		
770	768	2	melody__temp_1.2	42	E	diat	76	0,5	2,5		
771	769	2	melody__temp_1.2	42	D	diat	74	1	3		
772	770	2	melody__temp_1.2	43	F	diat	77	0,5	0		
773	771	2	melody__temp_1.2	43	E	diat	76	0,5	0,5		
774	772	2	melody__temp_1.2	43	D	diat	74	0,5	1		
775	773	2	melody__temp_1.2	43	C	diat	72	0,5	1,5		
776	774	2	melody__temp_1.2	43	B	diat	71	0,5	2		
777	775	2	melody__temp_1.2	43	C	diat	72	0,5	2,5		
778	776	2	melody__temp_1.2	43	B	diat	71	0,5	3		
779	777	2	melody__temp_1.2	43	B-	alt	70	0,5	3,5		
780	778	2	melody__temp_1.2	44	B	diat	71	0,5	0		

Apéndice E: Muestras del Código Utilizado para la Creación de Métricas

```
import os
import json
import music21 as m21
from music21 import *
import numpy as np
import math
import pandas as pd


#PRUEBAS DISTROBUCION DE ALTERAIONES
#============================================
#Cantidad devariable  por compás df_1 - CORPUS
#============================================

def distrib_prob(df, var_bernoulli, exito, cant_clases):
    mappings_1 = {}

    for i in range(len(cant_clases)+1):
        mappings_1(i) = 0


    #recorre las canciones
    for i in range(len(df('Song_ID').unique())):   # i=0
           print("Analizando  canción:  "  +  df.loc(df('Song_ID')==i,
'Tema').iloc(0))


        #recorres los compases de esa canción i     i=0  j=1
                 for   j   in   range(len(df.loc(df('Song_ID')   ==
i)('Compas').unique())) :
           #inicializa la variable binomial que va acontar llos exitos
en la variable bernoulkii
            x = 0

            #recorre las notas de ese compas j de la canción i   i=0
j=1  k=1
                for  k  in  range(len(df.loc((df('Song_ID')  ==  i)  &
(df('Compas') == j+1)))):

               #recorro el compás y actualizao el diccionario de esa
variable
               variable = df.loc((df('Song_ID') == i) & (df('Compas')
== j+1) , var_bernoulli).iloc(k)
```

```
                if variable == exito:
                    x += 1


            #actualiza el mapping sumandole una frecuencia al valor
de la variable binoliam observadop
            mappings_1(x) = mappings_1(x) + 1


    total_compases = 0
    distr_prob = ()

    for i in sorted(mappings_1):
        total_compases += mappings_1(i)

    for i in sorted(mappings_1):
        distr_prob.append(mappings_1(i)/ total_compases)

    return distr_prob


#==================================
# PRUEBA DE JENSEN-SHANNON DIVERGENCE
#==================================

#calculate the kl divergence

def kl_divergence(p, q):

    KL = 0
    for i in range(len(p)):
        if not (p(i) == 0 or q(i) == 0):
            KL +=  p(i) * math.log(p(i)/q(i),2.0)

    return KL

# calculate the js divergence
def js_divergence(p, q):
    m = 0.5 * (p + q)
    return 0.5 * kl_divergence(p, m) + 0.5 * kl_divergence(q, m)


def calcula_JS(p, q):


    # define distributions
    p = np.asarray(p)
    q = np.asarray(q)
```

```
    # calculate JS(P || Q)
    JS_pq = js_divergence(p, q)
    distance_JS_pq = math.sqrt(JS_pq)
                                5
    #print('JS(P || Q) divergence: %.3f bits' % js_pq)

    #print('JS(Q || P) distance: %.3f' % math.sqrt(js_pq))

    return  JS_pq , distance_JS_pq



#----------------------------------------
# Corre la función en compases del CORPÙS
#----------------------------------------

wd  =  "C:/Users/Carlos/Desktop/Doctorado  UNLZ/Tesis  -  CODIGO  y
REFERENCIA/000 CODIGO DEFINITIVO/2022-10-08 ULTIMO/NUEVO - Procesado
en Funciones/DF_COMPASES"
df_1 = pd.read_excel(wd + "/df_compases_corpus.xlsx")
df_1.info()

VAR_BERNOULLI = 'Alter'
EXITO = "alt"
CLASES_CANT_ALTER = range(16)

prob_1 =  distrib_prob(df_1, VAR_BERNOULLI , EXITO, CLASES_CANT_ALTER)
print(prob_1)


#-----------------------------------------------------------
# Corre la función en compases de GENERADOS POR TEMPERATURA
#-----------------------------------------------------------

wd  =  "C:/Users/Carlos/Desktop/Doctorado  UNLZ/Tesis  -  CODIGO  y
REFERENCIA/000 CODIGO DEFINITIVO/2022-10-08 ULTIMO/NUEVO - Procesado
en Funciones/DF_COMPASES"

df_JS_cant_alteraciones  =  pd.DataFrame(columns  =  ('Temp',  'JS_pq',
'Dist_JS_pq'))


#for i in (("0.1" , "0.2", "0.3", "0.4", "0.5", "0.6", "0.7", "0.8",
"0.9", "1.0")):


for i in np.arange(0.1 , 1.6, 0.1):
```

```
      df_2   =  pd.read_excel(wd  +  "/df_compases_generados_TEMP_"  +
str(i)(:3) + ".xlsx")
    df_2.info()

    VAR_BERNOULLI = 'Alter'
    EXITO = "alt"
    CLASES_CANT_ALTER = range(16)

        prob_2   =    distrib_prob(df_2,   VAR_BERNOULLI    ,   EXITO,
CLASES_CANT_ALTER)
    print(prob_2)


    #=============================================
    #PRUEBA DE JENSEN-SHANNON DIVERGENCE
    #calculate the kl divergence
    #=============================================

    JS_pq , distance_JS_pq = calcula_JS(prob_1, prob_2)

    df_JS_cant_alteraciones = df_JS_cant_alteraciones.append({'Temp':
i,'JS_pq': JS_pq , 'Dist_JS_pq':distance_JS_pq} , ignore_index = True)
```