



**INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO**

Trabajo Terminal

**Generador de versos musicales en el idioma
inglés por medio de procesamiento de
lenguaje natural y redes neuronales**

TT2020-B002

Presentan:

Espinosa de los Monteros Lechuga Jaime Daniel
Nava Romo Edgar Adrián
Salgado Gómez Alfredo Emilio

Directores:

Olga Kolesnikova
Ariel López Rojas

Firmas de Directores

Firmado por:

Profesor: Ariel López Rojas

Doctora Olga Kolesnikova

Índice.

Índice	2
I Trabajo Terminal I	10
1. Introducción	11
1.1. Objetivos	14
1.1.1. Objetivo general	14
1.1.2. Objetivos particulares	14
1.2. Justificación	14
1.3. Metodología	16
1.3.1. Implementación	18
1.3.2. Iteraciones	19
1.4. Organización del documento	20
2. Marco teórico	21
2.1. Inteligencia artificial	21
2.2. Redes neuronales	22
2.2.1. Neuronas	22
2.2.2. Aprendizaje de las redes neuronales	24
2.2.3. Recurrent Neural Network (RNN) – Long Short Term Memory (LSTM)	25
2.3. Procesamiento de lenguaje natural	27
2.3.1. Tareas del procesamiento de lenguaje natural	27
2.3.2. Usos del procesamiento de lenguaje natural	28
2.4. Transformers	28
2.4.1. Arquitectura de un transformer	29
2.5. BERT	30
2.5.1. ¿Qué es BERT?	30
2.5.2. ¿Cómo funciona BERT?	31
2.5.3. BERT en la actualidad	32

2.6.	GPT-2	32
2.6.1.	¿Qué es GPT-2?	32
2.6.2.	Arquitectura GPT-2	33
2.7.	BERT vs GPT-2	34
2.8.	Base de datos	35
2.8.1.	Base de datos NoSQL o no relacionales	35
2.9.	Género musical	36
2.9.1.	Género Pop	36
2.10.	Canción	36
2.10.1.	Elementos de una canción	36
2.10.2.	Estructura de una canción	38
2.11.	Cadenas de Markov	40
2.12.	Apache	41
2.13.	Servidor web	42
2.14.	Certificado SSL	43
2.15.	Plataformas en la nube para generar modelos de aprendizaje automático	44
2.15.1.	Amazon Web Services	44
3.	Análisis	48
3.1.	Problemática	48
3.2.	Nuestra solución	48
3.2.1.	¿Por qué utilizar inteligencia artificial?	49
3.2.2.	¿Por qué utilizar procesamiento de lenguaje natural?	49
3.2.3.	¿Por qué utilizar redes neuronales Long Short Term Memory (LSTM)?	50
3.3.	Herramientas a utilizar	50
3.3.1.	Software	50
3.3.2.	Hardware	56
3.4.	Algoritmos analizados	58
3.4.1.	Cadenas de Markov	58
3.4.2.	Recurrent Neural Network	58
3.4.3.	BERT	59
3.5.	Pruebas	59
3.5.1.	Pruebas con cadenas de Markov	59
3.5.2.	Pruebas con recurrent neural networks	62
3.5.3.	Pruebas con BERT	63
3.6.	Estudio de factibilidad	63
3.6.1.	Factibilidad técnica	64
3.6.2.	Factibilidad operativa	66
3.6.3.	Factibilidad económica	67

3.6.4. Aspectos económicos	68
3.6.5. Aspectos legales	69
3.7. Análisis de riesgo	70
4. Diseño	75
4.1. Arquitectura del sistema	75
4.1.1. Descripción de la arquitectura del sistema	75
4.1.2. Funcionamiento general del sistema	76
4.1.3. Diagrama caso de uso	77
4.1.4. Historias de usuario	79
4.1.5. Requerimientos funcionales y no funcionales	81
4.2. Base de datos	83
4.2.1. Descripción	83
4.2.2. Obtención de la base de datos	83
4.2.3. Género musical	85
4.2.4. Resultados	86
4.3. Limpieza de la base de datos	87
4.3.1. Descripción	87
4.3.2. Remover caracteres innecesarios	88
4.3.3. Eliminar duplicados	88
4.3.4. Evitar errores ortográficos o de similitud	88
4.3.5. Convertir los tipos de dato	88
4.3.6. Tratar los valores nulos o faltantes	88
4.3.7. Resultados	89
4.4. Redes neuronales a utilizar	90
4.4.1. Descripción	90
4.4.2. Redes neuronales para generar el modelo	90
4.4.3. Modelo	90
4.4.4. Resultados	91
4.5. Generación del modelo en la nube	94
4.5.1. Descripción	94
4.5.2. Amazon Sagemaker	94
4.5.3. Suministrado de datos	96
4.5.4. Resultados	97
4.6. Aplicación web	99
5. Desarrollo	102
5.1. Desarrollo del modelo	102
5.1.1. Amazon AWS	102
5.1.2. Google Cloud Platform	102
5.1.3. Kaggle	104

5.2.	Creación del modelo	107
5.2.1.	Embedding o incrustación	107
5.2.2.	Bidireccional	107
5.2.3.	Dropout	107
5.2.4.	Densidad	108
5.2.5.	Método de compilación, algoritmo de optimización y métrica de rendimiento	108
5.3.	Desarrollo del back-end	110
5.4.	Desarrollo de la página web	116
5.5.	Desplegando componentes	123
5.5.1.	Despliegue del back-end	123
5.5.2.	Despliegue de la página web	123
5.6.	Pruebas	126
6.	Conclusiones	127
7.	Trabajo a futuro	128
7.1.	Generar rimas en los textos	128
7.2.	Generar textos a partir de otros géneros	130
A	Anexos	132
.1.	Cadenas de Markov	133
.2.	Pruebas utilizando long short term memories	136

Índice de figuras.

1.1.	Funcionamiento de la planificación por cubetas	18
2.1.	Estructura de una neurona [18]	23
2.2.	Neurona artificial	23
2.3.	Diagrama de una red neuronal recurrente	26
2.4.	Arquitectura general de un transformer [25]	29
2.5.	Representación de entrada de BERT y la separación de oración por palabras y la asignación de valores.[26]	31
2.6.	Arquitectura del modelo de GPT-2 [31]	33
2.7.	Ejemplo análisis de una oración por GPT-2 [33]	34
2.8.	Diagrama de la estructura para la generación de las letras musicales	39
2.9.	Diagrama de estados finitos de una cadena de Markov [37] . .	41
3.1.	Texto generado utilizando LSTM y la oración en inglés "Love flowers"	62
3.2.	Texto generado utilizando LSTM y la oración en inglés "It's a cruel and random world"	62
3.3.	Error tensorflow BERT	63
3.4.	Matriz de riesgo de costo	71
3.5.	Matriz de riesgo de calendario	71
3.6.	Matriz de riesgo tecnológico	72
3.7.	Matriz de riesgo operacional	73
3.8.	Matriz de riesgos externos	74
3.9.	Matriz general de riesgos	74
4.1.	Diagrama general del sistema	76
4.2.	Diagrama de caso de uso	77
4.3.	Diagrama de obtención de dataset	83
4.4.	Diagrama entidad-relación de la base de datos	84
4.5.	Diagrama entidad-relación de la base de datos procesada . .	85
4.6.	Diagrama entidad-relación final	86

4.7.	Resultados de limpieza en la base de batos	89
4.8.	Diagrama de diseño del modelo	91
4.9.	Diagrama de diseño final del modelo	92
4.10.	Diseño de la estructura del modelo	93
4.11.	Diagrama de instancias de AWS	94
4.12.	Instancia creada en AWS	96
4.13.	Diagrama del funcionamiento de SageMaker	97
4.14.	Diagrama simple del proceso seguido para generar el modelo en la nube	98
4.15.	Diagrama de conexión entre modelo y usuario	99
4.16.	Página inicial que se mostrará al usuario	100
4.17.	Aplicación web, página una vez generada la letra musical.	101
5.1.	Máquina creada para el desarrollo del modelo.	103
5.2.	Libreta y entrenamiento del modelo.	103
5.3.	Cubeta donde se encuentra la base de datos que contiene las letras de las canciones.	104
5.4.	Estadísticas de las palabras	105
5.5.	Tokenizado de las palabras [86]	106
5.6.	Padding del tokenizado de las palabras	106
5.7.	Información sobre el estado del Back-end	112
5.8.	Cuadro del método Post	113
5.9.	Páginas trabajadas	117
5.10.	Página de bienvenida	118
5.11.	Formulario	118
5.12.	Leyenda mostrada	120
5.13.	Texto generado	120
5.14.	Archivo de texto descargado	122
5.15.	Instalando CLI de Vercel	123
5.16.	Indicando acciones para el despliegue	124
5.17.	Detección del tipo de proyecto a desplegar	124
5.18.	Desplegando el proyecto	125
5.19.	Aplicación web desplegada	125
5.20.	Accediendo a la aplicación web desplegada	126
7.1.	Estructura primer verso	129
7.2.	Estructura de la letra de la canción con rimas	130

Índice de cuadros.

1.1. Productos similares	13
1.2. Comparación de metodologías ágiles	16
2.1. Comparación entre tecnologías BERT y GPT-2	35
2.2. Comparación de servidores web	42
2.3. Tabla comparativa de las diversas plataformas contempladas .	45
2.4. Tabla comparativa GCP vs AWS	47
3.1. Equipo de cómputo 1	56
3.2. Equipo de cómputo 2	57
3.3. Equipo de cómputo 3	57
3.4. Equipo de cómputo ideal	64
3.5. Herramientas de software a utilizar	64
3.6. Equipo de cómputo 1	65
3.7. Equipo de cómputo 2	65
3.8. Equipo de cómputo 3	65
3.9. Horas de trabajo	66
3.10. Riesgo de costo	70
3.11. Riesgo de calendario	71
3.12. Riesgo tecnológico	72
3.13. Riesgo operacional	73
3.14. Riesgos externos	73
4.1. Caso de uso 1	78
4.2. Caso de uso 2	78
4.3. Caso de uso 3	78
4.4. Caso de uso 4	78
4.5. Historia de usuario 1	79
4.6. Historia de usuario 2	79
4.7. Historia de usuario 3	79
4.8. Historia de usuario 4	79
4.9. Historia de usuario 5	80

4.10. Historia de usuario 6	80
4.11. Historia de usuario 7	80
4.12. Historia de usuario 8	80
4.13. Historia de usuario 9	80
4.14. Historia de usuario 10	81
4.15. Historia de usuario 11	81
4.16. Historia de usuario 12	81

Parte I

Trabajo Terminal I

Capítulo 1

Introducción

La industria musical obtiene ganancias a través de la creación y divulgación de la música física y digitalmente (Bourreau and Gensollen 2006 [5]), dejando que aficionados y emprendedores de la música no tengan oportunidad de avanzar en su carrera por falta de creatividad, tiempo y/o recursos, haciendo que la creación de nuevas letras para sus canciones sea un gran obstáculo, nuestra propuesta implica la utilización de nuevas tecnologías que permitan la generación de letras para integrar con sus canciones. Esta es una de las tareas más populares y desafiantes en el área de procesamiento del lenguaje natural. Hay una gran cantidad de trabajos (Generating Text with Recurrent Neural Networks [6], Convolutional Neural Networks for Sentence Classification[7]) que proponen generar texto utilizando redes neuronales recurrentes y/o convolucionales. Sin embargo, la mayoría de los trabajos actuales solo se enfocan en generar una o varias oraciones, ni siquiera un párrafo largo y mucho menos una canción completa.

Las letras de canciones, como un tipo de texto, tienen algunas características propias, estas se constituyen de rimas, versos, coros y en algunos casos, patrones de repetición. Coro se refiere a la parte de una canción que se repite sin modificaciones dentro de la misma después de un verso, mientras que en el verso suelen cambiar una o varias líneas que lo componen. Estas características particulares hacen que generar letras musicales sea mucho más difícil que textos normales.

La mayoría de las investigaciones actuales sobre generación de letras vienen con muchas condiciones, como dar una pieza de melodía (Automatic Generation of Melodic Accompaniments for Lyrics [8]), o solo generar un tipo específico de letra (Conditional Rap Lyrics Generation with Denoising Autoencoders [9]). Sin embargo, la generación de letras por medio de in-

teligencia artificial dado un estilo y tema en particular, ha sido muy poco trabajado y debido a esto planeamos centrarnos en este nuevo problema. Estamos interesados en ver si el modelo propuesto puede aprender diferentes características de un género musical y generar letras que sean acorde a este. Actualmente, en el mercado se encuentran cuatro aplicaciones web que tienen una funcionalidad similar a la propuesta en este Trabajo Terminal:

- These lyrics do not exist.
- Bored humans - lyrics_generator.
- DeepBeat.
- Premium Lyrics.

En el cuadro 1 que se presenta a continuación, se muestran las características de aplicaciones web similares y comparándolas con nuestra propuesta:

Cuadro 1.1: Resumen de productos similares comparados con nuestra propuesta

Software	Características	Precio
These Lyrics do not Exist	Aplicación web que genera letras completamente originales de varios temas, hace uso de IA para generar coros y versos originales; se puede escoger el tema principal de la letra, género musical e incluso el estado de ánimo al que iría dirigido.	Gratis (Contiene Anuncios)
Boredhumans lyrics-generator	Aplicación web en el que la IA fue entrenada con una base de datos con miles de textos para generar canciones totalmente nuevas. La letra que crea es única y no una copia de alguna que exista actualmente, sin embargo, no permite modificar la letra.	Gratis
DeepBeat	Aplicación web que por medio de IA genera letras de música enfocada principalmente en el género rap. Si una línea no es del agrado se puede sustituir por alguna de las otras propuestas de las que ofrece.	Gratis
Premium Lyrics	Aplicación web que proporciona versos compuestos en distintos idiomas por artistas independientes que se escogen manualmente de acuerdo a su originalidad y calidad.	\$3 a \$75 Dólares por letra musical
Nuestra propuesta	Aplicación web que haciendo uso de una IA va a generar letras musicales originales a partir de un género musical en exclusivo, lo que asegurará un resultado original con coros y versos distintos cada vez que se utilice.	Gratis

1.1. Objetivos

1.1.1. Objetivo general

Crear una herramienta de apoyo para estudiantes o aficionados interesados en este rubro que se les dificulte componer nuevas letras musicales de un solo género musical debido a la carencia de creatividad, la falta de conocimientos en la estructura del género o que no tengan inspiración suficiente para poder crear nuevas canciones, esto con el fin de impulsar la carrera de futuros artistas en la industria musical que no tengan los suficientes recursos para poder contratar servicios particulares de compositores.

1.1.2. Objetivos particulares

- Generar un conjunto de datos (dataset) con letras musicales en un género musical para efecto de entrenamiento en la red semántica.
- Hacer uso de alguna herramienta que utilice aprendizaje automático (machine learning) e implementar su uso en la nube para ayudar a procesar las letras musicales en un género específico.
- Implementar un módulo analizador de semántica para entrenar redes neuronales.
- Desarrollar una interfaz web intuitiva en versión prototipo que utilice una aplicación web alojada en la nube para tener una visualización del verso musical generado a partir de un género.

1.2. Justificación

El crear nuevas composiciones musicales puede llegar a ser muy difícil, estresante e incluso agotador para cualquier aficionado o incluso algunos expertos en este medio, esto se debe a la falta de creatividad y/o tiempo de quien lo quiera realizar [10]. En ocasiones se pueden contratar servicios particulares para la producción de letras musicales, sin embargo, puede ser muy costoso y en ocasiones el resultado final no alcanza a llenar las expectativas de la inversión que se hace; por ende, se pretende crear una herramienta para estudiantes, aficionados o cualquier persona interesada en este rubro que se les dificulte componer nuevas letras musicales.

Normalmente estas son creadas por el humano y tienden a estar compuestas por patrones de acuerdo al género musical [11]. Algunos ejemplos de estos

patrones pueden ser las rimas, enunciados, frases cortas y que tengan una semántica correcta, estos pueden ser encontrados utilizando algoritmos de inteligencia artificial, específicamente, procesamiento de lenguaje natural y una investigación profunda en la composición de letras en estos géneros.

Se eligió el idioma inglés debido a que existe una gran cantidad de datasets para procesar, al igual que herramientas y documentación para este idioma.

Nos proponemos orientar esta solución en un entorno utilizando la nube, donde la información de configuración, servicios y datos necesarios pueden mantenerse de forma independiente a la implementación, facilitando la adaptación y flexibilidad de la plataforma.

Nuestro proyecto ayudará al usuario utilizando herramientas como el procesamiento de lenguaje natural, redes neuronales, aprendizaje de máquina (machine learning) y servidores en la nube. Se hará uso de un conjunto de datos (dataset) y herramientas alojadas en la nube (Google Cloud Platform o Amazon Web Services) para procesar estos datos; se pretende utilizar un módulo que encuentre patrones por medio de redes neuronales para analizar la semántica mediante técnicas y herramientas ya existentes de procesamiento de lenguaje natural. Se van a realizar pruebas y experimentos con estas herramientas antes de la implementación (BERT [12], spaCy[13]) para poder generar versos. A su vez se va a desarrollar una interfaz web intuitiva en versión prototipo donde el usuario va a poder utilizar esta herramienta la cual le va a mostrar la letra musical que se va a generar en ese momento.

A diferencia de los proyectos señalados en la Tabla 1.1 nuestra propuesta se va a centrar en generar letras musicales con métodos y tecnologías distintas a los que se usaron, esto es, aunque se utilicen los mismos géneros musicales se tendrán resultados completamente diferentes con propuestas distintas.

En el desarrollo de este proyecto haremos uso de los conocimientos adquiridos durante el transcurso de la carrera. Se van a utilizar técnicas de diseño de proyectos aprendidas en el curso de Ingeniería de Software, se van a aplicar los conocimientos de programación adquiridos en unidades de aprendizaje como Inteligencia Artificial, Procesamiento de Lenguaje Natural, Web Application Development, Programación Orientada a Objetos, Análisis de Algoritmos, así como técnicas de construcción de documentos y análisis de semántica vistas en Análisis y Diseño orientado a Objetos y Comunicación Oral y Escrita.

1.3. Metodología

Para el desarrollo de este trabajo terminal se utilizará la metodología ágil Scrumban, que combina algunas partes de la metodología Scrum y Kanban, debido a que este proceso de gestión reduce la complejidad al momento de desarrollar un producto el cual tratará de satisfacer las necesidades de los clientes. Además, permite trabajar colaborando entre pares de manera eficiente, es decir, en equipo, para obtener el mejor resultado posible.

Scrumban combina la estructura utilizada por Scrum, con los métodos basados en el flujo y visualización de Kanban. Es decir, que permite a los equipos trabajar de manera ágil usando la metodología Scrum y la simplicidad de Kanban sin tener que utilizar las actualizaciones de roles y es más sencillo de adoptar.

En la siguiente tabla se pueden observar las principales diferencias entre las tres metodologías:

Cuadro 1.2: Tabla comparativa de las distintas metodologías ágiles contempladas

	Scrum	Kanban	Scrumban
Procesos	Iterativo e incremental desarrollando sprints	Continuo	Iterativo e incremental de forma continua desarrollando iteraciones
Personas	Las personas son el centro	Las personas son el pilar	Equipo motivado con personas como pilar y en el centro
Producto	Foco en la efectividad	Foco en la eficiencia	Balance entre efectividad y eficiencia
Organización	Mejora continua del producto	Mejora continua del proceso	Mejora continua del producto y del proceso
Equipo	De 3 a 9 personas	No hay limitaciones	El equipo no requiere de un número específico de integrantes
Roles	Scrum master, Product owner, Scrum team	Pueden incluir especialistas o integrantes generalizados	No requiere un rol específico

Scrumban hace uso de iteraciones, las cuales monitorea con el apoyo de un recurso visual, como lo puede ser un tablero. Las reuniones para planificar se llevan a cabo cuando son necesarias para determinar las tareas a implementar hasta la próxima iteración. Para que estas iteraciones se mantengan cortas, se utiliza un límite de trabajo en progreso (WIP por sus siglas en inglés Work in Progress). Cuando WIP desciende de cierto nivel, se establece una acción para que el equipo sepa cuándo y qué tarea planificar a continuación.

Iteración

En Scrumban, las iteraciones son cortas para garantizar que el equipo pueda adaptarse al entorno cambiante durante el proyecto. La duración de estas iteraciones en este proyecto se medirán como máximo en lapsos de dos semanas.

Priorización

Esta se da de tal forma que las tareas más importantes se colocan en la parte superior en la tabla de planificación seguidas por las tareas menos importantes.

Antes de llegar al tablero las tareas deben pasar por 3 etapas donde se van depurando las que se van a realizar para largo plazo (1 año), medio plazo (6 meses) y corto plazo (3 meses) siendo esta última de donde salen las tareas más claras que se pueden completar y que ganan mayor prioridad para la próxima iteración.

Principio de Elección

Cada miembro del equipo elige una o varias tarea de la sección “Tareas Pendientes” que va a completar antes de la siguiente reunión.

Congelación de Funciones

Se utiliza cuando hay una fecha límite del proyecto próxima, significando que sólo pueden trabajar sobre las tareas previamente pensadas sin cabida para implementar nuevas características.

Triage

Ocurre después de la congelación de funciones, es aquí donde el líder del proyecto decide las características en desarrollo que se completarán y cuáles quedarán sin terminar.

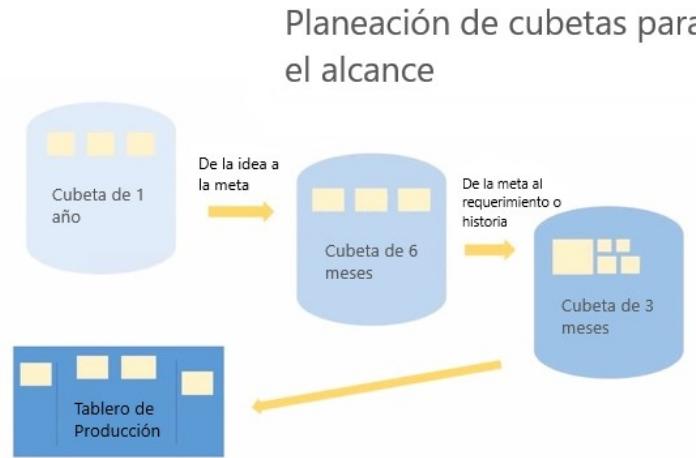


Figura 1.1: Funcionamiento de la planificación por cubetas

1.3.1. Implementación

En el proyecto se aplicará esta metodología utilizando la aplicación de Trello para llevar a cabo el tablero, contando con cuatro listas importantes:

- Scrum Backlog: Contará con las características generales de la iteración en la que se pretende trabajar.
- To Do: Lista en la que cada integrante del equipo agrega funcionalidades tanto las que son funcionales como las que no lo son que tendrá el proyecto, en sus debidas reuniones se discutirá sobre la priorización de las tarjetas que se agreguen y cada integrante elegirá las que quiera desarrollar y cuántas pueda hacer.
- In Progress: Cuando el integrante del equipo ya está trabajando en alguna de las tareas que eligió, pasa a formar parte de esta lista, esto quiere decir que en la siguiente reunión que se tenga tendrá que dar sus debidos resultados o en su defecto, comentar que ya se terminó.
- Done: Las tareas que forman parte de la iteración que ya se terminaron, van pasando a esta lista, dejando a las otras listas con menos tarjetas

a realizar cada vez. Cuando ya estén todas las tareas realizadas, se puede volver a Backlog para poder proseguir con la siguiente iteración del proyecto, en caso de que queden tarjetas y las fechas límites estén próximas a terminar, se tendrán que parar las tareas que no sean vitales al proyecto para que se puedan realizar las tareas con más importancia.

1.3.2. Iteraciones

El proyecto se dividirá en cuatro iteraciones por trabajo terminal, cada iteración tiene una duración establecida de cuatro semanas o un mes, dejando cada iteración de la siguiente manera:

Trabajo Terminal I

- Se construirá una base de datos acorde al proyecto, la cual contendrá las letras musicales de distintos artistas, así como su género.
- Se realizará una limpieza a los datos extraídos del dataset.
- Se seleccionarán e implementarán las herramientas necesarias para trabajar en la construcción de un modelo utilizando redes neuronales.
- Se generará un modelo en la nube para su entrenamiento utilizando las redes neuronales previamente elegidas.

Trabajo Terminal II

- Se implementará la interfaz que el usuario final verá.
- Se integrará el la interfaz con la aplicación que utiliza el modelo utilizando un servidor web.
- Se realizarán las pruebas pertinentes, así como la re-ingeniería requerida.
- Se desarrollará el manual técnico y el de usuario.

1.4. Organización del documento

Para dar inicio a este trabajo terminal, presentamos de manera breve la estructura de este reporte, con el objetivo de que el lector pueda tener un mejor entendimiento del trabajo.

Capítulo 2. Marco teórico

En esta parte del documento, se describen puntos esenciales de nuestro trabajo como definiciones, técnicas, herramientas, servicios, así como investigación realizada para llevar a cabo en la implementación dentro del trabajo.

Capítulo 3. Análisis

Dentro de este capítulo se hará un análisis tanto del problema propuesto, como sus posibles soluciones, se harán análisis de estudio en cuanto a la factibilidad técnica, operativa y económica con la finalidad de conocer los recursos necesarios para la elaboración de este trabajo terminal. Se mencionan las herramientas a utilizar y se explica de manera general la arquitectura del sistema y las historias de usuario del mismo.

Capítulo 4. Diseño

En el cuarto capítulo, nos adentraremos en el desarrollo de las iteraciones propuestas, es decir, se encuentran los diagramas pertinentes para poder modelar nuestro trabajo terminal y proceder a la etapa de implementación. En este capítulo se desarrolla el diseño de cada parte del sistema y se muestra la interfaz de usuario propuesta junto con los requisitos de diseño.

Capítulo 2

Marco teórico

2.1. Inteligencia artificial

Por Inteligencia Artificial se entiende a una simulación de procesos computacionales cognitivos para que simulen el comportamiento de una mente humana, estos comportamientos abarcan diferentes áreas de investigación, como el razonamiento, aprendizaje, percepción, comunicación y la capacidad de desarrollarse en entornos más complejos. “La inteligencia no es una dimensión única, sino un espacio profusamente estructurado de capacidades diversas para procesar la información. Del mismo modo, la Inteligencia Artificial utiliza muchas técnicas diferentes para resolver una gran variedad de tareas que se encuentran en todas partes.” [14]

En ciencias de la computación, el término de inteligencia artificial hace referencia a cualquier inteligencia semejante a la humana presentado por una computadora o un equipo electrónico. De manera popular, la inteligencia artificial hace referencia a la capacidad de una computadora o máquina de imitar las facultades del cerebro humano, es decir, que es capaz de aprender mediante ejemplos y experiencias, así como reconocer objetos y clasificarlos, tomar decisiones, resolver problemas, entender y responder al lenguaje, igualmente combinar estas y otras capacidades para realizar funciones similares que un ser humano podría realizar. [15]

Como objetivo, la Inteligencia Artificial pretende hacer un uso de los recursos tecnológicos para desarrollar modelos computacionales y poder resolver problemas del mundo real evolucionando constantemente.

2.2. Redes neuronales

Una red neuronal es una herramienta derivada de la inteligencia artificial que utiliza modelos matemáticos para ser utilizada como un mecanismo de predicción del texto.

Las redes neuronales son una forma de hacer que las computadoras aprendan, donde la computadora aprende a realizar alguna tarea analizando ejemplos de entrenamiento. Por lo general, estos ejemplos han sido etiquetados previamente.

Modelado vagamente en el cerebro humano, una red neuronal consiste en miles de millones de nodos que realizan procesamiento los cuales están densamente interconectados. En la actualidad las redes neuronales están organizadas como capas de nodos, y estas capas están “alimentadas hacia adelante” (feed-forward), es decir, la información que se mueve a través de ellas solo fluye en una sola dirección. Un solo nodo puede estar conectado a muchos nodos en capas inferiores de las cuales recibe información y a su vez, puede estar conectado a nodos en capas superiores a los cuales envía información.

Cuando una red neuronal está siendo entrenada la información de entrenamiento pasa a alimentar las capas inferiores (de entrada), donde será procesada y pasada a capas posteriores donde será transformada hasta llegar a las capas superiores (de salida).[16]

2.2.1. Neuronas

Neurona natural

La neurona es una de muchas células la cual representa a la unidad estructural y funcional del sistema nervioso. Esta trabaja transmitiendo la información a través de impulsos nerviosos o químicos, desde un lugar del cuerpo hacia otra parte de este.

Dicho impulso nervioso o eléctrico viaja siempre en un mismo sentido, es decir, llega a la neurona mediante las dendritas, se procesa en el soma y posteriormente se transmite al axón, las neuronas no están en contacto entre sí, existe un espacio de separación denominado sinapsis o espacio sináptico. Una vez que el impulso nervioso llega al extremo final del axón este libera neurotransmisores al espacio sináptico transformando esta señal eléctrica en una señal química que en su funcionamiento, se introduce en la dendrita

ubicada en la neurona contigua, desencadenando un impulso eléctrico en la receptora, repitiendo este proceso n veces hasta llegar a su destino final.[17]

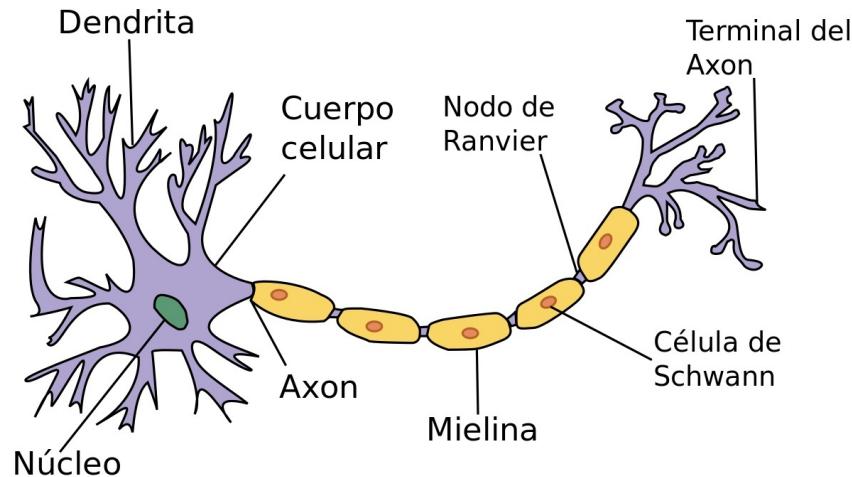


Figura 2.1: Estructura de una neurona [18]

Neurona artificial

La complejidad de las neuronas reales se abstrae mucho cuando se realiza el modelado de neuronas artificiales. Estas consisten básicamente en entradas, que se multiplican por pesos (fuerza de las respectivas señales), y luego calculada por una función matemática que determina la activación de la neurona. Otra función (que puede ser la de identidad) calcula la salida de la neurona artificial (a veces en dependencia de un cierto umbral). Las redes neuronales combinan neuronas artificiales para procesar información.[19]

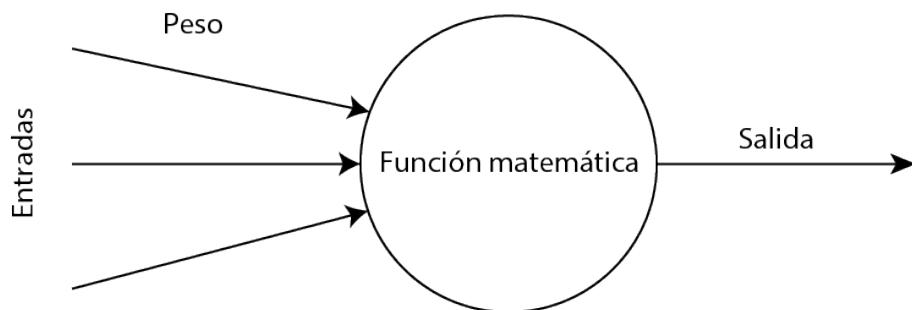


Figura 2.2: Neurona artificial

2.2.2. Aprendizaje de las redes neuronales

Si observamos la naturaleza, podremos ver que a pesar de existir muchos tipos de sistemas, los que pueden aprender son altamente adaptables. En su búsqueda por adquirir conocimientos, estos sistemas utilizan información del mundo exterior y modifican la información que ya han recopilado o modifican su estructura interna. Eso es exactamente lo que hacen las redes neuronales. Adaptan y modifican su arquitectura logrando aprender. Para ser más precisos, las redes neuronales cambian los pesos de las conexiones según la entrada y salida deseada.

¿Por qué tienen un peso?, bueno, si observamos la estructura de las redes neuronales, hay algunos componentes que podríamos cambiar, si queremos modificar su arquitectura. Por ejemplo, podríamos crear nuevas conexiones entre neuronas, eliminarlas y/o agregar más de estas. Incluso se podría modificar la función de entrada o la de activación. Resulta que cambiar los pesos es el enfoque más práctico. Además, la mayoría de los otros casos podrían cubrirse cambiando los pesos. La eliminación de una conexión, por ejemplo, se puede hacer estableciendo el peso en 0. Y una neurona se puede eliminar si establecemos los pesos de todas sus conexiones en cero.[20]

El entrenamiento es un proceso necesario para toda red neuronal, y consiste en que la red se familiariza con el problema que necesita resolver. En la práctica, generalmente se tienen algunos datos recopilados en función de los cuales necesitamos crear nuestras predicciones, clasificación, o cualquier otro procesamiento. Estos datos se denominan conjunto de entrenamiento y según el comportamiento y la naturaleza, tenemos algunos tipos de aprendizajes:

- **Aprendizaje no supervisado:** el conjunto de entrenamiento solo contiene entradas. La red intenta identificar entradas similares y clasificarlas en ciertas categorías.
- **Aprendizaje reforzado:** el conjunto de entrenamiento contiene entradas, pero la red también recibe información adicional durante la formación. Lo que sucede es que cuando la red calcula cada salida para una de las entradas, proporcionamos información que indica si el resultado fue correcto o incorrecto y, posiblemente, la naturaleza del error que cometió la red.
- **Aprendizaje supervisado:** el conjunto de entrenamiento contiene entradas y salidas deseadas. De esta manera, la red puede verificar su

salida calculada con la salida deseada y tomar las acciones pertinentes para reformular su cálculo.

2.2.3. Recurrent Neural Network (RNN) – Long Short Term Memory (LSTM)

Una red neuronal recurrente es un tipo de red neuronal artificial donde la salida de alguna capa en particular es salvada y sirve para retroalimentar la entrada de esta capa, lo cual ayuda a predecir futuras salidas de esta.

La primera capa esta formada de la misma manera que la Feed-forward Neural Network, es decir, solo pasa la información que entra a la siguiente capa inmediata, posteriormente la siguiente capa con el paso del tiempo comenzará a retroalimentarse, pero manteniendo la propagación frontal. Haciendo uso de esta retroalimentación la capa en futuras operaciones puede realizar predicciones, si estas predicciones no son los resultados esperados, el sistema aprende y trabaja para corregir sus futuras predicciones.[21]

Se distinguen por su “memoria”, ya que toman información de entradas anteriores para influir en la entrada y salida actuales. Mientras que las redes neuronales profundas tradicionales asumen que las entradas y salidas son independientes entre sí, la salida de las redes neuronales recurrentes depende de los elementos anteriores dentro de la secuencia. Si bien los eventos futuros también serían útiles para determinar la salida de una secuencia dada, las redes neuronales recurrentes unidireccionales no pueden tener en cuenta estos eventos en sus predicciones.[22]

Estos algoritmos de aprendizaje profundo se utilizan comúnmente para problemas relacionados con la traducción de idiomas, el procesamiento del lenguaje natural, el reconocimiento de voz y los subtítulos de imágenes.

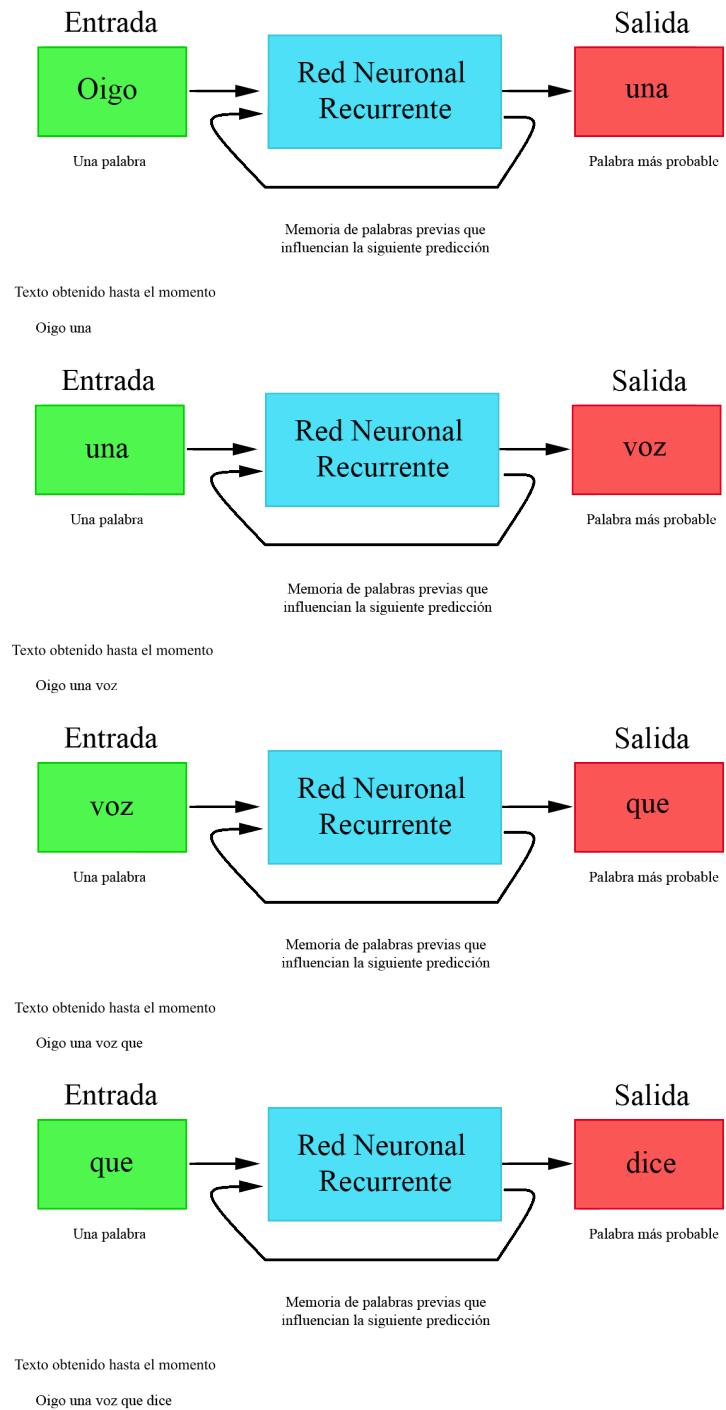


Figura 2.3: Diagrama de una red neuronal recurrente

2.3. Procesamiento de lenguaje natural

El Procesamiento del Lenguaje Natural (PLN) se refiere a una rama de la informática, específicamente, a una rama de la inteligencia artificial, la cual se ocupa de dar a las computadoras la capacidad de comprender texto y palabras de la misma manera que los seres humanos.

El PLN combina la lingüística computacional con modelos estadísticos, machine y deep learning. Juntas, estas tecnologías permiten a las computadoras procesar el lenguaje humano en forma de texto o datos de voz y ‘comprender’ su significado completo, es decir, la intención y el sentimiento del hablante o escritor. [23]

2.3.1. Tareas del procesamiento de lenguaje natural

- **Reconocimiento de voz:** Su tarea pertinente es la conversión de voz a texto, requiere convertir de manera confiable datos de voz en datos de texto. Lo cual hace al reconocimiento de voz una tarea especialmente desafiante es la forma en que las personas hablan: velocidad, arrastrando las palabras, con diferentes énfasis, entonación y acentos.
- **Etiquetado de palabras:** Es el proceso de determinar la parte gramatical de una palabra o fragmento de texto en particular, en función de su uso y contexto.
- **Desambiguación de las palabras:** Es la selección del significado de una palabra con múltiples significados a través de un proceso de análisis semántico que determina la palabra que tiene más sentido en el contexto dado.
- **Reconocimiento de entidad nombrada:** Identifica palabras o frases como entidades útiles, es decir, identifica nombres propios.
- **Resolución de correferencia:** Es la tarea de identificar si dos palabras se refieren a la misma entidad. El ejemplo más común es determinar la persona u objeto al que se refiere un determinado pronombre.
- **Análisis de sentimientos:** Intenta extraer del texto cualidades subjetivas (actitudes, emociones, sarcasmo, confusión, sospecha).
- **Generación de lenguaje natural:** Es la tarea de convertir información estructurada a lenguaje humano.

2.3.2. Usos del procesamiento de lenguaje natural

El procesamiento del lenguaje natural es el cerebro detrás de la inteligencia artificial en muchas aplicaciones del mundo real. Algunos ejemplos son los siguientes:

- **Aplicaciones dedicadas a la traducción:** un ejemplo de estos es Google Translate, el cual hace uso de BERT para el procesamiento del lenguaje natural así como para el etiquetado de las palabras que requiere traducir.
- **Asistentes virtuales y chatbots:** Existen hoy en día diferentes tecnologías que simulan ser asistentes virtuales como Siri o Alexa requieren de un reconocimiento de voz para distinguir comandos, así como para responder adecuadamente a estos. En el caso de los chatbots realizan un análisis contextual de las preguntas para proveer respuestas relacionadas a las mismas.
- **Análisis de sentimientos en redes sociales:** Consta de analizar el lenguaje utilizado en publicaciones dentro de las redes sociales, respuestas, reseñas y más para extraer actitudes y emociones en respuesta a productos, promociones y eventos; información que las empresas pueden usar en diseños de productos, campañas publicitarias y más
- **Resúmenes de textos:** Consta de realizar resúmenes y sinopsis de distintos textos.

2.4. Transformers

El Transformer en Procesamiento del Lenguaje Natural (PLN) es una arquitectura que tiene como objetivo codificar cada palabra que compone una frase en función a la secuencia que cada palabra sigue, permitiendo así agregar un contexto o una representación matemática dentro del texto.

Se suele trabajar con los transformadores en dos etapas:

- **Pre-entrenamiento:** En esta etapa, el modelo aprende cómo se estructura el lenguaje de forma general, así como de obtener un conocimiento general del significado de las palabras dentro de la frase trabajada.[24]
- **Afinado:** En esta etapa se añaden ciertas capas o funciones a la arquitectura para adaptar los modelos y que estos realicen ciertas tareas,

para después estos modelos ser re-entrenados y cumplan con las tareas establecidas de manera correcta.[24]

2.4.1. Arquitectura de un transformer

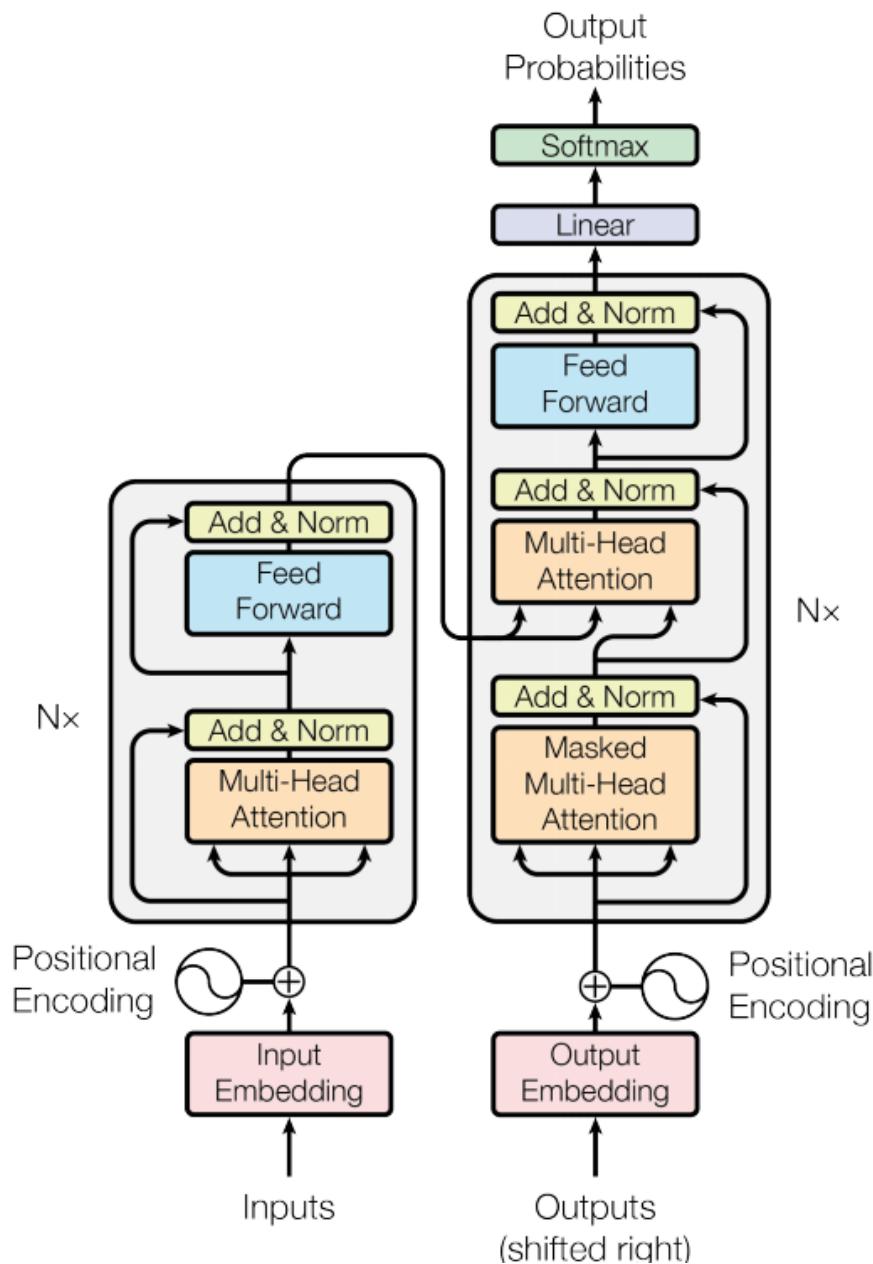


Figura 2.4: Arquitectura general de un transformer [25]

El codificador mapea una secuencia de entrada las cuales son representaciones de símbolos (x_1, \dots, x_n) a una secuencia de representaciones continuas $z = (z_1, \dots, z_n)$. Dado z , el decodificador genera una salida secuencia (y_1, \dots, y_m) de símbolos, un elemento a la vez. En cada paso, el modelo es autorregresivo, consumiendo los símbolos generados previamente como entrada adicional para generar el siguiente.

Los transformers siguen esta arquitectura general usando auto-atención, es decir, las capas están completamente conectadas tanto para el codificador como para el decodificador, que se muestran en las mitades izquierda y derecha de la Figura 2.4 anterior, respectivamente. [25]

2.5. BERT

2.5.1. ¿Qué es BERT?

Bidirectional Encoder Representations from Transformers (BERT) es un marco de aprendizaje automático de código abierto para el Procesamiento del Lenguaje Natural (PLN). BERT, significa representaciones de codificador bidireccional de transformers, el cual es un modelo de aprendizaje profundo en el que cada elemento de salida está conectado a cada uno de entrada y las ponderaciones entre ellos se calculan dinámicamente en función de su conexión. [26]

Al contar con la capacidad bidireccional, BERT está previamente entrenado en dos tareas de PLN diferentes, pero relacionadas: el modelado de lenguaje enmascarado y la predicción de la siguiente oración.

El objetivo del entrenamiento Modelado de Lenguaje Enmascarado (MLM) es ocultar una palabra en cada oración y luego hacer que el programa prediga qué palabra se ha ocultado en función del contexto de la que esta oculta. El objetivo del entrenamiento de predicción de la siguiente oración es que el programa prediga si dos oraciones dadas tienen una conexión lógica secuencial o si su relación es simplemente aleatoria.

2.5.2. ¿Cómo funciona BERT?

El objetivo de cualquier técnica utilizando PLN es comprender el lenguaje humano tal como se habla de forma natural. En el caso de BERT, esto normalmente significa predecir la siguiente palabra. Para hacer esto, los modelos normalmente necesitan entrenarse usando un gran repositorio de datos para usarlos en el entrenamiento, los cuales estarán especializados y etiquetados.

BERT, sin embargo, fue entrenado previamente usando solo un corpus de texto plano sin etiquetar (textos de Wikipedia en inglés). Continúa aprendiendo sin supervisión del texto sin etiquetar y mejorando incluso cuando se usa en aplicaciones prácticas. Su entrenamiento previo sirve como una capa base de “conocimiento”. A partir de ahí, BERT puede adaptarse al creciente cuerpo de contenido y consultas para ajustarse a las especificaciones del usuario. Este proceso se conoce como aprendizaje por transferencia.

BERT es posible gracias a la investigación de Google sobre transformers. El transformer es la parte del modelo que le da a BERT su mayor capacidad para comprender el contexto y la ambigüedad en el lenguaje. Se hace esto procesando cualquier palabra dada en relación con todas las demás palabras en una oración, en lugar de procesarlas una a la vez.

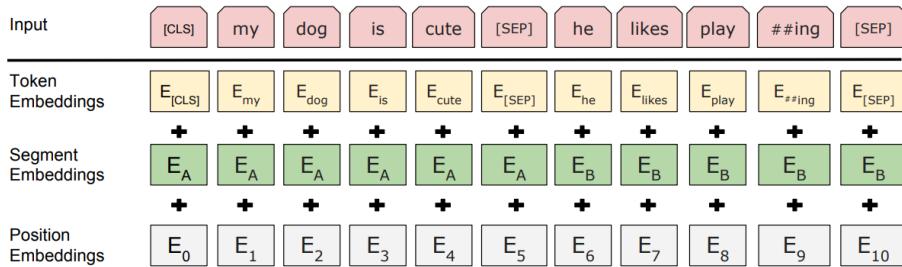


Figura 2.5: Representación de entrada de BERT y la separación de oración por palabras y la asignación de valores.[26]

En la figura anterior, lo que se realiza es que se inserta un token [Classification (CLS)] al principio de la primera oración y un token [Separate (SEP)] al final ella, cada palabra es separada en una ficha, a cada ficha se agrega una inserción que indica si se trata de la oración A o la oración B, finalmente se agrega una incrustación posicional a cada token para indicar su posición en la secuencia. [27]

Esto contrasta con el método tradicional de procesamiento del lenguaje, conocido como incrustación de palabras, en el que los modelos mapeaban cada

palabra en un vector, que representa sólo una dimensión, es decir, el significado de esa palabra.

2.5.3. BERT en la actualidad

BERT se utiliza actualmente en Google para optimizar la interpretación de las búsquedas que realizan los usuarios.

Así como en la realización de tareas tales como: [28]

- Generación de lenguaje basadas en secuencia a secuencia (Respuesta a preguntas, resúmenes de documentos, predicción de siguiente oración, chatbots).
- Comprensión del lenguaje natural (Entendimiento de la polisemia, co-referencia, desambiguación y clasificación de las palabras por sentimientos).

2.6. GPT-2

2.6.1. ¿Qué es GPT-2?

Generative Pretrained Transformer (GPT) es un transformer aprovechado para realizar tanto aprendizaje supervisado como no supervisado para el Procesamiento del Lenguaje Natural (PLN).

GPT-2 es el sucesor de GPT, es un gran modelo de lenguaje basado en transformers con 1500 millones de parámetros, entrenado en un conjunto de datos de 8 millones de páginas web. GPT-2 se entrena con un objetivo simple: predecir la siguiente palabra, dadas todas las palabras anteriores dentro de un texto. [29]

El conjunto de datos no requiere ningún paso de procesamiento previo. En otras palabras, se omiten las mayúsculas y minúsculas, la tokenización y otros pasos, ya que los autores creen que estos pasos de preprocesamiento restringen la capacidad del modelo y así ser capaz de evaluar todos los puntos de referencia del lenguaje.

Ya que en GPT-2 no se aplica una representación de las palabras ni a nivel de palabra ni a nivel de carácter. Se elige una opción que se encuentra situada en medio, que es la subpalabra. La subpalabra se puede obtener mediante el

algoritmo Byte Pair Encoding (BPE).

BPE es una forma de compresión donde se calculará una lista de subpalabras utilizando el siguiente algoritmo: [30]

- Dividir palabra en secuencia de caracteres.
- Unirse al patrón de frecuencia más alta
- Continuar con el paso anterior hasta alcanzar el número máximo pre-definido de subpalabras de iteraciones.

2.6.2. Arquitectura GPT-2

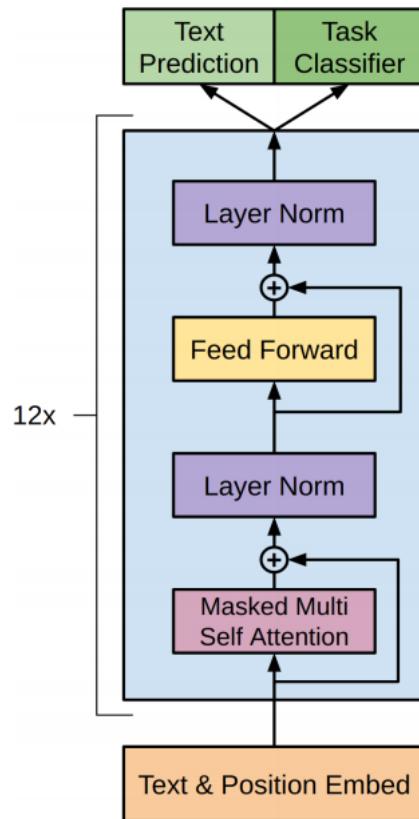


Figura 2.6: Arquitectura del modelo de GPT-2 [31]

El modelo de GPT-2 es un transformer decodificador de 12 capas, cada una con 12 mecanismos de atención independientes, llamados “cabezas”; el

resultado es $12 \times 12 = 144$ patrones de atención distintos. Cada uno de estos corresponde a una propiedad lingüística capturada por el modelo. [32]

Lo que le permite identificar la relación que existe entre las palabras, entendiendo la oración y permitiendo predecir la siguiente palabra en relación a la oración.

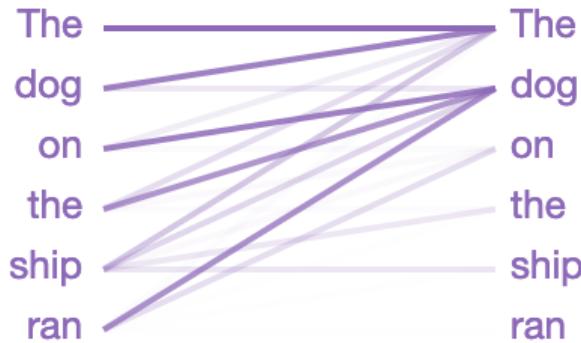


Figura 2.7: Ejemplo análisis de una oración por GPT-2 [33]

Las líneas, leídas de izquierda a derecha, muestran dónde presta atención el modelo prediciendo la siguiente palabra en la oración (la intensidad del color representa la fuerza de la atención). Entonces, al tratar de predecir la siguiente palabra después de correr (ran), el modelo presta mucha atención al perro (dog) en este caso. Esto ya que necesita saber quién o qué está corriendo para predecir qué viene a continuación. [33]

2.7. BERT vs GPT-2

Dentro de esta sección mostramos una pequeña comparación entre las tecnologías BERT y GPT-2, visualizada en la siguiente tabla: Con base en lo anterior se determinó que se va a utilizar BERT el resto del trabajo terminal para ayudarnos en la generación de los textos.

Cuadro 2.1: Comparación entre las tecnologías BERT y GPT-2

BERT	GPT-2
Es de naturaleza bidireccional.	Es de naturaleza autor-regresivo
El usuario puede entrenar sus propios modelos.	Puede resolver distintos problemas de Procesamiento del Lenguaje Natural (PLN) sin necesidad volver a entrenar la red neuronal.
La separación de palabras es manejada en tokens.	La separación de palabras es realizada por medio del algoritmo Byte Pair Encoding (BPE) generando subpalabras.
Su uso es enfocado al análisis y generación de textos.	Sus principales usos son para la generación de textos.

2.8. Base de datos

Una base de datos es un conjunto organizado de datos o información, los cuales pertenecen a un mismo contexto, se encuentran almacenados de forma física o digital con la finalidad de realizar distintas acciones como consultas futuras, ingreso de nuevos datos, actualización o eliminación de estos.

Las bases de datos se componen de una o más tablas divididas en columnas y filas, estas son las encargadas de guardar un conjunto de datos.

Una base de datos generalmente es manejada por un Sistema de gestión de base de datos (DBMS). En conjunto, el DBMS y los datos o información, unido a las aplicaciones asociadas a ellos, se les conoce como un sistema de base de datos.[34]

2.8.1. Base de datos NoSQL o no relacionales

Estas bases permiten que los datos no estructurados y/o semiestructurados se almacenen y manipulen, a diferencia de la base de datos relacional donde se define como deben de componerse todos los datos insertados en esta. Generalmente los registros de este tipo de base de datos suelen almacenarse como un documento de tipo JSON.

2.9. Género musical

Un género musical es hablar de una categoría la cual reúne una diversidad de composiciones musicales las cuales tienen ciertos criterios en común, estos pueden ser su instrumentación, cómo se va a emplear, el contexto en el que son desarrolladas o el contenido de su lirica.

2.9.1. Género Pop

El género de musical popular o mejor conocido como música pop, se trata de una combinación de otros géneros musicales, este tipo de género trata de ser ecléctico, es decir, trata de reunir ideas, tendencias, valores, etc. Aunque como todo, cuenta con ciertas características que lo diferencian, algunas de estas diferencias son que la duración de las canciones suele ser de corta a media duración, siguen una estructura simple de estrofa-estribillo, la cual consta de un grupo de versos a los cuales le acompañan un pequeño grupo de versos los cuales se repiten, por lo mismo, éstos se distinguen por el uso habitual de estribillos repetidos, así como de ganchos los cuales son un pasaje (una melodía) o una frase la cual busca llamar la atención del oyente.

2.10. Canción

Una canción por definición, es una composición literaria, generalmente escrita en versos, a la cual se le puede acompañar con música para poder ser cantada.[35]

2.10.1. Elementos de una canción

Los elementos que conforman una canción son los siguientes:

Introducción

Generalmente es una parte única la cual se encuentra al inicio de una canción, acompañada de una armonía o melodía compuesta solo para este inicio. El objetivo de la introducción es de atraer la atención y producir un ambiente.[36]

Verso

Es la parte encargada de comenzar a desarrollar la idea a transmitir, trata de contarnos el tema de la canción, y ya cuenta con una armonía bien

establecida.

Pre-coros

Es un arreglo que permite realizar una transición, su función principal es conectar el verso y coro. También ayuda a evitar que el coro se estanque en la monotonía.

Coro

Es una estrofa la cual se repite varias veces dentro de una composición. Su función principal es acentuar la idea más importante de la canción tanto en su letra como en lo musical. Se considera como una de las partes más importantes dentro de una canción y en algunas ocasiones este es repetido al inicio y final.

Puente

Es un interludio el cual conecta dos fragmentos de una canción, permitiendo construir una armonía entre ellas, suele ser usado para llevar a la canción al clímax, preparándose para el desarrollo final de la canción.

Cierre

Este busca terminar o concluir la pieza musical, una de las formas puede ser un ruptura brusca generada por un silencio imprevisto o por una secuencia de acordes. Pero la manera más ordinaria de cerrar es haciendo uso de la repetición de un coro.

2.10.2. Estructura de una canción

La estructura mínima de una canción está compuesta de:

- Verso
- Coro
- Verso
- Coro

La estructura más empleada en una canción es la siguiente:

- Introducción
- Verso
- Pre-coros
- Coro
- Verso
- Coro
- Puente
- Cierre

Se pretende trabajar la estructura de la generación de letras musicales de la siguiente forma:

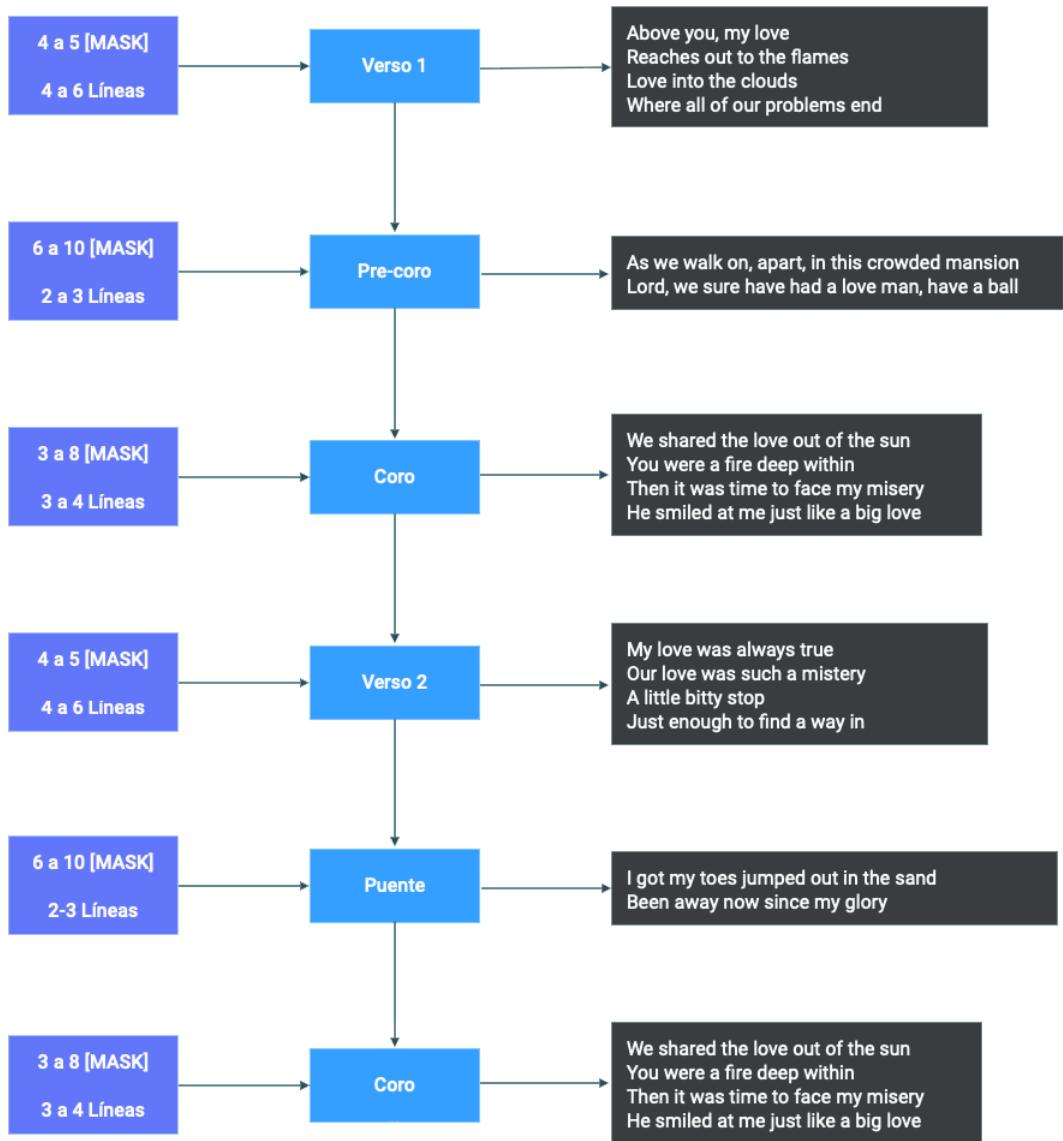


Figura 2.8: Diagrama de la estructura para la generación de las letras musicales

2.11. Cadenas de Markov

Las cadenas de Markov son un sistema matemático el cual experimenta con las transiciones de un estado a otro de acuerdo con ciertas reglas probabilísticas. La característica que define a una cadena de Markov es que no importa cómo llegó el proceso a su estado actual, y sus posibles estados futuros son fijos. En otras palabras, la probabilidad de pasar a cualquier otro estado depende únicamente del estado actual y del tiempo transcurrido. El espacio de estados o conjunto de todos los posibles estados, pueden ser cualquier cosa: letras, números, puntuaciones de un partido, acciones, etc.

Son procesos estocásticos, con la diferencia de que estos deben ser “sin memoria”, es decir, la probabilidad de las acciones futuras no depende ni se ve afectada de los pasos que la condujeron al estado actual. A esto se le denomina una propiedad de Markov.

En la teoría de probabilidad, el ejemplo más inmediato es el de una cadena de Markov homogénea en el tiempo, en la que la probabilidad de que ocurra cualquier transición de estado es independiente del tiempo.

En el lenguaje de probabilidad condicional y variables aleatorias, una cadena de Markov es una secuencia X_0, X_1, X_2, \dots de variables aleatorias que satisfacen la regla de independencia condicional. En otras palabras, el conocimiento del estado anterior es todo lo que se necesita para determinar la distribución de probabilidad del estado actual. Esta definición es más amplia que la explorada anteriormente, ya que permite probabilidades de transición no estacionarias y, por lo tanto, cadenas de Markov no homogéneas en el tiempo; es decir, a medida que pasa el tiempo los pasos aumentan y la probabilidad de pasar de un estado a otro puede cambiar. [37]

Las cadenas de Markov pueden ser modeladas mediante máquinas de estados finitos.

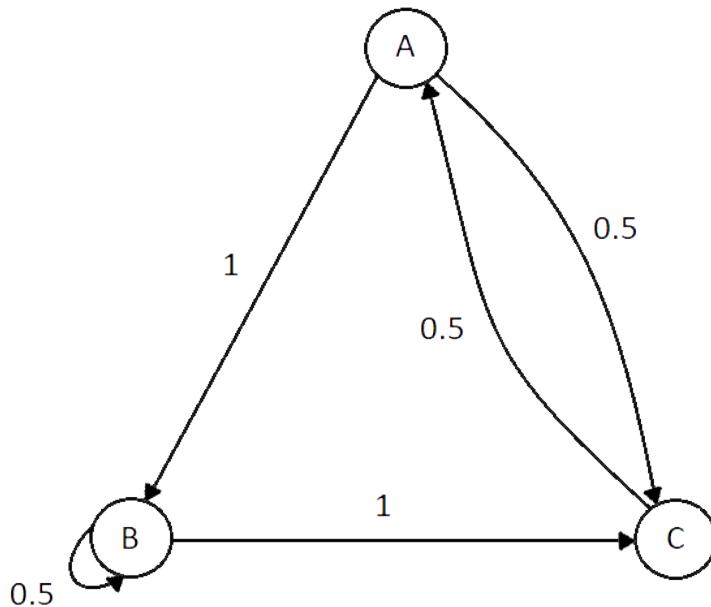


Figura 2.9: Diagrama de estados finitos de una cadena de Markov [37]

2.12. Apache

Es un servidor web gratuito y de código abierto el cual permite que los desarrolladores de sitios web puedan desplegar el contenido de ellas, este no es un servidor que se encuentre físicamente, sino que se trata de un software el cual ejecuta un servidor. Su función es instaurar la conexión entre un servidor y los usuarios del sitio web mientras estos intercambian archivos entre ellos (siguiendo una estructura cliente-servidor). El servidor y cliente interactúan mediante el protocolo HTTP, y el software de Apache es el encargado de mantener una comunicación ágil y segura entre las 2 partes.[44]

Apache trabaja sin problemas con otros sistemas de gestión de contenido (Drupal, Joomla, etc.), marcos de trabajo (Django, Laravel, etc.), y lenguajes de programación. Por estas razones se vuelve una opción sólida al momento de escoger entre los distintos tipos de plataformas de alojamiento web, como serían Virtual Private Server (VPS) o alojamiento compartido.[45]

Cuadro 2.2: Tabla comparativa de los distintos servidores web contemplados para nuestro proyecto

Apache	NGINX [38]
Es fácil de configurar, cuenta con muchos módulos, así como un entorno sencillo.	Usa hilos para manejar solicitudes del usuario.
De código abierto y gratuito, inclusive si se usa de manera comercial.	Nginx es uno de los servidores web que soluciona el problema c10k y seguramente de los más exitoso al hacerlo.
Software estable y confiable.	No crea un nuevo proceso por cada solicitud.
Frecuentemente actualizado incluyendo actualizaciones regulares a los parches de seguridad.	Maneja toda solicitud entrante en un único hilo.
Funciona de forma intuitiva, con sitios web hechos con WordPress.	El modelo basado en eventos de Nginx reparte las solicitudes de los usuarios entre procesos de trabajo de manera eficiente.
Comunidad grande y posibilidad de contactar con soporte disponible de manera sencilla en caso de cualquier problema.	Cuenta con mejor escalabilidad.
Presenta inconvenientes de rendimiento con sitios web los cuales tienen alto tráfico.	Es capaz de manejar un sitio web con demasiado tráfico con un uso de recursos mínimo.
Al contar con muchas preferencias de configuración puede generar vulnerabilidades de seguridad.	

En ambos casos, se mantienen en desarrollo, con actualizaciones constantes, así como frecuentemente sacando parches para evitar ataques de DDoS.

2.13. Servidor web

La función es mostrar sitios web en internet. A fin de conseguir este objetivo, actúa como un mediador entre el servidor y el dispositivo del cliente. Obtiene el contenido del servidor en cada petición hecha por el usuario y lo entrega al sitio web.[46]

Los servidores web son capaces de procesar archivos escritos en distintos lenguajes de programación como Java, PHP, Python, y otros.

Podría decirse que un servidor web es la herramienta responsable por la correcta comunicación entre el cliente y el servidor.

2.14. Certificado SSL

Conocido como Secure Socket Layer (SSL) o (Capa de Conexión Segura) es un estándar de seguridad global que fue originalmente creado por Netscape en los 90's. SSL crea una conexión encriptada entre tu servidor web y el navegador web de tu visitante permitiendo que la información privada sea transmitida sin que ocurran problemas como serían espionaje, manipulación de la información, y falsificación de los datos del mensaje.[47] Básicamente, la capa SSL permite que dos partes tengan una “conversación” privada.

Para establecer esta conexión segura, se instala en un servidor web un certificado SSL (también llamado ‘certificado digital’) que cumple dos funciones:

- Autentificar la identidad del sitio web, garantizando a los visitantes que no están en un sitio falso.
- Cifrar la información transmitida.

Hay varios tipos de certificados SSL [48] según la cantidad de nombres de dominio o subdominios que se tengan, como por ejemplo:

- Único: Asegura un nombre de dominio o subdominio completo (Fully Qualified Domain Name (FQDN) por sus siglas en inglés).
- Comodín: Cubre un nombre de dominio y un número ilimitado de sus subdominios.
- Multidominio: Asegura varios nombres de dominio.

2.15. Plataformas en la nube para generar modelos de aprendizaje automático

El entrenamiento de aprendizaje automático y de modelos de aprendizaje profundo involucra miles de iteraciones. Se necesitan esta gran cantidad de iteraciones para producir el modelo más preciso.

El cómputo en la nube permite modelar capacidad de almacenamiento y manejar cargas a escala [49], o escalar el procesamiento a través de los nodos. Por ejemplo, AWS ofrece instancias de GPUs con capacidad de memoria que va de los 8Gb's a los 256Gb's, estas instancias son cobradas a ritmos por hora.

Las GPUs son procesadores especializados diseñados para procesado complejo de imágenes. Azure de Microsoft ofrece GPUs de alto rendimiento de la serie NC para aplicaciones o algoritmos de cómputo de alto rendimiento.

2.15.1. Amazon Web Services

Dentro de las bondades que ofrecen los Servicios Web de Amazon (Amazon Web Services (AWS) por sus siglas en inglés) y que nos pueden ser de utilidad para las necesidades de nuestro proyecto se pueden encontrar:

- **SageMaker:** Es una plataforma de aprendizaje automático completamente administrado para científicos de datos y desarrolladores. La plataforma corre en Cómputo Elástico en la Nube (Elastic Compute Cloud (EC2) por sus siglas en inglés), y permite construir modelos de aprendizaje automático, organizar la información y escalar sus operaciones.

Algunas aplicaciones de aprendizaje automático en SageMaker van desde reconocimiento de voz hasta visión de computadora, e incluso recomendaciones basadas en el comportamiento aprendido del usuario.

El mercado de AWS ofrece modelos que se pueden usar en lugar de empezar desde cero, posterior a eso se puede entonces empezar a entrenar y optimizar el modelo; las elecciones más comunes son frameworks como Keras, TensorFlow, y PyTorch; Sagemaker puede optimizar y configurar estos frameworks automáticamente, o pueden ser entrenadas de manera personal.

Uno mismo puede incluso desarrollar su propio algoritmo construyéndolo en un contenedor de Docker o se puede hacer uso de una “Jupyter notebook” [51] para construir un modelo propio de aprendizaje automático y visualizar su información usada para el entrenamiento del modelo siendo este punto lo que más nos interesa para nuestro proyecto.

Cuadro 2.3: Tabla comparativa de las diversas plataformas contempladas

Característica	GCP [68]	AWS [69]	Azure [70]
Jupyter Notebook alojado de manera local o remota	Plataforma de IA	SageMaker Studio IDE	- Estudio de Notebooks de Azure de Aprendizaje Automático - Databrick de Azure
Entrenamiento Distribuido	Si	Si	Si
Versionado de Modelos	Si	Si	Si
Seguimiento de Experimentos	Si	Si	Si
AutoML (UI y API)	Tabla de AutoML	Auto-piloto de SageMaker	AutoML
Análisis de Errores	Tabla de AutoML con BigQuery	Debugger de SageMaker	Aprendizaje Profundo de Azure

La tabla anterior se ha basado en la investigación de otras comparaciones en línea [50] y se ha visto que los tres proveedores han alojado servicios de “Jupyter Notebook” (contienen tanto código de computadora como elementos ricos en texto como ecuaciones, figuras, etc.), experimentando seguimientos y control de versiones, y métodos de despliegue sencillos.

Dentro de las características únicas de cada una de las plataformas podemos encontrar que la Plataforma en la Nube de Google (GCP) usa un paquete llamado “what if tool” el cual se puede integrar junto a un “Jupyter Notebook” y de esa manera jugar con el modelo cambiando el umbral o un valor característico de un ejemplo dado, esto permite checar como es que ciertos cambios afectan el resultado predicho con anterioridad previo al cambio.

El debugger de SageMaker de AWS permite analizar cómo es que la ingeniería de características y el refinado del modelo son hechos, o de forma más concisa, permite ver qué sucede durante el entrenamiento del modelo.

Azure, por su parte, provee un módulo propio en su SDK el cual parece tener la mejor integración de entre las tres plataformas.

Costo y Rendimiento del Modelo

Azure y GCP puntúan ligeramente mejor que AWS en términos de rendimiento, esto no necesariamente significa que una plataforma es mejor que otra.

El costo de AWS fue considerablemente menor que GCP y Azure. Por supuesto, cada una tiene sus fuertes como puede verse en la tabla presentada anteriormente.

El diseñador de aprendizaje automático de Azure cuenta con una interfaz de arrastar y soltar el cual es bastante amigable con un usuario novato, esto es, porque requiere menos codeo y antecedentes técnicos. AWS y GCP parecen ser más enfocados a desarrolladores aunque puede resultar en más de trabajo ensamblar una cadena de procesos (pipeline), estos resultan más personalizables con los diferentes componentes disponibles. Estos componentes y la conexión de la cadena de procesos son usualmente desarrollados usando código y configuraciones, en vez de utilizar una interfaz.

Tanto GCP como AWS ofrecen un modelo de pago “pay-as-you-go”. Este modelo es el mejor para aquellos individuos que puedan llegar a esperar un

uso intermitente de la nube, ya que permite un enfoque flexible para añadir y remover servicios cuando se necesite. Por supuesto, este nivel de flexibilidad tiene un costo, haciendo al modelo “pay-as-you-go” el más caro por hora.

Cuadro 2.4: Tabla comparativa GCP vs AWS

Tipo de Instancia	Precio de EC2 (por hora)	Precio de Google (por hora)
Propósito General	\$0.134	\$0.15
Cómputo Optimizado	\$0.136	\$0.188
Optimizado de Memoria	\$0.201	\$0.295
GPU	\$0.526	\$1.4

La tabla mostrada anteriormente cabe destacar que es en dólares estadounidenses y se utilizó una comparación hecha en otro artículo. [52] En conclusión, la opción ideal para las necesidades de nuestro proyecto es AWS debido a que es más económico que las otras dos plataformas y cuenta con herramientas más útiles como SageMaker que nos permitirá saber cómo se está entrenando el modelo.

Capítulo 3

Análisis

3.1. Problemática

Los nuevos artistas musicales suelen frustrarse durante el proceso creativo para la composición de una letra musical, debido a que no cuentan con la creatividad o el tiempo para realizarlas, además de que esta tarea puede ser agotadora, difícil y estresante, en consecuencia, pueden llegar a abandonar sus sueños y aspiraciones.

Estos artistas pueden contar con ciertos servicios los cuales ofrecen letras de canciones, pero estas suelen estar precios muy altos o estas no cumplen con las expectativas de los artistas al invertir en ellas.

3.2. Nuestra solución

Debido a lo anterior se ha decidido desarrollar una herramienta de apoyo para estudiantes o aficionados interesados, a los cuales se les dificulte componer nuevas letras musicales, con la finalidad de impulsar la carrera de futuros artistas en la industria musical que no cuenten con los recursos suficientes para poder contratar servicios particulares de compositores.

Dicha herramienta de apoyo se realizará con ayuda de la inteligencia artificial, así como de un modelo de red neuronal recurrente para la generación de las nuevas letras musicales, los usuarios de esta herramienta podrán interactuar con la herramienta para generar letras personalizadas según sus propuestas dadas.

Para corroborar que sea la mejor opción, se implementarán pruebas con otras

herramientas como BERT y Cadenas de Markov con las que podremos comparar el resultado.

3.2.1. ¿Por qué utilizar inteligencia artificial?

Se eligió la inteligencia artificial como tecnología principal para desarrollar este sistema, se quiere investigar el funcionamiento de las redes neuronales y su impacto que puede llegar a tener en el área de la creatividad y generación de texto, que es un campo relativamente nuevo y con muchos alcances que aún quedan por descubrir.

Cabe destacar que aunque es un tema relacionado con la ingeniería en sistemas computacionales, no es algo que se aprenda como tema principal, por lo que requerirá una ardua investigación y desarrollo para poder lograr los objetivos propuestos.

La inteligencia artificial nos permite buscar patrones y al mismo tiempo soluciones nuevas y eficientes, para problemas que antes no se podía saber que una computadora podría hacer.

3.2.2. ¿Por qué utilizar procesamiento de lenguaje natural?

El Procesamiento de Lenguaje Natural, es una rama de la inteligencia artificial que es estudiada desde 1940-1950[81] y ha tenido avances que han podido ayudar a diferentes áreas tales como:

- Análisis de Sentimientos
- Clasificación de Texto
- Chatbots & Asistentes Virtuales
- Extracción de texto
- Traductores
- Auto correctores. Y en estos últimos años gracias al hardware que se tiene hoy en día, es posible tener avances en la generación de texto.

3.2.3. ¿Por qué utilizar redes neuronales Long Short Term Memory (LSTM)?

Se hará uso de este tipo de redes neuronales debido a que se ha visto que destacan para la generación de texto [78], sobre todo de textos que requieren de contexto, esto es, que las palabras previamente escritas servirán para la inferencia de las posibles palabras futuras. Otros tipos de redes neuronales como las convolucionales o las de multicapas se enfocan en trabajar las palabras de una en una como van llegando sin tomar en cuenta el contexto o sentido dado por las palabras previas, dando poca o nula coherencia al texto generado por estas.

En especial se pretende trabajar con las redes recurrentes de tipo LSTM debido a que estas pueden “recordar” estados previos y utilizar la información originada de esos estados previos para así poder generar con mayor eficiencia el resultado siguiente, la cual es una característica que no tiene una red neuronal recurrente estándar que en general solo se retroalimenta de un estado pasado o una GRU (Gated Recurrent Unit) la cual en términos generales tiene dos puertas las cuales controlan que información debe mantenerse en memoria así como cuál debe de entrar para reajustar el modelo y la salida a generar.

3.3. Herramientas a utilizar

3.3.1. Software

Para el desarrollo de software del prototipo, es necesario hacer mención de algunas de las siguientes herramientas y lenguajes de programación, para tener una idea clara sobre las herramientas que se usarán el motivo de su uso:

Python

Python es un lenguaje de programación orientado a objetos y de alto nivel con semántica dinámica. Sus estructuras de datos integradas de alto nivel, combinadas con tipado y enlace dinámico, lo hacen muy atractivo para el desarrollo rápido de aplicaciones, así como para su uso en scripts o para conectar componentes ya existentes. La sintaxis simple y fácil de aprender de Python enfatiza la legibilidad y, por lo tanto, reduce el costo de mantenimiento del programa. Python admite módulos y paquetes, lo que fomenta la modularidad del programa y la reutilización del código.[57]

Las principales ventajas de Python son:

- Es fácil y rápido desarrollar una aplicación.
- Cuenta con una gran cantidad de librerías
- Es fácil de entender el código y darle mantenimiento.

Principales desventajas son:

- En ocasiones la ejecución del código es lenta.
- Consume mucha memoria.

BERT

Bidirectional Encoder Representations from Transformers (BERT) es un marco de aprendizaje automático de código abierto para el Procesamiento del Lenguaje Natural (PLN). BERT, significa Representaciones de codificador bidireccional de transformers, el cual es un modelo de aprendizaje profundo en el que cada elemento de salida está conectado a los de entrada y las ponderaciones entre ellos se calculan dinámicamente en función de su conexión. [26]

Al contar con la capacidad bidireccional, BERT está previamente entrenado en dos tareas de PLN diferentes, pero relacionadas: el modelado de lenguaje enmascarado y la predicción de la siguiente oración.

El objetivo del entrenamiento del Modelado de Lenguaje Enmascarado (MLM) es ocultar una palabra en la oración y luego hacer que el programa prediga qué palabra se ha ocultado en función del contexto de la palabra oculta. El objetivo del entrenamiento de predicción de la siguiente oración es que el programa prediga si dos oraciones dadas tienen una conexión lógica secuencial o si su relación es simplemente aleatoria.

HTML

Hypertext Markup Language (HTML) es un lenguaje de marcado que define la estructura de una página web y su contenido. HTML consta de una serie de elementos que se utilizan para encerrar o envolver diferentes partes del contenido para que estos se visualicen o actúen de cierta manera. Las etiquetas adjuntas pueden hacer que una palabra o imagen sea un hipervínculo a otro lugar, pueden poner palabras en cursiva, hacer que la fuente sea más grande o pequeña, etc. [39]

HTML5 es la versión más reciente de HTML, la cual integra nuevos elementos, atributos y comportamientos. Permite describir de mejor manera el contenido de la página web, así como mejora su conectividad con el servidor y almacenamiento, posibilita que las páginas web puedan operar sin conexión usando los datos almacenados localmente del lado del cliente, otorga un mejor soporte al contenido multimedia, así como una mejor integración a APIs y un mejor diseño usando CSS3. [40]

CSS

Cascading Style Sheet (CSS) es el lenguaje para describir la presentación de las páginas web, así como hacerlas más atractivas. Permite adaptar la presentación a diferentes tipos de dispositivos. CSS es independiente de HTML y puede ser empleado con cualquier otro lenguaje de marcado basado en XML o SVG. Usando CSS se pueden controlar con precisión cómo se ven los elementos HTML en el navegador, que presentará para las etiquetas de marcado el diseño que cada uno desee. La separación de HTML de CSS facilita el mantenimiento de los sitios, compartir las hojas de estilo entre páginas y adaptarlas a distintos ámbitos. [41]

Es un lenguaje basado en reglas: cada usuario define las reglas que especifican los grupos de estilos que van a aplicarse a elementos particulares o grupos de elementos de la página web.

Antes de CSS, las etiquetas como fuente, color, estilo de fondo, alineación, borde y tamaño tenían que repetirse en cada elemento de una página web. Ahora con los CSS, podemos definir cómo se van a comportar las etiquetas, al ser guardado en un archivo por separado, esta misma configuración puede usarse en otra página web ahorrando tiempo diseñándola. Además de que CSS provee de mejor y más detallados atributos para cada etiqueta.

JavaScript

JavaScript es un lenguaje de programación o secuencias de comandos que permite implementar funciones complejas en las páginas web. Estos scripts pueden ser desarrollados en el mismo HTML para que sean ejecutados automáticamente cuando se carga dicha páginas web, estos scripts se proporcionan y ejecutan como texto sin formato. No necesitan una preparación especial ni una compilación para ejecutarse. [42]

JavaScript puede ejecutarse no solo en un navegador, sino también en un

servidor, o en cualquier dispositivo que tenga un programa especial llamado JavaScript engine, el cual permite interpretar y ejecutar los scripts.

JavaScript permite crear contenido dinámico dentro de las páginas web, reaccionar ante algunas acciones realizadas por los usuarios como lo son los clics del ratón, el movimiento del puntero o el presionar cierta tecla, permite enviar peticiones al servidor, así como descargar y subir archivos, además es posible obtener y configurar cookies, mostrar mensajes o alertas al usuario.

OpenSSL

Consiste en un paquete robusto de herramientas de administración y bibliotecas las cuales están relacionadas con la criptografía, estas suministran funciones criptográficas a otros paquetes como OpenSSH y navegadores web (permitiendo un acceso seguro a sitios HTTPS) [59]. Estas herramientas contribuyen al sistema a instaurar el protocolo Secure Socket Layer (SSL), de igual manera como otros protocolos relacionados con la seguridad, como el Transport Layer Security (TLS). OpenSSL posibilita la creación certificados digitales los cuales pueden aplicarse a un servidor, por ejemplo a Apache [43].

MongoDB

MongoDB es un sistema de base de datos multiplataforma dirigido a documentos, de esquema libre, es decir, que cada registro o entrada es capaz de contar con un esquema de datos distinto, con columnas o atributos que pueden variar o no de un registro a otro.

Sus características más destacadas son su sencillo sistema de consulta de la base de datos y la velocidad. Alcanzando así un balance perfecto entre rendimiento y funcionalidad. MongoDB utiliza un modelo NoSQL el cual es un modelo de agregación que se basan en la noción de agregado, entendiendo el agregado como una colección de objetos relacionados que se desean tratar de forma semántica e independiente [58].

Las ventajas que ofrece MongoDB como herramienta de desarrollo de base de datos no relacionales son:

- La base de datos no tiene un esquema de datos predefinido.
- El esquema puede variar para instancias de datos que pertenecen a una misma entidad.
- En ocasiones el gestor de la base de datos no es consciente del esquema de la base de datos.

- Permite reducir los problemas de concordancia entre estructuras de datos usadas por las aplicaciones y la base de datos.
- Frecuentemente se aplican técnicas de des-normalización de los datos.

Flask

Flask es un mini marco (framework) web, esto es, un módulo de Python el cual permite desarrollar aplicaciones web. No cuenta con un Manejador de Objetos Relacionales u ORM por sus siglas en inglés, pero si cuenta con características como el enrutamiento de URLs y un motor de plantillas. En general es un marco de aplicación web WSGI.

La Web Server Gateway Interface (WSGI) es una especificación que describe como se va a comunicar un servidor web con una aplicación web, y como se pueden llegar a enlazar distintas aplicaciones web para procesar una solicitud o una petición.

Las principales ventajas de Flask son:

- Permite escalar con facilidad la aplicación.
- Es de fácil desarrollo.
- Es flexible con la plantillas que te da por default.
- Es modular.

Principales desventajas son:

- No cuenta con muchas herramientas o librerías de apoyo.
- No puede manejar peticiones múltiples al mismo tiempo.

Gunicorn

Gunicorn, también conocido como unicornio verde “Green Unicorn”, es una de las muchas implementaciones de un Web Server Gateway Interface (WSGI) y se usa comúnmente para ejecutar aplicaciones web hechas con Python. Esta implementa la especificación WSGI de frameworks como Django, Flask o Bottle.

Amazon EC2

Amazon Elastic Compute Cloud (EC2) [82] proporciona una infraestructura de tecnologías de información que se ejecuta en la nube y funciona como un centro de datos que se ejecuta en su propia sede. Es ideal para empresas que necesitan rendimiento, flexibilidad y potencia al mismo tiempo.

Amazon EC2 es un servicio que permite alquilar un servidor o máquina virtual de forma remota para ejecutar aplicaciones. Las principales ventajas de Amazon EC2 son:

- Cuenta con plantillas predeterminadas de máquinas virtuales y servidores.
- Permite configurar la memoria, almacenamiento, CPU y otras características dependiendo las necesidades del usuario.
- Precios que dependen del uso y características del equipo o servidor alquilado.

Las principales desventajas son:

- Tiene una curva de aprendizaje alta para los nuevos usuarios.
- Los costos de soporte técnico son muy elevados.

Amazon SageMaker

Amazon SageMaker [83] es un servicio que ayuda a científicos y desarrolladores a construir, entrenar e implementar de manera rápida y sencilla modelos de machine learning.

Para construir el modelo, este servicio cuenta con algoritmos de machine learning más utilizados que vienen preinstalados. También está preconfigurado para que pueda ejecutar Apache MXNet y TensorFlow.

Para el entrenamiento, con un solo clic en la consola de servicio, es fácil comenzar a entrenar su modelo. Amazon SageMaker se encarga de cada infraestructura y facilita la escalabilidad. lo que permite entrenar los modelos a escala de peta-bytes Si se desea acelerar y simplificar el proceso de entrenamiento, se puede ajustar automáticamente el modelo para obtener la mejor precisión.

Para desplegar el modelo entrenado, se aloja en un clúster de escalado automático de Amazon EC2.

Amazon S3

Amazon Simple Storage Service (S3) [84], como su nombre lo indica, es un servicio web proporcionado por Amazon Web Services (AWS) que proporciona almacenamiento altamente escalable en la nube.

Apache

Es un servidor web gratuito y de código abierto el cual permite que los desarrolladores de sitios web puedan desplegar el contenido de ellas, este no es un servidor que se encuentre físicamente, sino que se trata de un software el cual ejecuta un servidor [60]. Su función es instaurar la conexión entre un servidor y los usuarios del sitio web mientras estos intercambian archivos entre ellos (siguiendo una estructura cliente-servidor). El servidor y el cliente interactúan mediante el protocolo HTTP, y el software de Apache es el encargado de mantener una comunicación ágil y segura entre las 2 partes.

Apache trabaja sin problemas con otros sistemas de gestión de contenido (Drupal, Joomla, etc.), marcos de trabajo (Django, Laravel, etc.), y lenguajes de programación. Por estas razones se vuelve una opción sólida al momento de escoger entre los distintos tipos de plataformas de alojamiento web, como serían Virtual Private Server (VPS) o alojamiento compartido.

3.3.2. Hardware

Se usarán los equipos de cómputo con los que los integrantes del equipo contamos actualmente, los cuales se especifican a continuación:

Cuadro 3.1: Equipo de cómputo 1

Equipo de cómputo utilizado.	
Procesador	Ryzen 5 3600
Tarjeta de video	Amd Radeon Rx580
Memoria RAM	32 Gb
Disco duro	1Tb HDD y 512Gb SSD

Cuadro 3.2: Equipo de cómputo 2

Equipo de cómputo utilizado.	
Marca	Apple
Modelo	iMac Late 2012
Procesador	Intel Core i5
Tarjeta de video	NVIDIA GeForce GT 640M 512 Mb
Memoria RAM	8 Gb
Disco duro	1Tb y 256 Gb SSD

Cuadro 3.3: Equipo de cómputo 3

Equipo de cómputo utilizado.	
Procesador	Amd FX-8350
Tarjeta de video	Nvidia Geforce 1050ti
Memoria RAM	16 Gb
Disco duro	1Tb HDD

3.4. Algoritmos analizados

La Inteligencia Artificial es un campo emergente donde se desarrollan nuevas aplicaciones de innovación continuamente. Para este generador de letras musicales será necesario implementar diferentes estrategias para poder evaluar cuál es la mejor opción, para ello, se utilizarán tres recursos importantes para ver la evolución de las pruebas y poder evaluar el resultado óptimo a nuestras posibilidades para poder implementarlo en este proyecto.

3.4.1. Cadenas de Markov

Este sería el algoritmo más sencillo a utilizar, consiste en generar texto de acuerdo a otro texto previo, sin embargo funciona de manera mecánica dando valores aleatorios para generar un texto, posiblemente los enunciados que se generan tienen una estructura medianamente correcta, sin embargo el sentido que tendrá el texto no tendrá ningún sentido, se hará esta prueba para ver el alcance que se tiene con este tipo de algoritmo y poder comparar los resultados finales del proyecto.

3.4.2. Recurrent Neural Network

Anteriormente se explicó lo que eran las redes neuronales recurrentes, en el algoritmo, se aplica para predecir palabras dependiendo de los datos anteriores, a esto se le llaman datos secuenciales, debido a que este algoritmo recuerda la entrada debido a su memoria interna.

Centrándonos en la generación de nuevas letras musicales, si observamos atentamente el uso de este tipo de redes neuronales, los datos serán secuenciales ya que las letras de canciones dependen una de otra. Por lo tanto se usarán redes neuronales del tipo Long Short Term Memory (LSTM) de Recurrent Neural Network (RNN).

Este tipo de redes neuronales son una extensión de Recurrent Neural Network (RNN) que ayuda a recordar, en el siguiente diagrama se muestran los componentes de una red neuronal de este tipo:

- Estado de la celda
- Olvida la puerta
- Puerta de entrada
- Puerta de salida

- Memoria celular actual
- Salida de celda actual

Los resultados de este tipo de redes neuronales puede variar según el dataset que se utiliza y si la limpieza se hizo correctamente, de igual forma importa mucho el hardware utilizado para que el tiempo de generación del modelo no sea tardado y no consuma más de lo que tiene la computadora.

3.4.3. BERT

Como última opción de algoritmo, se utilizará BERT para generar nuevo texto, a través de las llamadas máscaras y separadores, es posible crear nuevas oraciones, sin embargo si se usa esta herramienta, es necesaria la implementación de otro tipo de herramientas, ya que al ser bidireccional, la generación de texto no saldrá si no se ingresa algún texto previamente.

La idea es utilizar una API que genere palabras con rima y generar cierta cantidad de rimas que sirvan como texto previo para que BERT pueda generar algún texto sin ningún problema.

3.5. Pruebas

3.5.1. Pruebas con cadenas de Markov

Una de las primeras pruebas que se realizó durante el desarrollo del proyecto fue la generación de textos utilizando cadenas de markov. Se ingresará el dataset limpio y organizado para realizar un cálculo y asignación de los valores estos serán las oraciones y palabras que aparecen. Los valores asignados variarán entre los decimales de 0 y 1 sin llegar a ser enteros nunca, estos son asignados dependiendo a la frecuencia en que la palabra va apareciendo.

Una vez asignadas, es posible determinar si la generación del texto se desea pseudo-aleatoria o utilizar valores que cumplan ciertas condiciones como generar textos arriba o abajo de 0.5 o el tamaño del texto generado.

Para esta prueba se decidió no influir en la selección de palabras por sus valores, dejando este atributo pseudo-aleatorio, sin embargo para no generar textos de tamaños diferentes, los parámetros de las palabras fueron manipulados para que cada oración diera un determinado número, esto para evitar que los versos fuesen de un tamaño no deseado.

A continuación se muestra el código utilizando nuestra base de datos para generar texto usando cadenas de Markov:

```
1 import markovify
2 import random
3 import pandas as pd
4 import os
5
6 def versos(cantidad):
7     for sentence in range(cantidad):
8         generatedlyrics = markovifyModel.make_short_sentence(random.choice(
9             lenghtlyric))
10        print(generatedlyrics)
11
12 def generarcoros(cantidad):
13     for sentence in range(cantidad):
14         generatedlyrics = markovifyModel.make_short_sentence(random.choice(
15             lenghtcoros))
16         coro.append(generatedlyrics)
17
18 def printcoro():
19     for sentence in coro:
20         print(sentence)
21
22 lenghtversos = [4, 5, 6]
23 lenghtlyric = [80, 100, 120]
24 lenghtcoros = [60, 80, 100]
25 coro = []
26 m_columns = ['Lyric']
27 for _ in range(4): os.chdir('..')
28 path = os.path.abspath(os.getcwd() + "/Datasets" + "/Pop.csv")
29 df = pd.read_csv(path, encoding='latin-1', usecols=m_columns)
30 markovifyModel = markovify.Text(df['Lyric'])
31
32 markovifyModel = markovifyModel.compile()
33 generarcoros(random.choice(lenghtversos))
34 for beginning in range(1,2):
35     print("\nVerse " + str(beginning))
36     versos(random.choice(lenghtversos))
37     print("\nChorus")
38     printcoro()
39     for medium in range(3,4):
40         print("\nVerse " + str(medium))
41         versos(random.choice(lenghtversos))
42     print("\nChorus")
43     printcoro()
44     print("\nChorus")
45     printcoro()
```

A continuación un ejemplo de los resultados obtenidos:

```
1 Verse 1
2 It's where I am cannibal.
3 My first love broke my heart was stolen.
4 But you rather be!. Every moment, everything can change your
   life, I'mma change your name.
5 My fairytale comes close to me.
6 And I know that I've got to keep.
7
8 Chorus
9 She let her head in my soul in a skirt.
10 Safe in the plastic prizes never changes.
11 You win my Grammy. you commin cause I came up to the people
    say.
12 With a boy in the corner so close.
13
14 Verse 2
15 You son of a hustla,. Of a cemetery in the car.
16 And then maybe it's just the way to turn.
17 I'll pray for me.
18 And if there's no way to you.
19
20 Chorus
21 She let her head in my soul in a skirt.
22 Safe in the plastic prizes never changes.
23 You win my Grammy. you commin cause I came up to the people
    say.
24 With a boy in the corner so close.
25
26 Chorus
27 She let her head in my soul in a skirt.
28 Safe in the plastic prizes never changes.
29 You win my Grammy. you commin cause I came up to the people
    say.
30 With a boy in the corner so close.
```

Como se puede observar, el texto carece de sentido alguno y en algunas ocasiones se observa que la cadena completa es parte de una canción, por lo que resulta peligroso si el usuario desea usar esta cadena por cuestiones de derecho de autor.

En conclusión, se optó por no utilizar cadenas de Markov debido a que en ocasiones las oraciones que genera, en realidad es texto entero reutilizado de las oraciones que se encuentran en el dataset, de igual forma llegaron a salir resultados utilizando la misma palabra en repetidas ocasiones, se tiene en cuenta que los usuarios objetivos este proyecto, se encontrarán en posición

de buscar ideas nuevas para su proyecto, ideas que contengan un sentido y palabras acordes a lo que ellos buscan, es por ellos que seguimos probando con otros algoritmos, para ver las matemáticas detrás de este proceso, consulte la sección de Anexos .1.

3.5.2. Pruebas con recurrent neural networks

Como segunda prueba, se recurrió a utilizar este tipo de redes neuronales [85], y utilizando como optimizador adam[79] el cuál nos servirá para generar el modelo de la mejor forma de acuerdo con el artículo [80]se han tenido resultados satisfactorios utilizando este tipo de redes neuronales. En la sección de anexos .2 se mostrará el notebook hecho en Jupyter para mostrar cada paso de la prueba junto con todos los resultados obtenidos.

A continuación mostramos algunos de los resultados obtenidos:

```
complete_this_song("Love flowers", 80)

"Love flowers get to her love and you make it okay but i'm not here i still still make you w
anna be the one for you ooh oh oh oh oh baby baby baby baby baby baby baby baby you're
gonna get you home and you go standing in your way and i don't want to miss a thing because
i'm happy clap along if you know what happiness is is enough for me and you can do i know th
at"
```

Figura 3.1: Texto generado utilizando LSTM y la oración en inglés "Love flowers"

```
complete_this_song("It's a cruel and random world", 80)

"It's a cruel and random world is not much between how how could you do and i see your life
is a ghost and my heart is a ghost town my heart is a ghost town my heart is a ghost town my
heart is a ghost town my heart is a ghost town my heart is a ghost town my heart is a ghost
town my heart is a ghost town my heart is a ghost town my heart is a ghost town my heart"
```

Figura 3.2: Texto generado utilizando LSTM y la oración en inglés "It's a cruel and random world"

3.5.3. Pruebas con BERT

Como última prueba fue generar algunos textos utilizando BERT, pero desafortunadamente estas pruebas no fueron exitosas debido a que si se desea trabajar con este algoritmo se requiere hacer uso de Tensorflow y para trabajar con esta biblioteca es necesario contar con una tarjeta de video de Nvidia con características específicas como las son la capacidad de la memoria de dicha tarjeta, hardware con el que no contamos. Se intentaron realizar las pruebas, pero se obtuvo lo siguiente:

```
C:\Users\kioki\Documents\GitHub\LyricGenerator\src\Neural Networks\bert>python -c "import tensorflow as tf; print(tf.__version__)"  
2021-09-21 16:59:48.692996: W tensorflow/stream_executor/platform/default/dso_loader.cc:55] Could not load dynamic library 'cudart64_100.dll'  
; dlerror: cudart64_100.dll not found  
2021-09-21 16:59:48.693481: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up  
on your machine.  
1.15.0
```

Figura 3.3: Error tensorflow BERT

En la imagen anterior se observa que tensorflow requiere del archivo cudart64_100.dll disponible solo en tarjetas de video de Nvidia y siendo más específicos en las llamadas CUDA GPUs las cuales son aquellas tarjetas de video de la familia RTX o Quadro y que para trabajar con CUDA (Arquitectura Unificada de Dispositivos de Cómputo) requiere de un gran ancho de banda en la memoria de las tarjetas y los dispositivos de la familias antes mencionadas son los que cuentan con la arquitectura y la capacidad de memoria para realizar estas tareas.

3.6. Estudio de factibilidad

El estudio de factibilidad es un instrumento que sirve para orientar la toma de decisiones, así como para determinar la posibilidad de desarrollar un negocio o proyecto; corresponde a la última fase de la etapa pre-operativa del ciclo del proyecto. Se formula con base en información que tiene la menor incertidumbre posible para medir las posibilidades de éxito o fracaso de un proyecto, apoyándose en el resultado se tomará la decisión de proceder o no con su implementación.

Este estudio establecerá la viabilidad, si existe, del trabajo planteado previamente.

- **Factibilidad Técnica:** Este aspecto evalúa que la infraestructura, es decir, equipos, software, conocimiento, experiencia, etc. que se poseen son los necesarios para efectuar las actividades requeridas para la realización del trabajo terminal.

- **Factibilidad Operativa:** Analiza si el personal posee las competencias necesarias para el desarrollo del proyecto.
- **Factibilidad Económica:** Consiste en el análisis de los recursos financieros necesarios para llevar a cabo la elaboración de este proyecto.

3.6.1. Factibilidad técnica

Dentro de este apartado se explican detalladamente las tecnologías que se utilizarán, así como las características de nuestros equipos de cómputo actualmente. La elección de estas herramientas estuvo basada tanto en las tecnologías que más se utilizan en la actualidad como las que cuentan con el mayor soporte para su trabajo en la nube, esto se debe a que los equipos actuales no soportarían todo el trabajo.

Cuadro 3.4: Equipo de cómputo ideal

Equipo de cómputo ideal.	
Procesador	Intel i9 10900KF
Tarjeta de video	NVIDIA RTX 3080 10Gb
Memoria RAM	32Gb DDR4
Disco duro	2Tb HDD y 512Gb SSD

En la tabla anterior se muestra el equipo que requeriríamos para poder trabajar sin mayor dificultad al momento de realizar los entrenamientos de los modelos de redes neuronales que necesitamos para la realización de este trabajo, debido a la situación actual de alzas en precios muy altas, en el mercado actual los componentes para el equipo son escasos y su precio en el mercado aumentó. Por esa razón se utilizarán servicios de la nube para la realización de estos entrenamientos y nuestros equipos para el desarrollo del resto del proyecto.

Cuadro 3.5: Herramientas de software a utilizar

Herramientas de Software a utilizar	
Sistema Operativo	Linux, Mac, Windows
Navegador Web	Google Chrome
Lenguaje de Programación	Python
Servidor	Apache y Gunicorn
Servicio Nube	Amazon Web Services

Además de las herramientas de software a utilizar, es necesario mencionar el equipo de hardware que se utiliza, tanto para desarrollar, como para probar e implementar cada uno de los prototipos a lo largo de este trabajo terminal.

Cuadro 3.6: Equipo de cómputo 1

Equipo de cómputo 1 utilizado.	
Procesador	Ryzen 5 3600
Tarjeta de video	Amd Radeon Rx580
Memoria RAM	32 Gb
Disco duro	1Tb HDD y 512Gb SSD

Cuadro 3.7: Equipo de cómputo 2

Equipo de cómputo 2 utilizado.	
Marca	Apple
Modelo	iMac Late 2012
Procesador	Intel Core i5
Tarjeta de video	NVIDIA GeForce GT 640M 512 Mb
Memoria RAM	8 Gb
Disco duro	1Tb y 256 Gb SSD

Cuadro 3.8: Equipo de cómputo 3

Equipo de cómputo 3 utilizado.	
Procesador	Amd FX-8350
Tarjeta de video	Nvidia Geforce 1050ti
Memoria RAM	16 Gb
Disco duro	1Tb HDD

Junto con las herramientas de hardware y software a utilizar es necesario mencionar los servicios básicos que son relevantes para el desarrollo de este trabajo terminal como lo son:

- Luz Eléctrica
- Agua Potable
- Internet

Estos servicios forman parte de la factibilidad técnica ya que sin ellos no se podría realizar este proyecto y por eso mismo generan un costo, este mismo se desglosa más adelante en la sección 3.6.3.

3.6.2. Factibilidad operativa

A continuación se presenta una tabla con los recursos operativos del trabajo terminal que se calcularon con base en los recursos humanos con los que se cuenta actualmente y un análisis de las horas en las que el personal estará en operación:

Cuadro 3.9: Relación de horas de trabajo estimadas para la realización de este trabajo terminal

Horas a trabajar en el desarrollo del trabajo terminal					
Mes	No. de Días	Sábado y Domingo	Días hábiles	Horas de trabajo por día	Horas Totales
Marzo	31	8	22	2	44
Abril	30	8	15	2	30
Mayo	31	10	19	2	38
Junio	30	10	17	2	34
Agosto	31	10	18	2	24
Septiembre	30	8	20	2	40
Octubre	31	10	20	2	40
Noviembre	31	8	18	2	36

Con esto podemos concluir que contamos con 286 horas, suficiente tiempo para el desarrollo de este trabajo terminal, ya que las horas totales de trabajo están contempladas para cada uno de los integrantes del equipo

3.6.3. Factibilidad económica

Luego de haber realizado el estudio de factibilidad técnica así como el operacional es necesario tomar en cuenta un estudio de factibilidad económica el cual desglosará todo el gasto económico realizado para la elaboración de este trabajo terminal:

- **Capital Humano:** Se tienen contemplados aproximadamente 36 días laborales, es decir, 288 horas para la elaboración de este trabajo terminal en el cual participaremos los tres integrantes
- **Capital Técnico:** Se cuenta con las viviendas y el equipo de cómputo principal de cada uno de los integrantes.

Respecto a los costos monetarios de todo el proyecto se toma a consideración lo siguiente:

- **Servicios**

Se considera un gasto mensual aproximado de \$1,400.00 que al ser multiplicado por todo el tiempo de elaboración contemplado nos da un total de \$11,200.00, por integrante.

- **Software**

Durante algunos periodos se va a hacer uso principalmente de herramientas gratuitas y de software libre, en cuanto al servicio en la nube se pretende hacer uso de AWS y trabajar inicialmente con los planes gratuitos que ofrece, en caso de que se lleguen a consumir los recursos de este plan, entonces se procederá a hacer el cambio a otro plan superior donde el costo actual se cobra en \$ 0.15 centavos de dólar por hora de uso.

- **Hardware**

Se utilizarán los equipos de cómputo personal de cada integrante, lo que da un costo aproximado de \$24,012.00 aplicando los parámetros de vida útil de un equipo de cómputo y su depreciación anual del hardware.

- **Recursos Humanos**

Se estima un gasto aproximado de \$12,000.00 por cada integrante del equipo para la elaboración del proyecto con lo que se generará un gasto total de \$36,000.00

Tomando en cuenta el salario promedio de un desarrollador de software el cual es de \$15,000.00 mensuales, dato fundamentado en la búsqueda de salario promedio en la página web de mx.indeed.com, se consideró este salario

para el desarrollo del proyecto, el cual tendría una duración de 8 meses aproximadamente para finalizar este proyecto, dando como resultado un salario total para los integrantes del equipo de \$360,000.00; el costo final del desarrollo de este trabajo terminal sería de:

\$453,612.00

Conclusión

Tras realizar el estudio de factibilidad en este proyecto, es pertinente decir que los integrantes no cuentan con el apoyo financiero y que el hardware mencionado ya es propiedad de los integrantes, por lo que el trabajo terminal se califica como “*Viable*” iniciando de manera inmediata su implementación acorde las fechas mencionadas.

3.6.4. Aspectos económicos

Las herramientas que usaremos en el trabajo terminal y las cuales validarán la viabilidad del proyecto se presentan a continuación:

Herramientas Open Source a utilizar en el presente trabajo terminal

- Datamuse API [61]
- MongoDB - SSPL* [62]
- BERT - Apache 2.0 [63]
- Apache - Apache 2.0 [64]
- React - MIT [65]
- Python - PSFL [66]
- Amazon - Elastic 2.0 y SSPL [67]

*SSPL: Aunque es clasificado como Open Source por parte de MongoDB, al estar basado en la licencia de software libre de GPL3, la Iniciativa de Open Source (OSI) no la reconoce como licencia Open Source al describirla como una “licencia Open Source no genuina”.

Como se puede apreciar en el software presentado, las licencias son del tipo Open Source con lo cual validamos la viabilidad económica del proyecto

el cual no representará gasto monetario más allá del necesitado para el entrenamiento de la red neuronal, para el cual haremos uso de Amazon SageMaker que tiene un costo de:

Cuota gratuita primeros 2 meses:

- Instancia de bloc de notas: 250 horas.
- Entrenamiento: 50 horas.

Cuotas:

- Instancia de bloc de notas (4 CPU's virtuales y 16Gb de memoria): 0.20 centavos de dólar/hora.
- Entrenamiento (4 CPU's virtuales y 16Gb de memoria): 0.23 centavos de dólar/hora.
- Procesamiento (4 CPU's virtuales y 16Gb de memoria): 0.20 centavos de dólar/hora.

3.6.5. Aspectos legales

Copyright

Los derechos de autor son una forma de protección para obras originales de autoría fijadas en un medio de expresión tangible. Los derechos de autor cubren tanto las obras publicadas como las que no o son inéditas.[53]

Los derechos de autor protegen un gran rango de obras como lo son: obras literarias, musicales, pictóricas o escultóricas, coreografías, escenas dramáticas, producciones cinematográficas y demás audiovisuales, programas de cómputo, fotografías, entre muchas otras.

Originalidad

Los derechos de autor protegen la forma en que se expresan las ideas. Esta manera única de expresar las palabras, notas musicales, colores, formas, etc. son elegidas y organizadas u ordenadas. Es la expresión lo que hace que una obra sea original. Esto significa que puede haber muchos trabajos diferentes sobre la misma idea y todos de ellos estarán protegidos por derechos de autor, siempre y cuando expresen esta idea de una manera original.[54]

Según la Ley Federal del Derecho de Autor, la forma en cómo se expresa

un autor al momento de crear su obra es lo que esta protegido por la ley ante acciones fraudulentas.[55]

Con base en la Ley Federal del Derecho de Autor Capitulo IV articulo 107 “Las bases de datos o de otros materiales legibles por medio de máquinas o en otra forma, que por razones de selección y disposición de su contenido constituyan creaciones intelectuales, quedarán protegidas como compilaciones.”[55]

Por ende las canciones generadas por nuestro sistema no se pretenden ni pueden registrar, ya que estos resultados generados serian nuestros.

3.7. Análisis de riesgo

Dentro de este apartado determinaremos la probabilidad de que surja alguna problemática o riesgo el cual pueda impactar el desarrollo del proyecto, ya sea en términos del cronograma, la calidad o los costos.

Algunos riesgos que enfrentaremos durante el desarrollo del proyecto son los siguientes:

- **Riesgo de costo**

Si sobrepasamos el costo de desarrollo previsto, quizá nos enfrentemos a problemas relacionados con el alcance y los requerimientos tanto funcionales como los que no lo son del proyecto.

Cuadro 3.10: Riesgo de costo

Riesgo: Costos Elevados	
Impacto: Medio alto (7).	Nivel: Alto (49)
Probabilidad: Medio alto (7)	

Costos Elevados



Figura 3.4: Matriz de riesgo de costo

■ Riesgo de calendario

Si se realizó una mala estimación del tiempo necesario para el desarrollo, se asignaron mal los recursos o si hubo alguna afectación dentro del recurso humano, lo más probable es que el proyecto no se termine de desarrollar en el tiempo establecido.

Cuadro 3.11: Riesgo de calendario

Riesgo: Falta de Tiempo	
Impacto: Medio bajo (4).	Nivel: Medio Medio (16)
Probabilidad: Medio bajo (4)	

Falta de tiempo



Figura 3.5: Matriz de riesgo de calendario

■ Riesgo tecnológico

Si no logramos entender la complejidad de las herramientas que utilizaremos para el desarrollo del proyecto o nos toma demasiado tiempo que son nuevas para nosotros, como lo son la herramientas de AWS para la creación de los modelos de redes neuronales y su entrenamiento dentro de la misma plataforma, se podrían presentar problemas como: integración a la página web, creación fallida de los modelos de las redes neuronales, falta de adaptación con el hardware utilizado, entre otros.

Cuadro 3.12: Riesgo tecnológico

Riesgo: Fallas de Equipos	
Impacto: Bajo (1).	Nivel: Bajo (7)
Probabilidad: Medio alto (7)	

Fallas de equipos



Figura 3.6: Matriz de riesgo tecnológico

■ Riesgo operacional

Si no se plantean posibles soluciones a problemas que podrían suceder durante el proyecto, si no se presenta un correcto liderazgo y buenas aportaciones por parte de uno o más miembros, si existe una mala comunicación, falta de motivación y si no hay una monitorización, se puede ver mermado el avance del proyecto.

Cuadro 3.13: Riesgo operacional

Riesgo: Desorganización en el equipo	
Impacto: Medio bajo (4).	Nivel: Medio (16)
Probabilidad: Medio bajo (4)	

Desorganización en el equipo



Figura 3.7: Matriz de riesgo operacional

■ Riesgos externos

Factores fuera de nuestro alcance que pueden afectar el desarrollo del proyecto, entre ellos puede ser algún cambio en las leyes, en alguna norma, desastre natural, etc.

Cuadro 3.14: Riesgos externos

Riesgo: Factores Externos	
Impacto: Alto (10).	Nivel: Bajo (10)
Probabilidad: Bajo (1)	

Factores externos

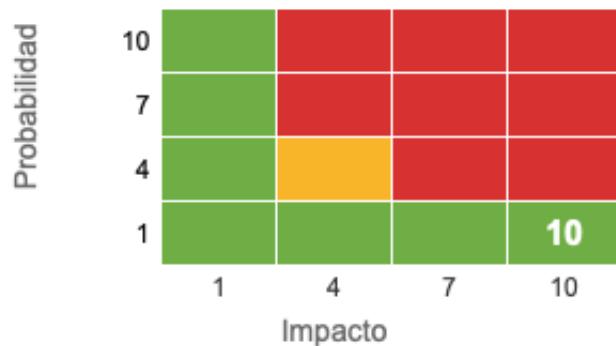


Figura 3.8: Matriz de riesgos externos

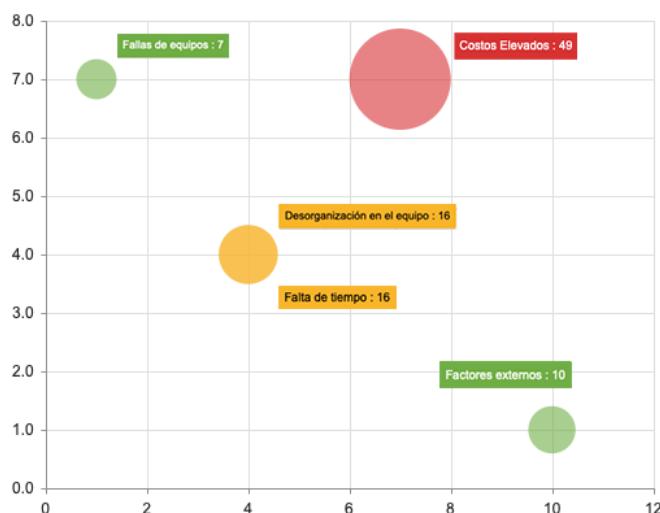


Figura 3.9: Matriz general de riesgos

Se utilizó la herramienta llamada “Matriz de Evaluación de Riesgos de ITM Platform” para generar este análisis de riesgo. [71]

Capítulo 4

Diseño

4.1. Arquitectura del sistema

4.1.1. Descripción de la arquitectura del sistema

Para poder crear esta herramienta de apoyo para estudiantes o aficionados en el área musical con la creación de letras musicales, se propone crear un sistema basado en inteligencia artificial usando técnicas de procesamiento de lenguaje natural que pueda auxiliarlos y puedan lograr su meta propuesta.

El sistema se compone de tres bloques los cuales se comunicarán vía red:

1. **Base de datos:** En este módulo se almacenan los datos de canciones, se trabajará especialmente con una tabla que contenga como principales campos:

- Artista
- Género
- Nombre de la Canción
- Letra
- Idioma

Dicha base de datos se almacenará en un sistema especializado que se conecte a la nube para poder trabajar y almacenar estos datos sin mayor problema.

2. **Modelado de la red neuronal:** Este bloque va ser el encargado de generar las letras musicales para cada usuario que ingrese. Para la generación de dichas letras musicales, se utilizará un servicio en la nube para procesar la solicitud del usuario.

3. **Página Web:** Este primer bloque es el que se encuentra directamente enlazado con el usuario de nuestro sistema y al mismo tiempo con el servicio en la nube que aloja a las redes neuronales, el usuario utilizando su navegador podrá realizar peticiones al servicio en la web.

4.1.2. Funcionamiento general del sistema

El siguiente diagrama muestra como se espera que quede el funcionamiento general del proyecto.

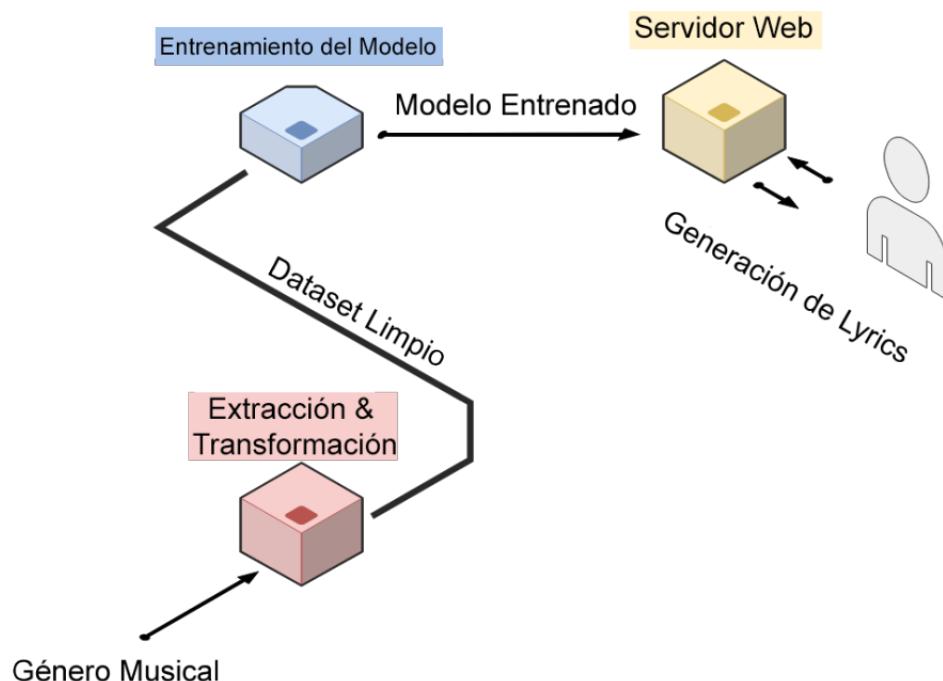


Figura 4.1: Diagrama general del sistema

Como se observa en la figura anterior, se tendrán tres procesos principales para lograr el objetivo final.

En primer instancia se tendrá que obtener la base de datos y por medio de extracción y transformación, se obtendrá el dataset deseado para poder generar el modelo.

Después de un proceso de limpieza de este dataset, se hará un entrenamiento del modelo en la nube para poder generar funciones utilizando este modelo para poder generar textos. una vez teniendo esta aplicación funcional, se

configurará el servidor web para que se pueda comunicar con el usuario final y con la aplicación que utiliza el modelo vía web, logrando así los objetivos.

4.1.3. Diagrama caso de uso

A continuación mostramos el caso de uso de nuestra aplicación web y su interacción con el usuario:

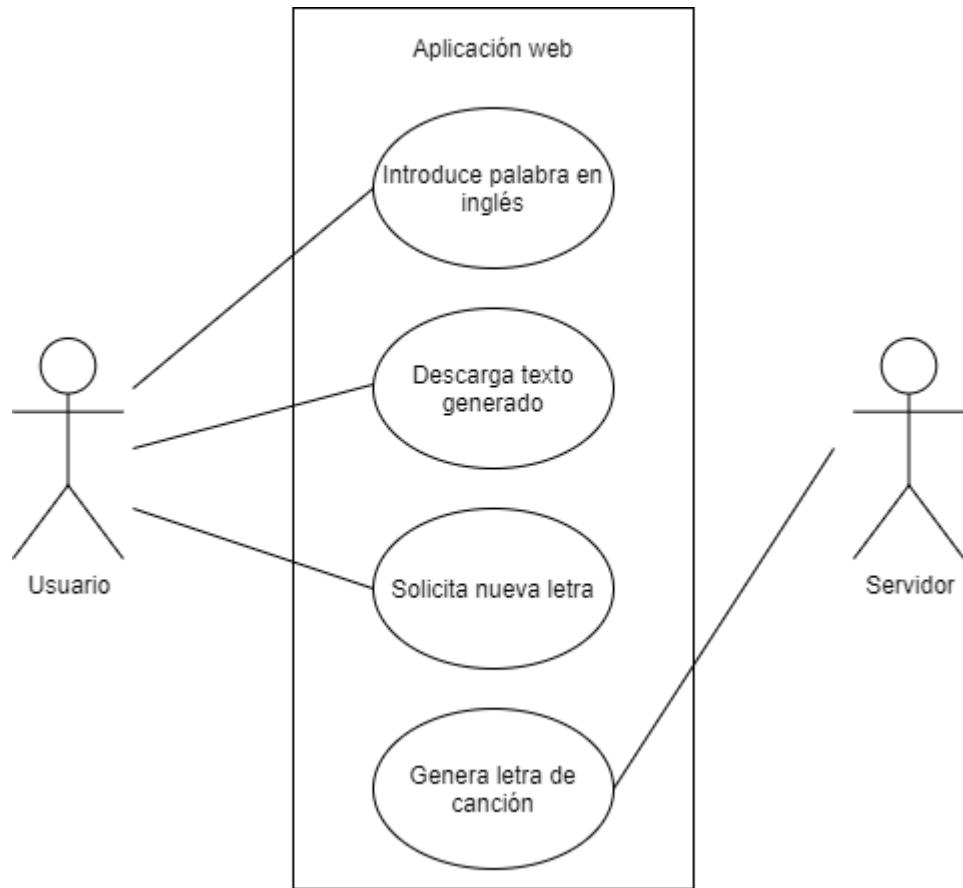


Figura 4.2: Diagrama de caso de uso

Cuadro 4.1: Caso de uso 1

Caso de uso	Introduce palabra en inglés
Actores	Usuario
Descripción	El usuario dentro de la aplicación web proporciona una palabra en el idioma inglés para con ella poder generar un texto

Cuadro 4.2: Caso de uso 2

Caso de uso	Descargar texto generado
Actores	Usuario
Descripción	El usuario dentro de la aplicación web tiene la posibilidad de descargar el texto generado por el modelo

Cuadro 4.3: Caso de uso 3

Caso de uso	Solicitar nueva letra
Actores	Usuario
Descripción	Al usuario dentro de la aplicación web se le da posibilidad de volver a generar otra letra musical usando los mismos parámetros que proporcionó o utilizando unos nuevos

Cuadro 4.4: Caso de uso 4

Caso de uso	Genera letra de canción
Actores	Servidor
Descripción	El servidor envia el texto generado por el modelo a la aplicación web, la cual se encarga de mostrarla al usuario final

4.1.4. Historias de usuario

Cuadro 4.5: Historia de usuario 1

Historia de usuario 1	
Nombre:	Usuario:
Generar una letra de canción.	Visitante

Descripción: Como un visitante de la página web quiero generar una letra de una canción a partir de una palabra que proporcione.

Validación: Con la palabra proporcionada por el usuario y con ayuda del modelo entrenado se generará una letra musical relacionada a esta.

Cuadro 4.6: Historia de usuario 2

Historia de usuario 2	
Nombre:	Usuario:
Descargar una letra canción.	Visitante

Descripción: Como un visitante de la página web quiero poder descargar la letra de la canción que se generó a partir de la palabra que proporcioné.

Validación: El usuario podrá descargar un archivo de texto que contendrá la letra de la canción generada.

Cuadro 4.7: Historia de usuario 3

Historia de usuario 3	
Nombre:	Usuario:
Generar otra letra de canción con los mismos parámetros	Visitante

Descripción: Como usuario, me gustaría generar otra letra de canción con los mismo parámetros y palabra que ya había introducido.

Validación: Brindar una opción al usuario que una vez generada y desplegada la letra de la canción pueda generar otra usando los mismo parámetros que previamente había introducido.

Cuadro 4.8: Historia de usuario 4

Historia de usuario 4	
Nombre:	Usuario:
Validación de la palabra introducida.	Sistema

Descripción: La página web validará la palabra proporcionada por el usuario para con ella hacer la petición al servidor y generar una letra de canción relacionada a esta.

Validación: Con ayuda del modelo y la palabra introducida y procesada por la página web se genera y despliega la letra generada.

Cuadro 4.9: Historia de usuario 5

Historia de usuario 5	
Nombre: Limpiar base de batos.	Usuario: Desarrollador
Descripción: Como un desarrollador quiero una base de datos sin caracteres especiales o palabras repetidas.	Validación: Con ayuda de un script para quitar palabras innecesarias y caracteres inválidos

Cuadro 4.10: Historia de usuario 6

Historia de usuario 6	
Nombre: Pre-procesar la base de datos.	Usuario: Desarrollador
Descripción: Como un desarrollador quiero generar el modelo solo con los datos necesarios para optimizar los tiempos de entrenamiento.	Validación: Con un script para tokenizar y hacer pre-padding en la base de datos

Cuadro 4.11: Historia de usuario 7

Historia de usuario 7	
Nombre: Subir Base de Datos a la nube.	Usuario: Desarrollador
Descripción: Como un desarrollador quiero generar el modelo con los datos necesarios en la nube para optimizar espacio local.	Validación: Utilizando herramientas NOSQL para subir la base de datos y esta no ocupe espacio local

Cuadro 4.12: Historia de usuario 8

Historia de usuario 8	
Nombre: Generar Modelo en la nube.	Usuario: Desarrollador
Descripción: Como un desarrollador quiero generar nuevas letras de canciones a partir de un dataset proporcionado anteriormente.	Validación: Con un script en una plataforma en línea entrenar el modelo para generar nuevas canciones

Cuadro 4.13: Historia de usuario 9

Historia de usuario 9	
Nombre: Generar Modelo en la nube.	Usuario: Desarrollador
Descripción: Como un desarrollador quiero generar nuevas letras de canciones a partir de un dataset limpio proporcionado anteriormente.	Validación: Con un script en una plataforma en línea entrenar el modelo para generar nuevas canciones

Cuadro 4.14: Historia de usuario 10

Historia de usuario 10	
Nombre: Probar respuestas del modelo en la nube.	Usuario: Desarrollador
Descripción: Como un desarrollador quiero generar nuevas letras de canciones a partir del modelo generado en la nube	Validación: Con una función que genere algunos ejemplos usando el modelo generado en la nube

Cuadro 4.15: Historia de usuario 11

Historia de usuario 11	
Nombre: Desarrollar la vista del usuario (Página Web).	Usuario: Desarrollador
Descripción: Como un desarrollador quiero que el usuario tenga una vista intuitiva y fácil de utilizar con la que pueda generar sus lyrics apropiadamente.	Validación: Creando la página web con vistas sencillas y ordenadas para que el usuario no se pierda en la navegación en ella.

Cuadro 4.16: Historia de usuario 12

Historia de usuario 12	
Nombre: Conectar la vista del usuario con el modelo.	Usuario: Desarrollador
Descripción: Como un desarrollador quiero que la vista del usuario y el modelo funcionen en conjunto para que se pueda generar la lyric de la canción y se vea apropiadamente.	Validación: Conectando la página web con el modelo generado y las funciones para generar una lyric nueva

4.1.5. Requerimientos funcionales y no funcionales

A continuación, se presentan los requerimientos funcionales mas importantes de nuestro proyecto:

- RF1: Permitirá a cualquier usuario que acceda a la página, generar uno o más versos musicales.
- RF2: Permitirá al usuario descargar la letra generada como archivo de texto.
- RF3: Permitirá al usuario generar cuantas veces quiera una letra musical.
- RF4: Solo se generarán las letras musicales si se cumple con el requisito obligatorio, proporcionado por el usuario, el cual será: una palabra en el idioma inglés.

Ahora, se presentan algunos de los requerimientos no funcionales más importantes de nuestro proyecto

- RNF1: El servicio web debe ser capaz de procesar más de diez solicitudes simultaneas.

- RNF2: La página debe ser capaz de operar adecuadamente con hasta cincuenta usuarios simultáneamente.
- RNF3: El tiempo que le toma a un usuario promedio en el aprendizaje del sistema de la interfaz deberá de ser menor a cinco minutos.
- RNF4: La página proporcionará alertas de error que sean informativos y que ayuden a la experiencia de usuario.
- RNF5: La aplicación web debe tener un diseño responsivo, es decir, que garantice una adecuada visualización tanto en computadoras, tabletas y dispositivos móviles.

4.2. Base de datos

4.2.1. Descripción

En este apartado se hará el diseño de la base de datos para poder generar un modelo acorde a los requerimientos funcionales del proyecto.

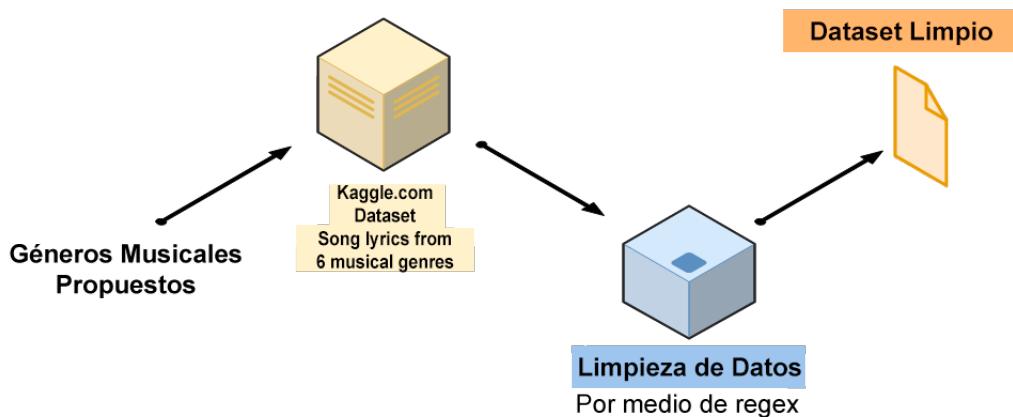


Figura 4.3: Diagrama de obtención de dataset

4.2.2. Obtención de la base de datos

Para iniciar la búsqueda de la base de datos primero se eligieron las características esenciales que se muestran a continuación para poder generar el modelo:

- Letra de la canción.
- Idioma.
- Artista.

Se hizo la búsqueda de datasets con estas características en páginas que contienen datasets con licencia del tipo open source, tales como Kaggle [72],

Google datasets [74], AWS Open Data [75] y en papers de investigación.

Encontramos diferentes datasets que contaban con algunas de las necesidades del sistema pero por una u otra razón no se podían utilizar, tal fue el caso del dataset de Music4All [76] que contaba con las necesidades del sistema pero el uso de los datos era muy restringido.

En Kaggle [72] se encontró un dataset llamado “Song lyrics for 6 musical genres” [73] el cual contiene todos los datos necesarios para el sistema con un total de 160,790 letras de canciones con las tablas mostradas a continuación:

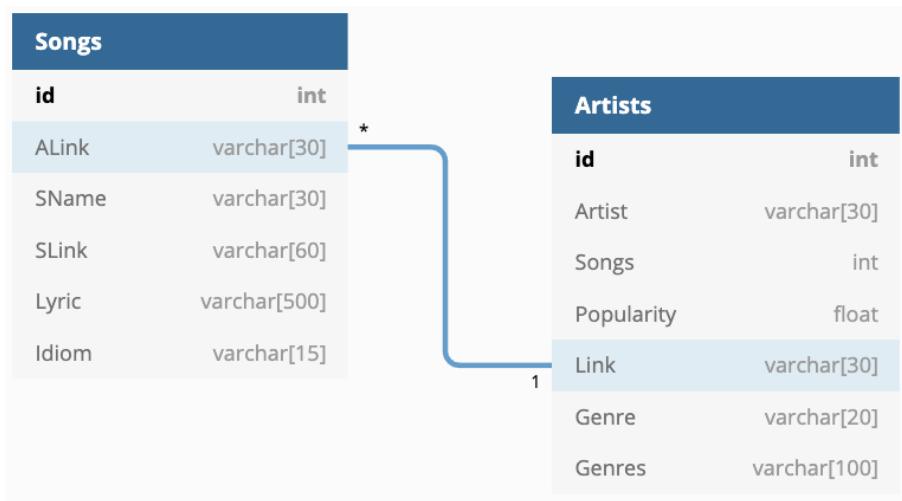


Figura 4.4: Diagrama entidad-relación de la base de datos

Dicha base de datos se sacó originalmente de la página de vagalume.com [77] e incluye 6 tipos de géneros musicales mencionados a continuación, los cuales son:

- Rock
- Pop
- Hip Hop
- Samba
- Sertanejo

- Funk Carioca

Se cuenta con dos tablas que cumplen con las características ideales para realizar el modelo acorde a los requerimientos funcionales.

Como se puede observar, se tienen las columnas “Genre” que utilizaremos para separar el dataset en géneros musicales y “ALink” que servirá como enlace a la segunda tabla.

Se utilizarán las columnas de “Lyric” para poder entrenar el modelo, “Idiom” para seleccionar y utilizar las canciones en el idioma inglés y “ALink” que nos servirá para fusionar la tabla mencionada anteriormente con la actual y así poder usar esas columnas, quedando un resultado como en la siguiente figura:

Songs_with_Artists	
id	int
ALink	varchar[30]
SName	varchar[30]
SLink	varchar[60]
Lyric	varchar[500]
Idiom	varchar[15]
Artist	varchar[30]
Popularity	float
Link	varchar[30]
Genre	varchar[20]
Genres	varchar[100]

Figura 4.5: Diagrama entidad-relación de la base de datos procesada

4.2.3. Género musical

El género musical que elegimos es Pop en inglés debido a que es un género musical flexible en su estructura y en su léxico, otros géneros contemplados fueron Hip Hop y Rock que también cuentan con un extenso catálogo de datos pero se encontraron modelos que ya utilizan esos géneros.

4.2.4. Resultados

Al necesitar extraer información precisa para el entrenamiento del modelo no es necesario que la base de datos sea muy compleja, razón por la cual las tablas mostradas en el diagrama entidad-relación se ve así, siendo lo de mayor relevancia el mostrar que un artista puede tener muchas canciones pero una canción le pertenece únicamente a un artista.

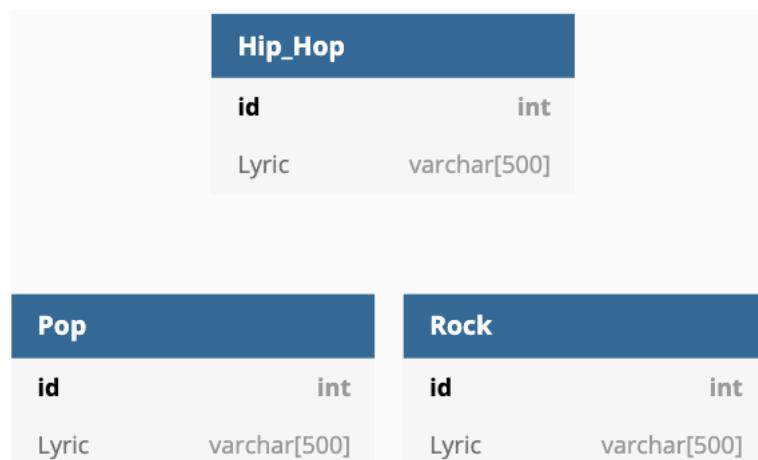


Figura 4.6: Diagrama entidad-relación final

4.3. Limpieza de la base de datos

4.3.1. Descripción

Una vez obtenida la base de datos, es necesario hacer una inspección de esta misma para verificar que los estos datos sean útiles para proceder a generar un modelo, este paso resulta de suma importancia, ya que de esto depende que la precisión de los resultados del modelo.

Para ello se identificarán datos que resulten incompletos, incorrectos, inexacos, no pertinentes para luego sustituirlos, modificarlos o en su caso, eliminar estos datos.

Una vez realizada esta limpieza, la base de datos será compatible con el entrenamiento y los resultados generados al final serán más entendibles.

Estas inconsistencias en el conjunto de datos pueden ser causados por errores de entradas del usuario, corrupción de los datos, errores en el área de scraping, o simplemente tienen caracteres que no nos interesa procesar.

Se utilizará la librería de “Pandas” y en caso de que se requiera, utilizar la librería de expresiones regulares para hacer esta limpieza, estás dos librerías de Python ayudarán a que los datos puedan ser utilizados para realizar el debido preprocesamiento como siguiente paso sin tener que cambiar cada campo manualmente, lo que mejora la eficiencia para poder lograr el objetivo propuesto.

Para limpiar los datos seguiremos el estándar de limpieza de datos:[56]

1. Remover caracteres innecesarios
2. Eliminar Duplicados
3. Evitar errores ortográficos de similitud
4. Convertir los tipos de dato
5. Tratar los valores nulos o faltantes

4.3.2. Remover caracteres innecesarios

Es necesario remover este tipo de elementos del conjunto de datos para que al momento de entrenar el modelo, no arroje resultados con valores ilegibles o incluso, que no se pueda tratar la base de datos correctamente.

Para este paso, es necesario tener conocimiento de expresiones regulares básicas para poder delimitar los espacios dobles, letras o caracteres especiales ilegibles, saltos de línea, tabuladores, entre otros que puedan causar errores en los siguientes pasos.

4.3.3. Eliminar duplicados

Pueden existir datos duplicados dentro de la base de datos que para el entrenamiento sería inútil además que solicitaría recursos innecesarios al sistema el meter más datos de los necesarios, para ello, con la función drop.duplicates() de pandas, se podrá lograr el acometido sin problema alguno.

4.3.4. Evitar errores ortográficos o de similitud

Existen librerías en el idioma inglés para corregir la ortografía, en este proyecto se quiere generar la letra apegada a la jerga utilizada en el idioma para poder generar las nuevas letras de canciones, por lo que no será necesario hacer a profundidad este paso.

4.3.5. Convertir los tipos de dato

Se va a corroborar que todas las celdas contengan únicamente el tipo varchar con un límite de 500 caracteres, número elegido al tener una media de todos los elementos de 350 caracteres aproximadamente. El número de filas que contienen más de 500 caracteres es muy bajo por lo que al cortar las canciones que tienen más puede ayudar a ahorrar la memoria utilizada a la hora del entrenamiento.

4.3.6. Tratar los valores nulos o faltantes

Tener datos nulos en el entrenamiento puede llegar a ser fatal debido a que además de utilizar demasiada memoria, a la hora de generar resultados, estos resultan en nulos, además que tratar la misma base de datos a la hora del preprocesamiento puede arrojarnos errores a la hora de manipular la base de datos.

4.3.7. Resultados

Una vez realizada esta limpieza general del conjunto de datos, el resultado esperado es tener una base de datos útil y solo con los datos necesarios para poder tratarlos con el debido proceso para empezar el preprocesamiento de entrenamiento del modelo de la red neuronal, este proceso es necesario hacer solo una sola vez para cada género musical ingresado, en caso de que se lleguen a necesitar más datos, será necesario repetir el proceso para poder generar resultados de la manera más óptima posible. A continuación se muestran las primeras líneas de la base de datos limpias.

Lyric	
0	Hey, slow it down. What do you want from me. W...
1	Died last night in my dreams. Walking the stre...
2	I've paid my dues. Time after time. I've done ...
3	So I got my boots on,. Got the right amount of...
4	I wish that this night would never be over. Th...

Figura 4.7: Resultados de limpieza en la base de batos

4.4. Redes neuronales a utilizar

4.4.1. Descripción

En este apartado expondremos las herramientas pensadas para el desarrollo del modelo de red neuronal, así como la manera en que podrían integrar a nuestro proyecto y las conexiones que puede tener tanto con la base de datos como con el servidor en la nube o la interfaz web.

4.4.2. Redes neuronales para generar el modelo

Se decidió trabajar con un modelo generado por nosotros usando como base redes neuronales de tipo LSTM bidireccionales debido a los problemas encontrados al tratar de usar BERT, este tipo de redes se pueden entrenar e ir ajustándolas hasta obtener resultados que consideremos satisfactorios. Al contar con la característica de ser bidireccional esta puede analizar el texto anterior para poder generar el texto futuro.

4.4.3. Modelo

Para el diseño del modelo se tiene contemplado el siguiente modelo del cual se puede decir que una vez se tiene la salida del transformer se pasa por una capa de clasificación, posterior a ello se usa la función “Softmax” de Python para calcular la probabilidad de aparición de cada palabra que se tiene en el vocabulario y cuyo resultado final determinará la palabra usada para la máscara (marcada de color rojo).

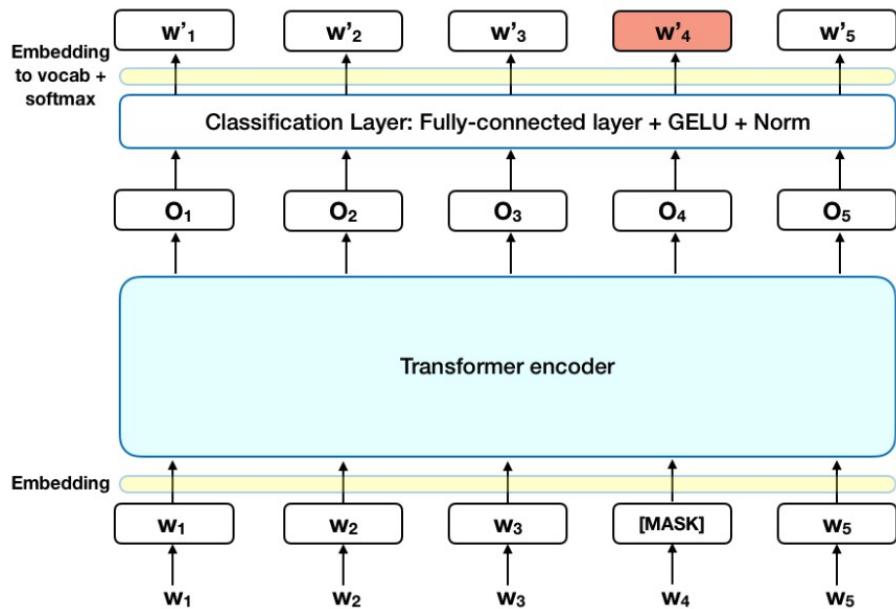


Figura 4.8: Diagrama de diseño del modelo

4.4.4. Resultados

Habiendo hecho el análisis del modelo anterior y tomándolo como referencia para nuestro proyecto podemos adaptarla de forma tal que las entradas sean en su mayoría [MASK] las cuales serán las palabras predichas siendo la razón por la cual todas las derivadas de ésto están en rojo ya que representan una entradas del tipo MASK.

Siendo el resultado esperado uno muy similar a la estructura que se muestra a continuación y en la que se puede ver lo siguiente:

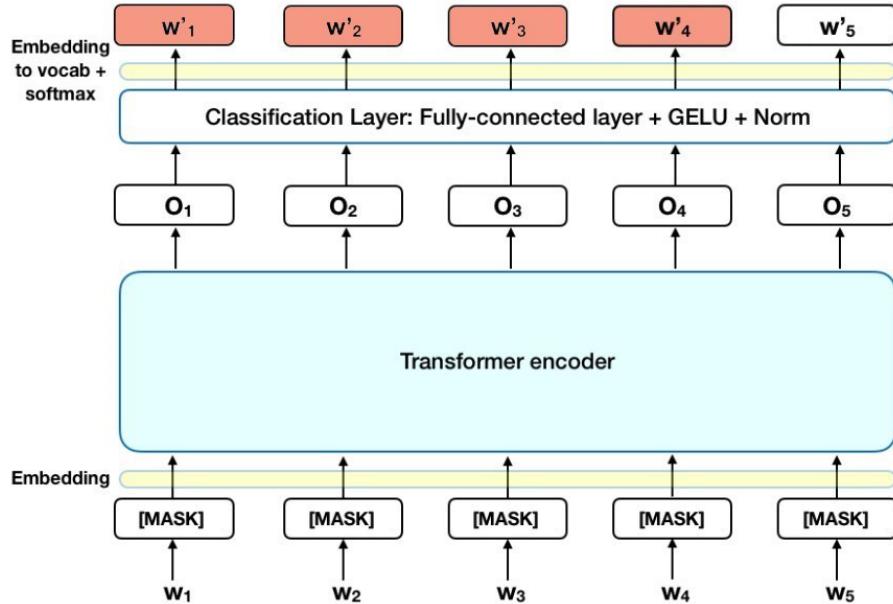


Figura 4.9: Diagrama de diseño final del modelo

La estructura de las letras musicales generadas se planea que sean de la siguiente forma:

Verso

- Estará compuesto de 4 a 6 oraciones y el valor será decidido por un número al azar.
- Las palabras que serán predichas en todo el verso serán de 4 a 5 y el valor será decidido por un número al azar.

Pre-coros y puente

- Estará compuesto de 2 a 3 oraciones y el valor será decidido por un número al azar.
- Las palabras que serán predichas en todo el verso serán de 6 a 10 y el valor será decidido por un número al azar.

Coro

- Estará compuesto de 3 a 4 oraciones y el valor será decidido por un número al azar.

- Las palabras que serán predichas en todo el verso serán de 3 a 8 y el valor será decidido por un número al azar.

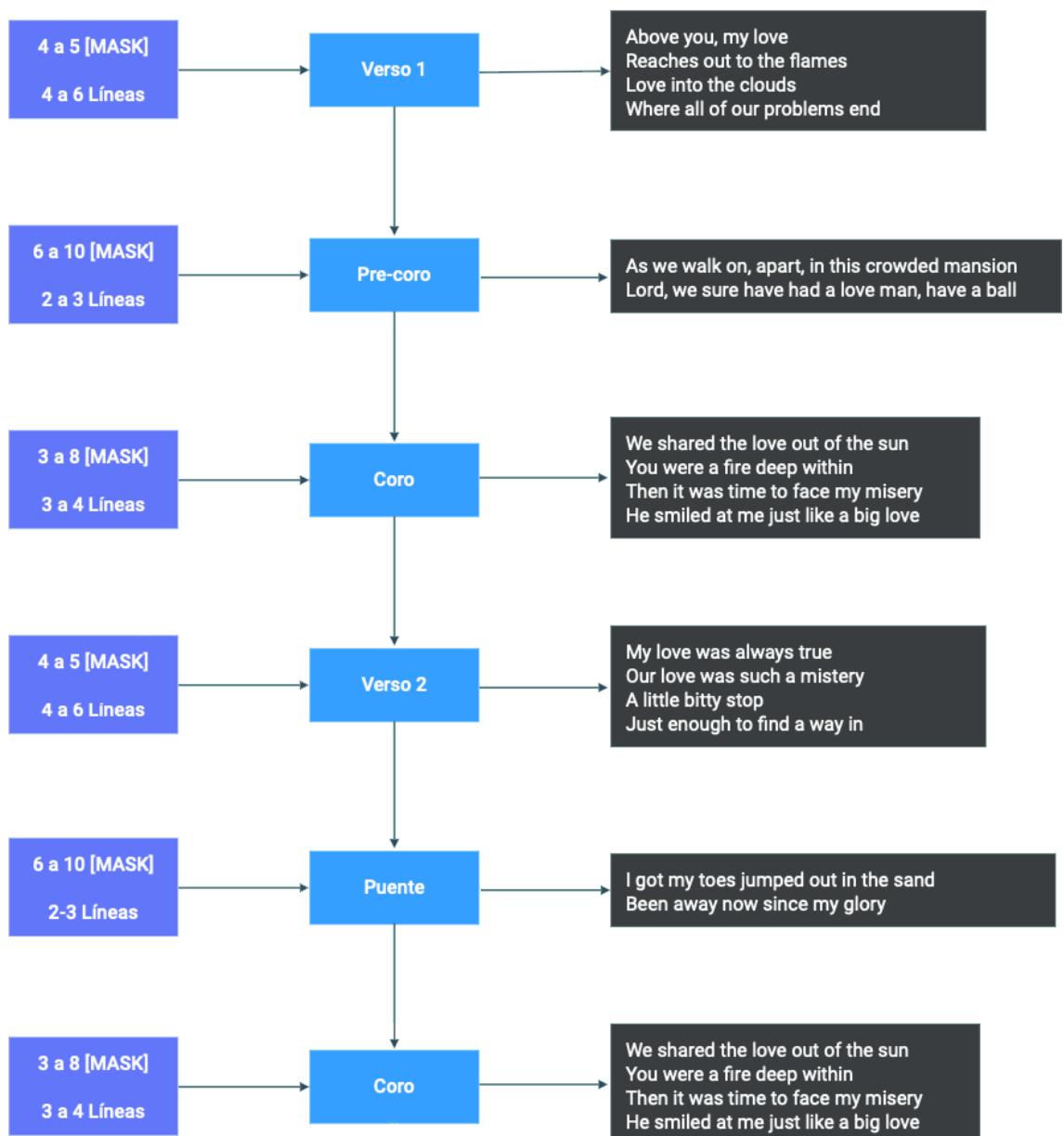


Figura 4.10: Diseño de la estructura del modelo

4.5. Generación del modelo en la nube

4.5.1. Descripción

En este apartado se hablará de las herramientas que utilizaremos, la interacción entre el desarrollador y el sistema para suministrar la base de datos a Amazon Sagemaker, el desarrollo que seguirá AWS para ir desde el entrenamiento del modelo hasta el despliegue del mismo para poder ser utilizado. El diagrama a continuación muestra el funcionamiento general para este capítulo:

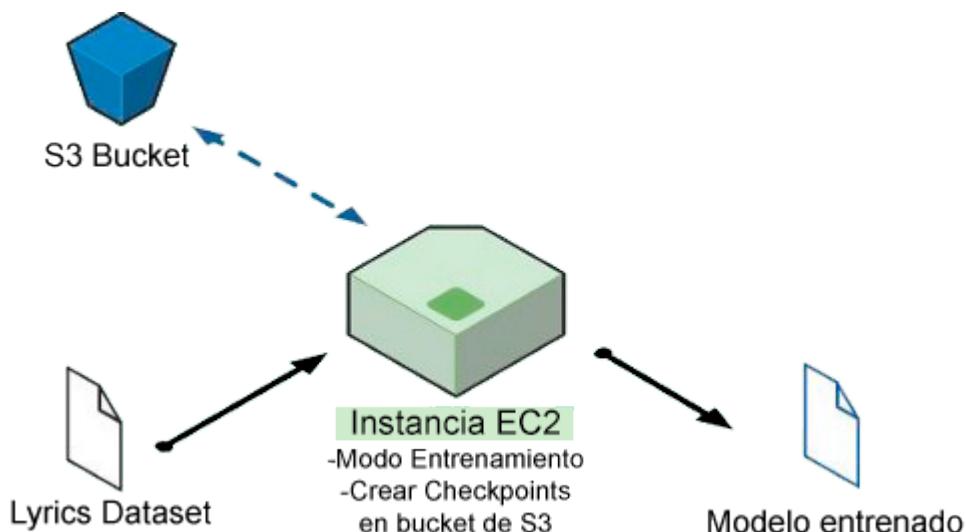


Figura 4.11: Diagrama de instancias de AWS

Como se observa, ingresará la base de datos generada y se utilizará una instancia EC2, que a su vez usará en caso de necesitar más recursos un bucket de S3 para crear checkpoints resultando con el modelo entrenado a través de un número dado de epochs.

4.5.2. Amazon Sagemaker

Amazon SageMaker es un servicio que ayuda a científicos y desarrolladores a construir, entrenar e implementar de manera rápida y sencilla modelos de machine learning. SageMaker suprime las tareas difíciles y tediosas de cada paso del proceso de machine learning para que sea más fácil crear modelos de alta calidad.

Dentro de las virtudes de SageMaker, y razón por la cual lo preferimos sobre la competencia, es que cuenta con 2 herramientas esenciales las cuales nos servirán al momento de entrenar el modelo:

- SageMaker Experiments: Ayuda a organizar y rastrear iteraciones en los modelos de aprendizaje automático al capturar los parámetros de entrada, configuraciones y resultados para así guardarlos como “experimentos”, permitiendo que de forma posterior se pueda llevar una línea de tiempo de lo que ha sucedido y comparar estos resultados de una manera visual.
- SageMaker Debugger: Registra métricas del entrenamiento en tiempo real, genera advertencias y consejos de solución cuando se detectan problemas de entrenamiento comunes, así como monitorea los recursos del sistema para brindar recomendaciones respecto a cómo reasignar recursos y así reducir los costos.
- Managed Spot Training: Es una herramienta que permite entrenar los modelos de aprendizaje automático con instancias de Amazon EC2 que pueden demorar más al entrenar a cambio de abaratar los costos de entrenamiento hasta en un 90 %.

4.5.3. Suministrado de datos

Al ingresar a SageMaker lo primero que nos es requerido hacer es una instancia bloc de notas (Jupyter Notebook) dentro del cual escribiremos el código en Python que ocuparemos para ejecutar todo el procedimiento visto en la iteración tres.

Instancia de bloc de notas

Dentro de Amazon Sagemaker creamos un bloc de notas (Jupyter Notebook) dentro del cual podremos operar con código Python y el cual tiene como ventaja principal el correr código por bloques evitando de esta manera el tener que reescribirlo y pudiendo correr todo de manera secuencial nuevamente o insertar nuevos bloques entre los previamente creados según veamos más conveniente.

Instancias de bloc de notas					
Crear instancia de bloc de notas					
Nombre	Instancia	Hora de creación	Estado	Acciones	
LyricGenerator	ml.t3.medium	May 25, 2021 00:02 UTC	InService	Abrir Jupyter	Abrir JupyterLab

Figura 4.12: Instancia creada en AWS

Dentro de la instancia de Jupyter se va a realizar lo siguiente:

- Filtrar y preparar el contenido del dataset que le proporcionemos.
- Entrenar el modelo, el cual podremos supervisar desde una instancia proporcionada por Amazon SageMaker durante este paso llamado “Instancia de Entrenamiento de Aprendizaje de Máquina”.
- Generar el modelo así como dar estadísticas del mismo para poder corroborar que fue generado exitosamente.
- Desplegar el modelo, para este paso de la misma forma que ocurre con el entrenamiento del mismo, Sagemaker proporciona una instancia que nos permitirá alojarlo para su posterior uso llamada “Instancia de Alojamiento de Aprendizaje de Máquina”.

Posterior a todos los pasos mencionados anteriormente que se realizarán dentro de la instancia en Jupyter, se puede crear ahí mismo una ‘API endpoint’ que responderá a una petición hecha por el usuario para acceder al modelo generado y así generar o devolver la letra.

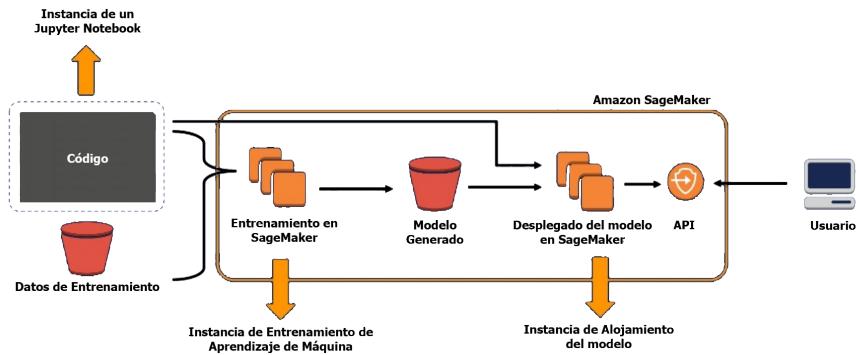


Figura 4.13: Diagrama del funcionamiento de SageMaker

Como se puede apreciar en el diagrama, dentro de Amazon SageMaker ocurrirán todos los procesos que requerimos se ejecuten para poder entrenar y generar el modelo al que el usuario final podrá contactar por medio de una API para solicitar la letra de una canción según sus requerimientos especificados.

4.5.4. Resultados

La generación del modelo de neural network utilizando herramientas en la nube, podrá ahorrar el proceso de entrenamiento cada vez que se corra algún algoritmo y teniendo acceso a herramientas proporcionadas por empresas como AWS, acortan de gran manera los tiempos que corriendo de forma local, tardaría bastantes horas en realizarse. El inconveniente de usar este tipo de herramientas es que si se tienen mal configuradas pueden causar grandes gastos al usuario que las utiliza. Sin embargo, se harán las pruebas pertinentes para que esto no ocurra.

A continuación se muestra un modelo aproximado para poder generar nuestro modelo en la nube:

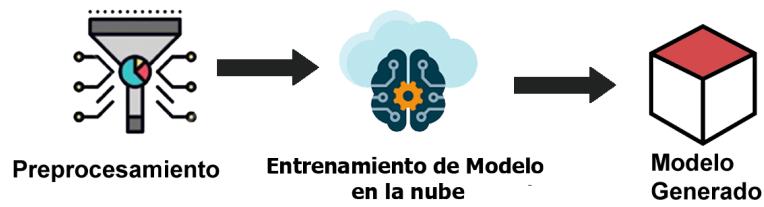


Figura 4.14: Diagrama simple del proceso seguido para generar el modelo en la nube

4.6. Aplicación web

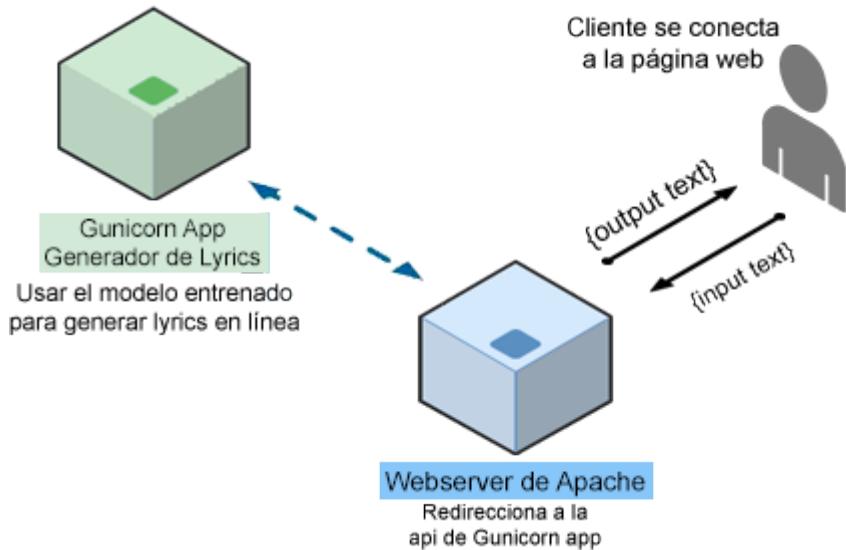


Figura 4.15: Diagrama de conexión entre modelo y usuario

La figura anterior, muestra el funcionamiento general entre la aplicación del modelo entrenado para generar lyrics en línea y el usuario final accediendo a la red desde su navegador.. En este apartado se mostrará un primer boceto de las vistas que verá el usuario final en la aplicación web.



Figura 4.16: Pagina inicial que se mostrará al usuario

The screenshot shows the Lyrics.ia website. At the top, there is a navigation bar with links to 'Inicio', 'Quiénes Somos', and 'FAQ'. The main title 'Lyrics.ia' is centered above a brief description: 'Somos un servicio gratuito, el cual ofrece la posibilidad de generar letras musicales nuevas, haciendo uso de la inteligencia artificial.' Below this, a section titled 'Letra generada por la IA' displays a block of generated lyrics. The lyrics are as follows:

```

And you can shine, you can't lose.
If I don't think of his father?
The plans I stand alone, but I don't mean
Skipper won't pop it all away?
I did I do for you?
I don't do it the shame?

The first to ask.
Can't believe you can.
Baby that song, play me closely then stole my heart feel this
way?
Into the mind
But the meteor men beg to differ
Judging by the sense of decency, sir?

If I couldn't even give it to let yourself drown?
Till He appeared and the way down the hall.
Climaxin, body bakin, makin me high as a bee with a little bit
louder now Hey!Carramba I tell myself I feel so good
The change of heart?

Again, this time, little girl?
They found Him in a rut You gotta I gotta carry on-ST!
I come to my song.
I only want to learn the meaning somewhere underneath.
It's not too late to find Happiness and I know for sure?
With her went my future will be!

Well there ain't no use I need you anymore
Fire, You cannot hurt me so I pray.
Me nigga, It's me on to it.
Ooh ooh and there were dead
There was a Jew, do you know?

If I couldn't even give it to let yourself drown?
Till He appeared and the way down the hall.
Climaxin, body bakin, makin me high as a bee with a little bit
louder now Hey!Carramba I tell myself I feel so good
The change of heart?

If I couldn't even give it to let yourself drown?
Till He appeared and the way down the hall.
Climaxin, body bakin, makin me high as a bee with a little bit
louder now Hey!Carramba I tell myself I feel so good
The change of heart?

```

At the bottom of the page are three buttons: 'Descargar' (Download), 'Generar Nuevamente' (Generate Again), and 'Reiniciar' (Reset). There are also social media sharing icons for Twitter, Instagram, and Facebook.

Figura 4.17: Aplicación web, pagina una vez generada la letra musical.

Capítulo 5

Desarrollo

5.1. Desarrollo del modelo

A continuación, explicaremos la herramienta que se utilizó para desarrollar el modelo de generación de letras musicales.

5.1.1. Amazon AWS

Originalmente se planeaba trabajar con Amazon AWS, pero al momento de crear las máquinas virtuales donde se iba a laborar con la creación del modelo, nos encontramos con la problemática de que, para realizar algunas acciones, los comandos requeridos para estas tenían un cargo monetario extra a la cuenta además de lo que se invertía en mantener la máquina virtual activa.

Aunado a esto la herramienta nos amigable con el usuario, y no te brinda un manual para facilitar el manejo de esta. Por esta razón, además de ser más costoso de lo planeado se decidió cambiar de plataforma para el desarrollo del modelo.

5.1.2. Google Cloud Platform

Después se probó trabajar con la plataforma de Google la cual es similar en la creación de máquinas virtuales a la plataforma de Microsoft Azure, esta plataforma de Google es más amigable e intuitiva para el usuario, al momento de crear tu cuenta en esta, se te otorgan 300 dólares, los cuales lo mas probable es que utilices en la maquina en la que vas a trabajar en un futuro.

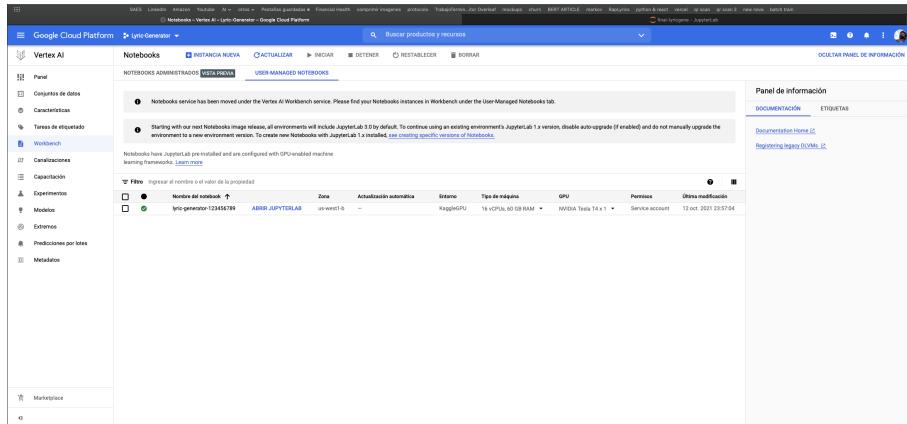


Figura 5.1: Máquina creada para el desarrollo del modelo.

Para el desarrollo de modelos en esta plataforma es como utilizar una libreta de Jupyter, creando una libreta en la máquina virtual que va a procesar la información y la cual va a estar trabajando con los datos del modelo, así como una cubeta donde se va a encontrar tu base de datos que el modelo va procesar, en este caso las letras de las canciones del genero pop.

```

# Log: Final-lyricgenerator.ipynb
# Add Checkpoint  Clear Log  Log Level: Debug
[1]: %%time
2021-08-32 00:28:43.000000 W tensorflow/core/framework/op_kernel.cc:108] Allocation of 2127386958 exceeds 10% of free system memory.
236/14248 [.....] - ETA: 6:38:48 - loss: 6.4959 - accuracy: 0.8446

```

Figura 5.2: Libreta y entrenamiento del modelo.

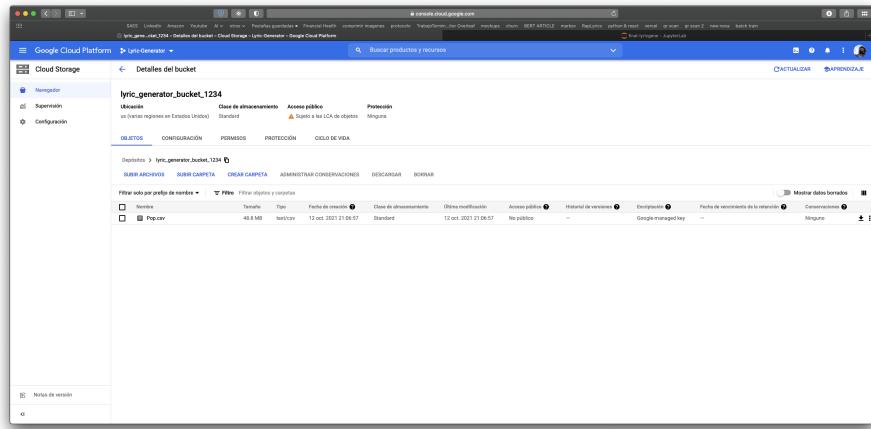


Figura 5.3: Cubeta donde se encuentra la base de datos que contiene las letras de las canciones.

No utilizamos esta herramienta al final, principalmente a que los resultados obtenidos fueron similares a los conseguidos con el modelo desarrollado en la plataforma de Kaggle, además debemos considerar que el mantenimiento de la máquina virtual en la plataforma de Google era de 265 dólares mensuales y no se cuenta con el presupuesto para este mantenimiento.

5.1.3. Kaggle

Al final de realizar prueba decidimos utilizar la plataforma de Kaggle la cual con una cuenta gratuita puedes hacer funciones similares a las presentadas en la plataforma de Google, con algunas limitantes en cuanto a los recursos que puedes emplear al momento de trabajar.

En esta plataforma también debes seleccionar los elementos con los cuales tu ambiente va a trabajar, en este caso el máximo de memoria RAM disponible era de 16Gb, un disco de 73Gb y un GPU de 13 Gb, estos recursos fueron los utilizados para el procesamiento de nuestro conjunto de datos, así como una libreta la cual va a ser el área donde vamos a estar trabajando.

Debido a las limitantes de procesamiento nos vimos en la necesidad de limitar el modelo a entrenar, en lugar de utilizar las 28441 letras de canciones del genero pop, solo se pudo trabajar con 700.

Con estas 700 lyrics, se decidió obtener información de ellas, siendo más específicos, estadísticas como el número de palabras en cada canción, esto

con el fin de determinar la frecuencia de distribución del número de palabras de cada texto y para tener una idea del promedio de palabras, esto con el fin de tenerlo en cuenta al momento de realizar la generación de texto.

count	700.000000
mean	248.065714
std	97.323252
min	31.000000
25%	178.000000
50%	233.000000
75%	309.000000
max	589.000000

Figura 5.4: Estadísticas de las palabras

Lo que podemos observar en la imagen anterior es:

- **Count:** el cual es el número de canciones analizadas.
- **Mean:** el promedio de palabras por canción.
- **Std:** la desviación estándar de las palabras.
- **Min:** la menor cantidad de palabras encontradas en una canción.
- **Max:** la mayor cantidad de palabras encontradas en una canción.

Además, se realizó una tokenización a las letras de nuestras 700 canciones, esto quiere decir que se separó cada palabra y cada palabra se convirtió en un número. Para este proceso se hizo uso de Keras y la clase Tokenizer(), la cual cuenta con dos métodos importantes:

- **_fit_on_text():** El cual actualiza el vocabulario interno en función de una lista de textos determinada o, en este caso, la columna "Lyrics", donde cada entrada de la lista será un token.
- **_texts_to_sequences():** El cual transforma cada texto dentro de la lista de textos proporcionados en una secuencia de números enteros; solo se considerarán las palabras conocidas por el tokenizador.

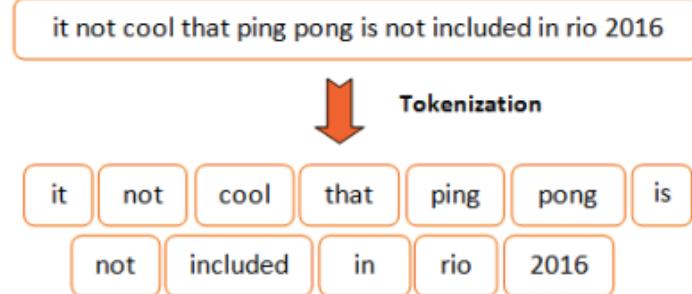


Figura 5.5: Tokenizado de las palabras [86]

Antes de la generación del modelo, es necesario normalizar todas las oraciones a una misma longitud, para evitar el desbordamiento de la memoria y conseguir que las capas del modelo sean mucho más profundas, este es un proceso simple el cual consiste en agregar ceros al comienzo del texto, dando como resultado capas del mismo tamaño.

La posición donde se sumarán los ceros viene determinada por el relleno del argumento, en este caso, se hará al comienzo de la secuencia.

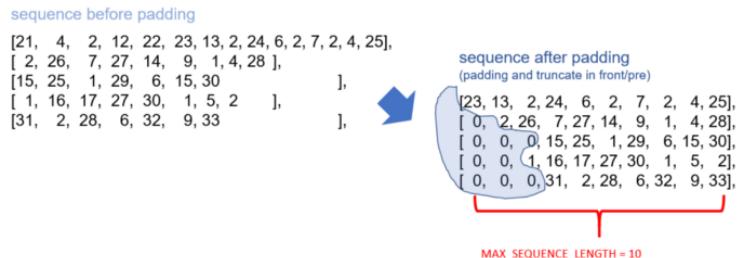


Figura 5.6: Padding del tokenizado de las palabras

5.2. Creación del modelo

En este caso se utilizó el modelo LSTM Bidireccional, este tipo de redes neuronales se ejecutan como su nombre lo indica: en dos direcciones. Esto quiere decir que va del pasado al futuro y viceversa, así es como el modelo conserva la información de ambos estados en cualquier momento. Las redes neuronales de LSTM se utilizan principalmente cuando el contexto está involucrado.

Los modelos en Keras se definen como una secuencia de capas, y el modelo secuencial se trata de agregar capas de una en una. Las capas son el componente básico de la red neuronal.

Dentro de estas capas podemos encontrar:

5.2.1. Embedding o incrustación

Es una capa central, solo se puede usar como la primera capa en un modelo, convierte los números enteros positivos en vectores densos de un tamaño fijo (el primer parámetro es el tamaño del vocabulario, el segundo parámetro es la dimensión de la incrustación densa y el tercer parámetro es sobre la longitud de las secuencias, este se requiere ya que usaremos una capa densa más adelante)

```
1 model.add(Embedding(total_words, 40, input_length=max_sequence_len-1))
```

5.2.2. Bidireccional

Es una capa recurrente, una envoltura bidireccional para las redes neuronales de tipo RNN's que recibirá una capa como entrada, siendo la capa LSTM la que elegimos, recibirá un entero positivo como entrada que se refiere a la cantidad de nodos de salida que se deben devolver.

```
1 model.add(Bidirectional(LSTM(250)))
```

5.2.3. Dropout

Es una capa de regularización. Esta capa establece aleatoriamente las unidades de entrada en 0, con una frecuencia del valor que le pasamos, en cada paso durante el tiempo de entrenamiento, lo que ayuda a evitar el sobreajuste.

```
1 model.add(Dropout(0.1))
```

5.2.4. Densidad

Es una capa central y una capa de red neuronal densamente conectada. Recibe como primer parámetro un entero positivo que se refiere a la cantidad de nodos de salida que deben devolverse. El segundo parámetro es el llamado activación que define el tipo de predicciones que puede hacer el modelo; para el tipo de problema que estamos considerando, el que se adapta mejor es softmax, que genera un vector de valores (entrada) que puede interpretarse como probabilidades de ser utilizado.

```
1 model.add(Dense(total_words, activation='softmax'))
```

5.2.5. Método de compilación, algoritmo de optimización y métrica de rendimiento

Perdida: También conocida como función de costos; funciona durante el proceso de optimización y su función es calcular el error del modelo. La entropía cruzada se utiliza para estimar la diferencia entre una distribución de probabilidad estimada y predicha. Se utilizará categorical_crossentropy porque es más adecuado para este tipo de problemas y se usa casi universalmente para entrenar redes neuronales de aprendizaje profundo debido a los resultados que produce. Optimización: Se encarga de reducir las pérdidas y brindar los resultados más precisos posibles. Adam es la opción elegida porque es la mejor opción que ofrece Keras para entrenar la red neuronal en menos tiempo y de manera más eficiente. Earlystop detendrá el entrenamiento si el modelo ha dejado de mejorar, esto se verificará al final de cada epoch. En este caso, la "precisión." "accuracy" se utilizará como métrica de rendimiento. El método fit es el encargado de entrenar el modelo para el número fijo de epochs dados.

```
1 model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
2 earlystop = EarlyStopping(monitor='loss', min_delta=0, patience=3, verbose=0, mode='auto')
3 history = model.fit(X, y, epochs=20, verbose=1, callbacks=[earlystop])
```

Quedando como resultado el código del modelo de la siguiente forma:

```

1 model = Sequential()
2 model.add(Embedding(total_words, 40, input_length=max_sequence_len-1))
3 model.add(Bidirectional(LSTM(250)))
4 model.add(Dropout(0.1))
5 model.add(Dense(total_words, activation='softmax'))
6 model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
7 earlystop = EarlyStopping(monitor='loss', min_delta=0, patience=3, verbose=0, mode='auto')
8 history = model.fit(X, y, epochs=20, verbose=1, callbacks=[earlystop])

```

Una vez completado el entrenamiento de nuestro modelo, lo que falta es importarlo para probar cómo funciona, en nuestro caso nombramos al modelo “song_lyrics_generator” y se importó de la siguiente forma, llamándola a través del enlace de nuestra libreta de Kaggle:

```

1 from keras.models import load_model
2 model = load_model('../input/songlyricmodel/song_lyrics_generator.h5')

```

Ya que contemos con el modelo importado, se creó una función para generar la letra de una canción utilizando el modelo previamente entrenado, el cual predecirá las siguientes palabras en base a la palabra(s) de entrada suministradas como ‘seed_text’. Para que esto funcione, se debe aplicar una tokenización al seed_text, luego se aplicará un relleno a las secuencias generadas y se pasará al modelo entrenado para que se pueda predecir la siguiente palabra.

```

1 def complete_this_song(seed_text, next_words):
2     for _ in range(next_words):
3         token_list = tokenizer.texts_to_sequences([seed_text])[0]
4         token_list = pad_sequences([token_list], maxlen=max_sequence_len-1,
5             padding='pre')
5         predicted = model.predict_classes(token_list, verbose=0)
6         output_word = ""
7         for word, index in tokenizer.word_index.items():
8             if index == predicted:
9                 output_word = word
10                break
11         seed_text += " " + output_word
12     return seed_text

```

5.3. Desarrollo del back-end

Para el back-end se decidió trabajar con una aplicación web desarrollada en Flask, debido a que de esta era fácil y rápida de implementar, además que permitía una mejor integración con el modelo que se elaboró usando Python y la aplicación web trabajada en React.

Si se desea desarrollar una aplicación web en Flask lo primero que se tiene que hacer o que se recomienda hacer es crear un ambiente virtual usando virtualenv esto para separar nuestro entorno Python instalado originalmente del que se va a utilizar en la aplicación. Una vez realizado este procedimiento, se inició el desarrollo del código.

```
1 # -*- coding: utf-8 -*-
2 from datetime import datetime
3 from flask import Flask, jsonify
4 from flask_restful import Resource, Api
5 from flask_apispec import marshal_with, doc, use_kwargs
6 from flask_apispec.views import MethodResource
7 from apispec import APISpec
8 from apispec.ext.marshmallow import MarshmallowPlugin
9 from flask_apispec.extension import FlaskApiSpec
10 from flask_cors import CORS
11 from random import randint
12 # Own Libraries
13 from .utils.schemas import HealthSchema, GetLyrics, PostLyrics
14 from .utils.generate_lyrics import GenerateLyric
15 from .utils.constants import FLASK_ENV, VERSION, PROJECT
16
17 app = Flask(__name__)
18 api = Api(app) # Flask restful wraps Flask app around it.
19 cors = CORS(app)
20
21 app.config.update({
22     'APISPEC_SPEC': APISpec(
23         title='Lyric Generator',
24         version='v1',
25         plugins=[MarshmallowPlugin()],
26         openapi_version='2.0',
27     ),
28     'APISPEC_SWAGGER_URL': '/swagger/', # URI to access API Doc JSON
29     'APISPEC_SWAGGER_UI_URL': '/swagger-ui/' # URI to access UI of API Doc
30 })
31 docs = FlaskApiSpec(app)
32
33 # Endpoints
34 class Lyrics(MethodResource, Resource):
35     """
36         Source endpoint to check health of the API
37     """
38     @doc(description='Health-check to see the status of the API.', tags=['Lyric Generator Status'])
39     @marshal_with(HealthSchema)
40     def get(self):
41         """
42             Get method represents a GET API method
43     
```

```

43     """
44     return {
45         'project': PROJECT,
46         'version': VERSION,
47         'environment': FLASK_ENV,
48         'date': datetime.now(),
49     }
50
51 """
52 Class to generate the lyric of the API
53 """
54 @doc(description='Send input to generate Lyric', tags=['Generate Lyric'])
55 @use_kwargs(GetLyrics, location=('json'))
56 @marshal_with(PostLyrics)
57 def post(self, **kwargs):
58     """
59     Post method represents a POST API method
60     """
61     response = {}
62     lyrics = GenerateLyric(kwargs)
63     response['chorus'] = lyrics.chorus(randint(10,15))
64     response['first_verse'] = lyrics.complete_this_song(randint(10,15))
65     response['generated_lyric'], response['percentage'] = lyrics.
66     complete_this_song(randint(10,15))
67     response['end_verse'] = lyrics.complete_this_song(randint(10,15))
68
69     return jsonify(response)
70
71 @doc(description='Health-check to see the status of the API.', tags=['
72     Lyric Generator Status'])
73 @marshal_with(HealthSchema)
74 def get(self):
75     """
76     Get method represents a GET API method
77     """
78     return {
79         'project': PROJECT,
80         'version': VERSION,
81         'environment': FLASK_ENV,
82         'date': datetime.now(),
83     }
84
85 # Add Endpoints
86 api.add_resource(Lyrics, '/lyrics')
87
88 # Add Swagger Documentation
89 docs.register(Lyrics)
90
91 # Handling of 404 errors.
92 @app.errorhandler(404)
93 def page_not_found(e):
94     return jsonify({'error': "resource not found", "code": "404"}), 404
95
96 # We run the Flask application and, if it is running in a development
97 # environment,
98 # we run the application in debug mode.
99 # to run only ``FLASK_ENV=test FLASK_APP=src/app python -m flask run --host
100 # =0.0.0.0 --port=80```
101 app.run(debug=FLASK_ENV == 'development', host='0.0.0.0')

```

El código app.py mostrado anteriormente es el código principal de nuestro back-end y es este el que se despliega en el navegador web, lo primero que se visualiza al entrar es información con respecto a esta aplicación como su nombre, versión y los plugin con los que esta cuenta.

Para poder ver un cómo es que realmente funciona el back-end se realizó una pequeña interfaz y es posible acceder a ella a través de la dirección x/swagger-ui/ en ella podemos encontrar con mayor detalle el estado actual del api, esta información se encuentra ubicada en el espacio nombrado como health.

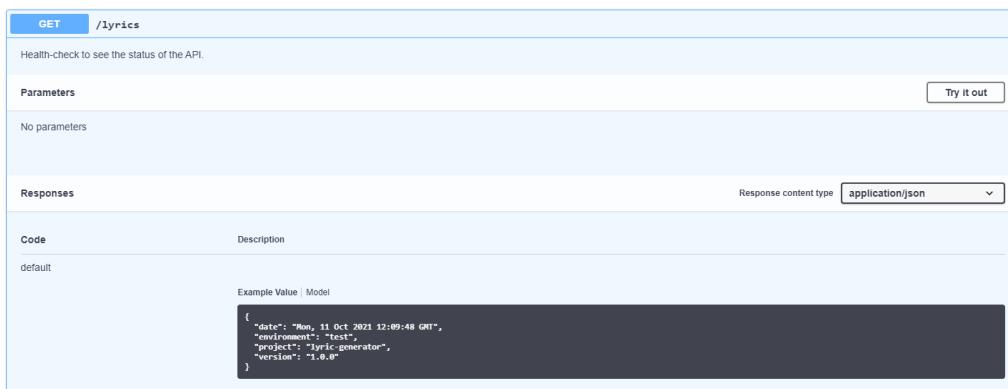


Figura 5.7: Información sobre el estado del Back-end

Un poco más abajo se encuentra el método post de la api, este es el encargado de recibir la información de la aplicación web realizada en React en formato json, siendo más específicos la palabra en inglés introducida por el usuario, para posteriormente enviar esta información al modelo.

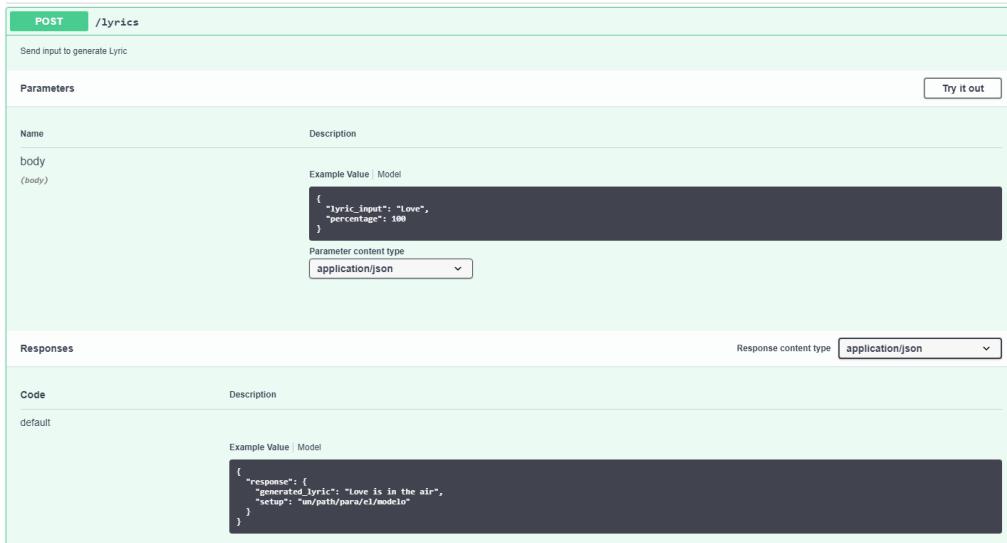


Figura 5.8: Cuadro del método Post

Para realizar la tarea más importante del proyecto, es decir, la generación del texto, se desarrolló el siguiente código:

```

1  from ..utils.constants import MODEL_PATH
2  from random import randint
3  # Keras
4  from tensorflow.keras.models import load_model
5  from tensorflow.keras.preprocessing.text import Tokenizer
6  from tensorflow.keras.preprocessing.sequence import pad_sequences
7  import pandas as pd
8  import numpy as np
9  import tensorflow as tf
10
11 # Aquí se carga el modelo y se vuelve a tokenizar para cargar
12 # las posibles respuestas del modelo
13 print(MODEL_PATH)
14 df = pd.read_csv(MODEL_PATH + '/pop_model.csv')
15 model = load_model(MODEL_PATH + '/song_lyrics_generator.h5')
16
17 tokenizer = Tokenizer()
18 tokenizer.fit_on_texts(df['Lyric'].astype(str).str.lower())
19 total_words = len(tokenizer.word_index)+1
20 tokenized_sentences = tokenizer.texts_to_sequences(df['Lyric'].astype(str))
21 tokenized_sentences[0]
22 input_sequences = list()
23 for i in tokenized_sentences:
24     for t in range(1, len(i)):
25         n_gram_sequence = i[:t+1]
26         input_sequences.append(n_gram_sequence)
27 max_sequence_len = max([len(x) for x in input_sequences])
28 input_sequences = np.array(pad_sequences(input_sequences, maxlen=
29                             max_sequence_len, padding='pre'))
30 X, labels = input_sequences[:, :-1], input_sequences[:, -1]
31 y = tf.keras.utils.to_categorical(labels, num_classes=total_words)

```

```

32
33     class GenerateLyric(object):
34         """
35             TODO:
36                 - Generate Model and integrate it in this class.
37                 - Integrate it in React
38         """
39     def __init__(self, kwargs):
40         self.seed_text, self.percentage = kwargs["lyric_input"], kwargs["percentage"]
41
42     def complete_this_song(self, next_words):
43         for _ in range(next_words):
44             token_list = tokenizer.texts_to_sequences([self.seed_text])[0]
45             token_list = pad_sequences([token_list], maxlen=max_sequence_len - 1, padding='pre')
46             predicted = np.argmax(model.predict(token_list, verbose=0), axis=1)
47             output_word = ""
48             for word, index in tokenizer.word_index.items():
49                 if index == predicted:
50                     output_word = word
51                     break
52             self.seed_text += " " + output_word
53     return self.seed_text, self.percentage
54
55     def chorus(self, next_words):
56         for _ in range(next_words):
57             token_list = tokenizer.texts_to_sequences([self.seed_text])[0]
58             token_list = pad_sequences([token_list], maxlen=max_sequence_len - 1, padding='pre')
59             predicted = np.argmax(model.predict(token_list, verbose=0), axis=1)
60             output_word = ""
61             for word, index in tokenizer.word_index.items():
62                 if index == predicted:
63                     output_word = word
64                     break
65             self.seed_text += " " + output_word
66     return self.seed_text, self.percentage
67
68     def main():
69         inp = input()
70         response = {}
71         lyrics = GenerateLyric(inp)
72         response['chorus'] = lyrics.chorus(randint(10,15))
73         response['first_verse'] = lyrics.complete_this_song(randint(10,15))
74         response['generated_lyric'], response['percentage'] = lyrics.
75         complete_this_song(randint(10,15))
76         response['end_verse'] = lyrics.complete_this_song(randint(10,15))
77         print(response)
78
79     if __name__ == '__main__':
80         main()

```

En este código lo primero que realiza es cargar el modelo previamente entrenado debido a que va a ser requerido para la generación de texto, utilizando la palabra introducida por el usuario es procesada y ocupada en nuestra clase

GenerateLyric donde la palabra al igual que el resto de palabras en nuestro modelo es tokenizada y separada en caso de haber recibido mas de una vocablo, posteriormente utilizado el modelo previamente cargado y valor de la palabra tokenizada va a tratar de predecir la siguiente palabra.

Este proceso de tokenizar el texto y generar la próxima palabra a partir del valor obtenido de la tokenizacion se va a repetir hasta que se cumpla con el tamaño del verso o coro en el que se esté trabajando.

Una vez terminado con la generación del texto, es enviado a la pagina web desarrollada en React como una respuesta.

5.4. Desarrollo de la página web

El desarrollo de la aplicación web se decidió realizarla haciendo uso de la biblioteca de Javascript conocida como React o ReactJS la cual consiste en crear interfaces interactivas, con la cual se puede diseñar vistas simples, en las cuales se actualizan y renderizarán los componentes necesarios cuando exista algún cambio en los datos.

Se decidió usar este recurso para el desarrollo de la página web, debido a que permite que esta sea compatible con dispositivos móviles sin la necesidad de volver a escribir código para estas tecnologías.

Para poder trabajar con React fue necesario descargar Node.js desde su página (<https://nodejs.org/es/>) la cual nos va a permitir crear aplicaciones web utilizando JavaScript, además de que al instalarlo obtendremos npm herramienta que posteriormente nos permitirá instalar paquetes con los cuales podemos añadir nuevas funciones a nuestra aplicación y que a su vez son compatibles con React.

Una vez que se tiene instalado Node.js, para poder ejecutar una aplicación web en terminal, debemos escribir npm start, lo cual ejecutara un servidor de desarrollo de manera local, al cual se puede acceder a través de la siguiente dirección (localhost:3000) en cualquier navegador.

Como React se basa en crear vistas, debíamos tener claro que queremos que el usuario vea, por ello se creó una carpeta en la cual se colocaron las páginas con las cuales se iban a trabajar y con las que el usuario va a interactuar. Estas fueron la página de inicio, la de preguntas frecuentes, la de ejemplos de canciones previamente generadas, una acerca de nosotros, una de contacto y una que despliega de manera gráfica el back-end.

```
✓ pages
# About.css
JS About.js
JS Dev.js
# Faq.css
JS Faq.js
# GenerateSong.css
JS GenerateSong.js
IMG Headphones.jpg
JS Home.js
# LyricPage.css
JS LyricPage.js
# Samples.css
JS Samples.js
```

Figura 5.9: Páginas trabajadas

La página más importante al momento de desarrollar la aplicación es la página principal, debido a que es esta la primera que va a ver el usuario al entrar y con la que más va a interactuar, en esta, aparece un mensaje con un botón el cual invita al usuario a generar su propia letra musical.

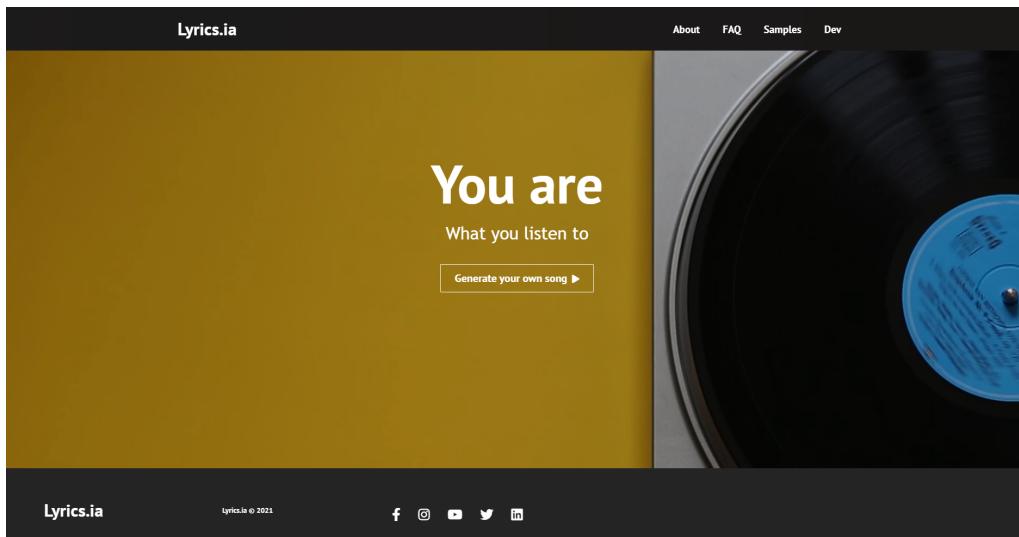


Figura 5.10: Página de bienvenida

Si el usuario le da clic al botón, como React usa estados para renderizar lo que ve el usuario, se muestra en pantalla el formulario donde se le pide al usuario que introduzca una palabra en el idioma inglés.

This screenshot shows the 'Generate tu letra de canción' (Generate your own song) form. It features a large bold title at the top. Below it, a subtitle reads 'Introduce una palabra en idioma inglés, así como escoge un porcentaje de rimas para poder generar tu canción.' A text input field is labeled 'Palabra en inglés:' and contains the placeholder 'Your word'. A horizontal slider is labeled 'Porcentaje de rimas dentro de la canción:' with a value set to 50%. A blue button labeled 'Generar cancion' is positioned below the slider.

Figura 5.11: Formulario

Es en esta parte donde los datos proporcionados por el usuario son leídos y posteriormente enviados a nuestro back-end para ser procesados futuramente por el modelo. A continuación, se muestra el código de la función la cual realiza este proceso.

```

1 function sendWord(engword, percentval) {
2
3     localStorage.setItem("EnglishWord-Value", engword);
4     localStorage.setItem("Percentage-Value", percentval);
5
6     fetch('http://localhost:5000/lyrics', {
7         method: 'POST',
8         headers:{'content-type':'application/json', 'Access-
9             Control-Allow-Origin':'*'},
10        body:JSON.stringify({ "lyric_input":engword, "
11            percentage":percentval})
12    }).then(response => {
13        return response.json()
14    }).then(json => {
15        this.setState({englishword: json[0]})
16    }).catch(error => {
17        console.log(error)
18    })
19 }

```

Esta función lo primero que se hace es recibir los datos (la palabra) que introdujo el usuario, esto se hace buscando el identificador del elemento de la página y obteniendo su valor, este, se enviará al back-end usando la función de fetch, lo que hace dicha función es buscar la url, que en este caso es donde se encuentra alojado nuestro back-end y busca el método post de este, como el método post de nuestro back-end recibe datos en formato json, los datos que introdujo el usuario deben ser enviados en este formato, usando JSON.stringify y poniéndole a la palabra un identificador que el back-end utilizará.

Después de que el usuario le dio clic al botón de generar canción se muestra una leyenda la cual le pide al usuario que espere unos segundos, que se está trabajando en su letra. Pasado un breve tiempo de espera, en pantalla se le mostrará la letra de la canción generada, esta presentación se realizará haciendo un efecto similar al de una máquina de escribir, así como 3 botones, el primero de ellos le permite descargar la letra generada en un archivo de texto, el segundo le permite al usuario generar una nueva canción utilizando los mismos parámetros con los que generó la letra actual y el último lo regresa al estado anterior para introducir nuevos parámetros y con estos generar una nueva letra.

Lyric generada

Nuestros gatos compositores estan trabajando |

[Descargar](#)

[Generar nuevamente](#)

[Nuevos parametros](#)

Figura 5.12: Leyenda mostrada



Lyric generada

Lyric generada|

[Descargar](#)

[Generar nuevamente](#)

[Nuevos parametros](#)

Figura 5.13: Texto generado

Adentrándonos más en esta sección, la primera función que debemos revisar es la que permite hacer la conexión con el back-end para que la página pueda recibir el texto generado por el modelo.

```
1  useEffect(() => {
2      fetchLyricAsync ()
3  }, [])
4
5  const fetchLyricAsync = async () =>{
6      const result = await axios.get("")
7      console.log(result)
8 }
```

Lo que se hace en el código anterior es realizar una conexión asíncrona con el back-end, esto para que mientras no se reciba una respuesta, el usuario en pantalla va a ver una leyenda que diga que se está trabajando en su texto, cuando en la conexión se reciba una respuesta la leyenda que ve el usuario cambiara, mostrando el texto recibido en la respuesta.

```
1 function download(filename, text) {
2     var element = document.createElement('a');
3     element.setAttribute('href', `data:text/plain;charset
4 =utf-8,${ encodeURIComponent(text)}`);
5     element.setAttribute('download', filename);
6
7     element.style.display = 'none';
8     document.body.appendChild(element);
9
10    element.click();
11
12    document.body.removeChild(element);
13 }
```

Con esta función se permite crear un archivo de texto el cual puede ser descargado en el equipo del usuario y el cual va a contener la letra de la canción que se generó en ese momento.

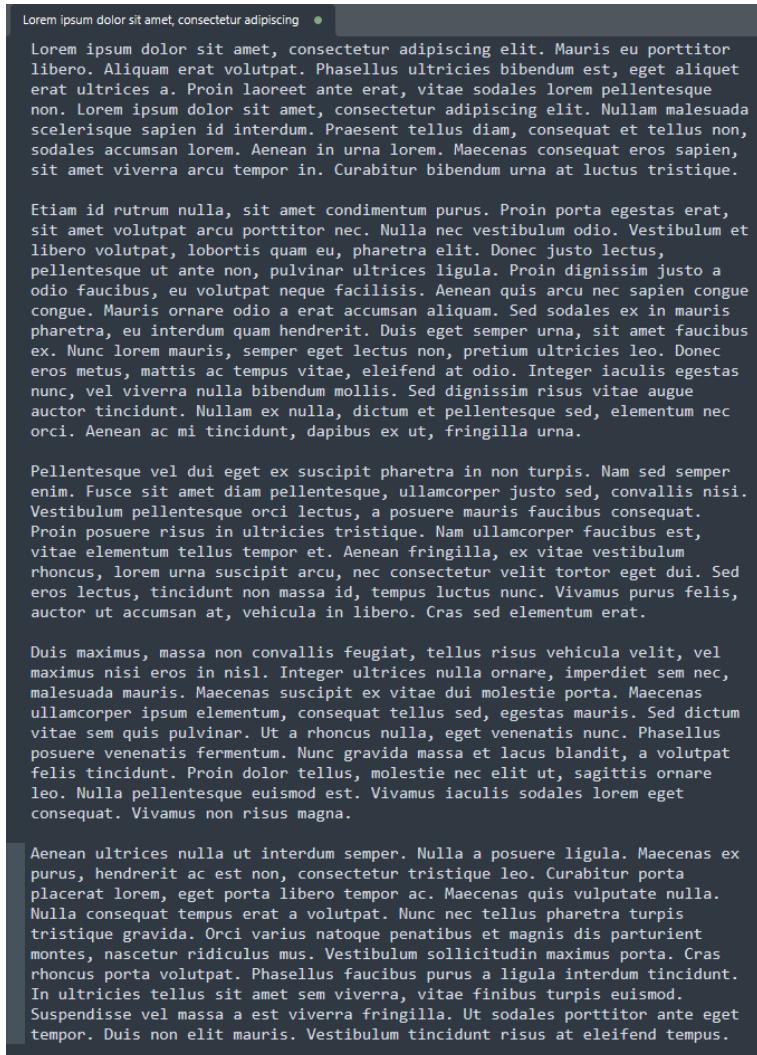


Figura 5.14: Archivo de texto descargado

Si el usuario utiliza el segundo botón, este, utilizando la misma palabra previamente introducida por el usuario llama a la función que conecta con el back-end enviándole una nueva petición de generación de texto y se renderiza nuevamente la vista con la leyenda de que espere unos segundos. En caso de utilizar el tercer botón solo activa el estado que renderiza la parte donde el usuario introduce los datos.

5.5. Desplegando componentes

5.5.1. Despliegue del back-end

5.5.2. Despliegue de la página web

Para el despliegue de la pagina web nos apoyamos de la plataforma Vercel, la cual permite que los desarrolladores puedan desplegar sus paginas de manera rápida, así como poder actualizarla y escalarla de manera sencilla, además permite hacer despliegues de proyectos que se encuentran dentro de una cuenta de Github

Lo primero que debemos hacer para poder trabajar con la línea de comandos de Vercel es instalarlo, para ello debemos recurrir a nuestra terminal e instarlo usando el comando `npm i -g vercel` o `yarn global add vercel`.

```
PS C:\Users\alfre\Documents\TT2\Vercel\tt_api> npm i -g vercel
> vercel@23.1.2 preinstall C:\Users\alfre\AppData\Roaming\npm\node_modules\vercel
> node ./scripts/preinstall.js
C:\Users\alfre\AppData\Roaming\npm\vc -> C:\Users\alfre\AppData\Roaming\npm\node_modules\vercel\dist\index.js
C:\Users\alfre\AppData\Roaming\npm\vercel -> C:\Users\alfre\AppData\Roaming\npm\node_modules\vercel\dist\index.js
+ vercel@23.1.2
added 96 packages from 110 contributors in 37.665s
PS C:\Users\alfre\Documents\TT2\Vercel\tt_api>
```

Figura 5.15: Instalando CLI de Vercel

Una vez instalado, debemos abrir la carpeta de nuestro proyecto a desplegar, y en una terminal dentro de esa carpeta solo es necesario escribir `vercel` para abrir el CLI de Vercel el cual nos preguntara si el proyecto contenido dentro de la carpeta es el que se va a configurar y desplegar, a lo cual damos una respuesta afirmativa, posteriormente nos pide quien lo va a desplegar, para ello usamos una cuenta creada en esta plataforma o vinculamos la de Github para acceder, se nos pregunta el nombre del proyecto y el inicio de los archivos del proyecto a desplegar.

```
PS C:\Users\alfre\Documents\TT2\Vercel2> vercel
Vercel CLI 23.1.2
? Set up and deploy “~\Documents\TT2\Vercel2”? [Y/n] y
? Which scope do you want to deploy to? alfredoesg
? Found project “alfredoesg/vercel2”. Link to it? [Y/n] n
? Link to different existing project? [Y/n] n
? What's your project's name? tt-webp
? In which directory is your code located? ./
```

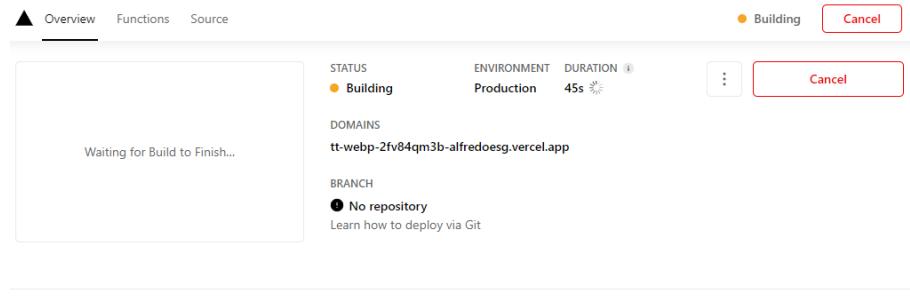
Figura 5.16: Indicando acciones para el despliegue

Después de dar las indicaciones anteriores el CLI de vercel detecta automáticamente el tipo de proyecto trabajado, en este caso una aplicación web usando React, para por último preguntar si se quiere cambiar la configuración del trabajo encontrado, en este caso decimos que no, ya que es correcto el tipo de aplicación encontrada con el trabajado.

```
Auto-detected Project Settings (Create React App):
- Build Command: `npm run build` or `react-scripts build`
- Output Directory: build
- Development Command: react-scripts start
? Want to override the settings? [y/N] n
🔗 Linked to alfredoesg/tt-webp (created .vercel)
🔍 Inspect: https://vercel.com/alfredoesg/tt-webp/4hRMKjZD9125a29aCsQXBhlq4T7tc [2s]
```

Figura 5.17: Detección del tipo de proyecto a desplegar

Vercel comienza a desplegar la aplicación web y nos da un enlace en el cual podemos ver como se encuentra el despliegue, si se presentara algún problema, en la terminal nos indicara cual es.



Deployment Status

Beta Provide feedback

▼ Building 44s

```
11:38:51.997 Retrieving list of deployment files...
11:38:53.390 Downloading 188 deployment files...
11:38:54.264 Analyzing source code...
11:38:55.854 Installing build runtime...
11:38:58.084 Build runtime installed: 2.229s
11:39:00.631 Looking up build cache...
11:39:00.776 Build Cache not found
11:39:02.063 Detected package.json
11:39:02.063 Installing dependencies...
11:39:02.383 var install v1.22.17
```

Figura 5.18: Desplegando el proyecto

Una vez completado el despliegue, se nos genera un enlace con el cual se puede acceder a la aplicación web ya desplegada desde cualquier dispositivo.

Overview Functions Source



STATUS Ready ENVIRONMENT Preview DURATION 1m 15s (1m ago)

DOMAINS tt-[webp-j6b1b7ika-alfredoeg.vercel.app](#)

BRANCH No repository Learn how to deploy via Git

Visit

Figura 5.19: Aplicación web desplegada

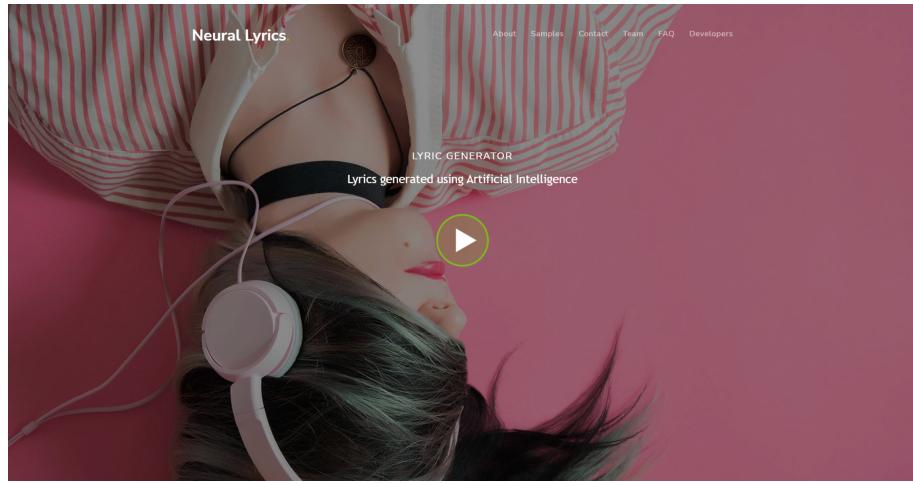


Figura 5.20: Accediendo a la aplicación web desplegada

En nuestro caso, previamente se había adquirido un dominio, para realizar el cambio del enlace proporcionado por Vercel a el enlace del dominio previamente adquirido, en nuestro proyecto, en la parte de configuración, en la de domino, se agrega el enlace y así es posible acceder a esta pagina web desde ese dominio adquirido.

5.6. Pruebas

Capítulo 6

Conclusiones

Capítulo 7

Trabajo a futuro

7.1. Generar rimas en los textos

Se pretende como trabajo a futuro poder generar textos con rimas, estas rimas se generarán a partir de la ultima palabra del primer verso de la letra de la canción generada, además se le pedirá al usuario que indique que tanto desea que rime una letra.

Las rimas generadas serán a partir de que estas se escriban de una manera similar a la palabra o que las estén suenan similares.

Se utilizaría la siguiente composición al momento de generar los textos

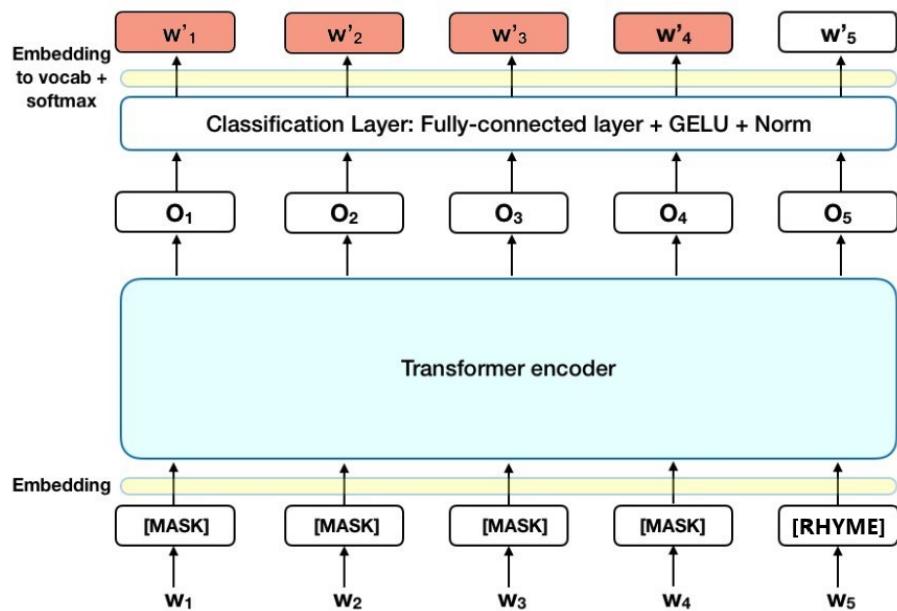


Figura 7.1: Estructura primer verso

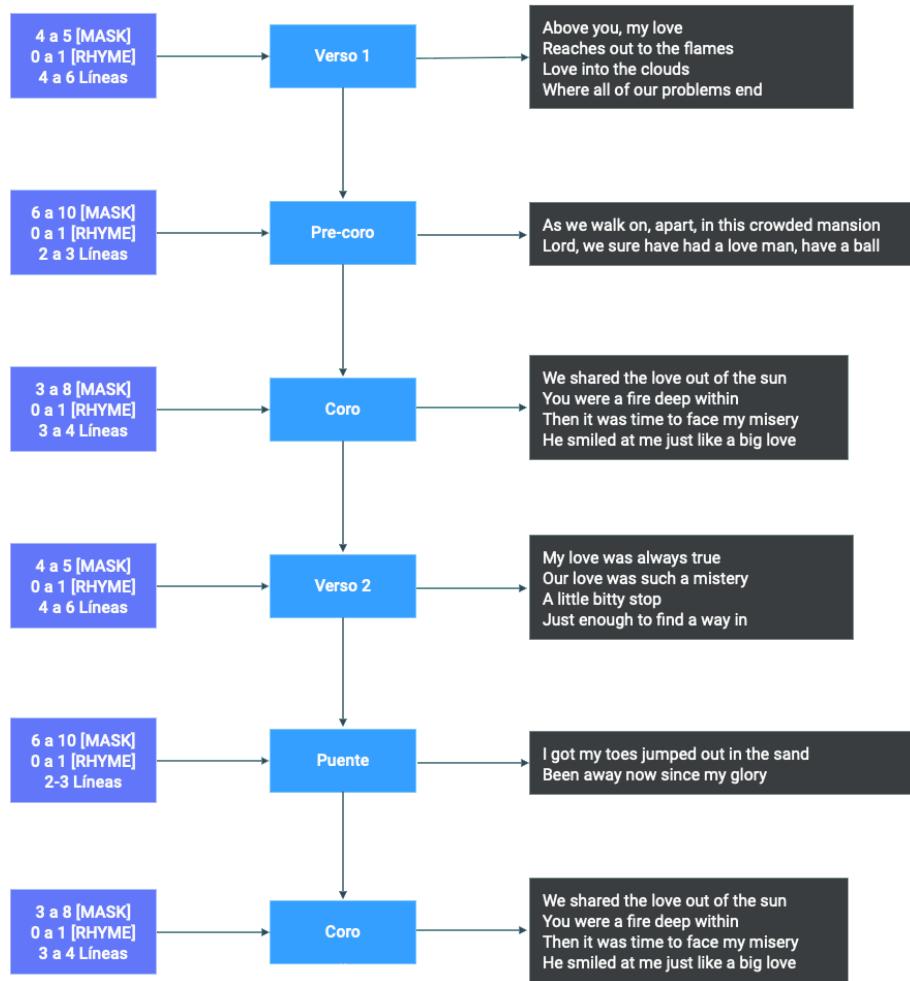


Figura 7.2: Estructura de la letra de la canción con rimas

7.2. Generar textos a partir de otros géneros

Ya que sabemos como entrenar un modelo, podemos utilizar la base de datos original o buscar otra con otros géneros musicales y realizar un procedimiento similar al que se siguió para conseguir el conjunto de datos de las letras de canciones del género pop.

Una vez obtenidos los conjuntos de datos separados de los géneros que queremos trabajar, hacer modelos usando redes neuronales y añadirlos a nuestro back-end.

Para que de esta forma si el usuario en la pagina principal decide crear una canción, se le muestre que puede seleccionar un genero musical que previamente se ha trabajado.

Anexos

.1. Cadenas de Markov

Cadenas de Markov

Una cadena de Markov es un proceso estocástico de tiempo discreto: un proceso que ocurre en una serie de pasos de tiempo, en cada uno de los cuales se hace una elección aleatoria. Una cadena de Markov consta de N estados.

Una cadena de Markov se caracteriza por una matriz de probabilidad de transición $N \times N$, P cada una de cuyas entradas está en el intervalo $[0, 1]$; las entradas en cada fila de P suman 1. La cadena de Markov puede estar en uno de los estados N en cualquier paso de tiempo dado; luego, la entrada P_{ij} nos dice la probabilidad de que el estado en el siguiente paso de tiempo sea j , condicionado a que el estado actual sea i . Cada entrada P_{ij} se conoce como probabilidad de transición y depende solo del estado actual i ; esto se conoce como propiedad de Markov. Así, la propiedad de Markov puede definirse de la siguiente manera:

Ecuación 1 :

$$\forall i, j, P_{ij} \in [0, 1]$$

Y ecuación 2:

$$\forall i, \sum_{j=1}^N P_{ij} = 1.$$

Una matriz con entradas no negativas que satisface la Ecuación 2 se conoce como matriz estocástica. Una propiedad clave de una matriz estocástica es que tiene un vector propio izquierdo principal correspondiente a su valor propio más grande, que es 1.

En una cadena de Markov, la distribución de probabilidad de los siguientes estados para una cadena de Markov depende solo del estado actual y no de cómo llegó la cadena de Markov al estado actual. La siguiente figura muestra una cadena de Markov simple con tres estados. Desde el estado medio A, procedemos con probabilidades (iguales) de 0.5 a B o C. Desde B o C, procedemos con probabilidad 1 a A. La matriz de probabilidad de transición de esta cadena de Markov es entonces:

$$\begin{pmatrix} 0 & 0.5 & 0.5 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

La distribución de probabilidad de una cadena de Markov sobre sus estados

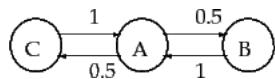


Figure 1: Una cadena de Markov simple con tres estados; los números en los enlaces indican las probabilidades de transición.

puede verse como un vector de probabilidad: un vector en cuyas entradas están en el intervalo $[0, 1]$, y las entradas suman 1. Un vector de probabilidad N -dimensional, cuenta con componentes correspondientes a uno de los estados N de una cadena de Markov, puede verse como una distribución de probabilidad sobre sus estados. Para nuestra cadena de Markov simple de la figura 1, el vector de probabilidad tendría 3 componentes que suman 1.

Podemos ver a un internauta aleatorio en el gráfico web como una cadena de Markov, con un estado para cada página web, y cada probabilidad de transición

representa la probabilidad de pasar de una página web a otra. La operación de movilizarse contribuye a estas probabilidades de transición. La matriz de adyacencia A del gráfico web se define de la siguiente manera: si hay un hipervínculo desde la página i a la página j , entonces $A_{ij} = 1$, de lo contrario $A_{ij} = 0$. Podemos derivar fácilmente la matriz de probabilidad de transición P para nuestra cadena de Markov a partir de la matriz $N \times N A$:

1. Si una fila de A no tiene 1, reemplace cada elemento por $1/N$. Para todas las demás filas, proceda de la siguiente manera.
2. Divida cada 1 en A por el número de 1 en su fila. Por lo tanto, si hay una fila con tres unos, cada uno de ellos se reemplaza por $1/3$.
3. Multiplica la matriz resultante por $1 - \alpha$.
4. Agregue α/N a cada entrada de la matriz resultante, para obtener P .

Podemos representar la distribución de probabilidad de la posición del internauta en cualquier momento mediante un vector de probabilidad \vec{x} . En $t = 0$, el internauta puede comenzar en un estado cuya entrada correspondiente en \vec{x} es 1 mientras que todos los demás son cero. Por definición, la distribución del internauta en $t = 1$ viene dada por el vector de probabilidad $\vec{x}P$; en $t = 2$ por $(\vec{x}P)P = \vec{x}P^2$, y así sucesivamente. Por lo tanto, podemos calcular la distribución del internauta sobre los estados en cualquier momento, dada solo la distribución inicial y la matriz de probabilidad de transición P .

Si se permite que una cadena de Markov se ejecute durante muchos pasos de tiempo, cada estado se visita a una frecuencia (diferente) que depende de la estructura de la cadena de Markov. En nuestra analogía de ejecución, el internauta visita ciertas páginas web (por ejemplo, páginas de inicio de noticias populares) con más frecuencia que otras páginas. Ahora hacemos esta intuición precisa, estableciendo las condiciones bajo las cuales la frecuencia de visita converge a una cantidad fija en un estado estable.

Este documento fue publicado por: Cambridge University Press (Julio 04, 2009) Markov chains. [En línea]. Disponible: <https://nlp.stanford.edu/IR-book/html/htmledition/markov-chains-1.html#fig:figmarkov>

.2. Pruebas utilizando long short term memories

Pop Lyric Generator

Text Generation is a type of Language Modelling problem.

Language Modelling is the core problem for a number of Natural Language Processing tasks such as speech to text, conversational system, and text summarization, etc.

Text Generation is a task which can be be architectured using deep learning models, particularly Recurrent Neural Networks.

In this project, we will experiment to generate new pop lyrics based on [this Dataset](#) (<https://www.kaggle.com/neisse/scrapped-lyrics-from-6-genres>).

Import Libraries

We have to import the libraries we're going to use, the most relevant are:

- **Pandas** in order to manipulate and analyze the data
- **Matplotlib** to visualize the data in graphics
- **Wordcloud** to help us when we are going to generate the next word of a sentence
- **Tensorflow & Keras** will help us in the process of machine learning and keras for deep learning.

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
import string, os
import tensorflow as tf

# keras module for building LSTM
from keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Embedding, Dropout, LSTM, Dense, Bidirectional
from keras.preprocessing.text import Tokenizer
from keras.callbacks import EarlyStopping
from keras.models import Sequential
```

Loading the Dataset

Once the Data was precleaned, it's going to be imported to the cloud. As it's mentioned in the documentation, it's on .csv because is easier to manipulate the data using Pandas library.

In [2]:

```
# Import csv file
df = pd.read_csv('../input/pop-lyrics/Pop.csv')
```

In [3]:

```
# Print first 10 rows to confirm is the right dataset
df.head()
```

Out[3]:

		Unnamed: 0	Artist	Songs	Popularity	ALink	Genre	Genres	SName	SLink
0	0		Adam Lambert	110	1.4	/adam-lambert/	Pop	Pop; Pop/Rock; Rock; RomÃ¢ntico; Dance; Electr...	Whataya Want From Me	/adam-lambert-want-fro...me.htm
1	1		Adam Lambert	110	1.4	/adam-lambert/	Pop	Pop; Pop/Rock; Rock; RomÃ¢ntico; Dance; Electr...	Ghost Town	/adam-lambert-town.htm
2	2		Adam Lambert	110	1.4	/adam-lambert/	Pop	Pop; Pop/Rock; Rock; RomÃ¢ntico; Dance; Electr...	We Are The Champions (feat. Kris Allen & Queen)	/adam-lambert-the-ch...feat-kris-allen-and-queen.htm
3	3		Adam Lambert	110	1.4	/adam-lambert/	Pop	Pop; Pop/Rock; Rock; RomÃ¢ntico; Dance; Electr...	If I Had You	/adam-lambert-you.htm
4	4		Adam Lambert	110	1.4	/adam-lambert/	Pop	Pop; Pop/Rock; Rock; RomÃ¢ntico; Dance; Electr...	Never Close Our Eyes	/adam-lambert-close-our-eyes.htm

Data Cleaning

In this step the dataset is going to be cleaned using Pandas library to remove the columns that are not going to be used and keep only the unique column called "Lyrics".

DATA CLEANING CHECKLIST

Up-to-date data



Data should be up-to-date in order to obtain maximum value from the data analysis.



Missing values



Count missing values and analyze where in the data they are missing. Missing values can disrupt some analyses and skew the results.



Duplicates



Duplicate IDs indicate multiple records for one person, e.g. someone holds multiple functions at the same time.



Numerical outliers



Numerical outliers are fairly easy to detect and remove. Define minimum and maximum to spot outliers easily.



Check IDs



Check data labels of all the fields to see whether some categorical values are mislabeled.



Define valid output



Define valid data labels for categorical data. Define data ranges for numerical variables. Non-matching data is presumably wrong.



In [4]:

```
# Drop all the columns except Lyrics
df.drop(['Artist', 'Songs', 'Popularity', 'Genre', 'Genres', 'Idiom', 'ALink',
         'SName', 'SLink'], axis=1, inplace=True)
df.drop(df.columns[df.columns.str.contains('unnamed', case = False)], axis = 1, inplace = True)
```

In [5]:

```
# Print first 10 rows to confirm the drops were made correctly
df.head()
```

Out[5]:

	Lyric
0	Hey, slow it down. What do you want from me. W...
1	Died last night in my dreams. Walking the stre...
2	I've paid my dues. Time after time. I've done ...
3	So I got my boots on,. Got the right amount of...
4	I wish that this night would never be over. Th...

Shaping the Dataset

Since kaggle does not have enough memory (16gb RAM), disk usage (73 GB) and GPU (13GB) to process the entire dataset, we will have to limit our model from 28441 to 700 lyrics. Kaggle give It's worth mentioning that this same test was performed locally and took approximately 100 hours to perform.

In [6]:

```
# Dataframe shape to show how many columns and rows the dataframe have  
df.shape
```

Out[6]:

```
(28441, 1)
```

In [7]:

```
# Take first 700 rows  
df = df[:700]
```

In [8]:

```
df.head()
```

Out[8]:

	Lyric
0	Hey, slow it down. What do you want from me. W...
1	Died last night in my dreams. Walking the stre...
2	I've paid my dues. Time after time. I've done ...
3	So I got my boots on,. Got the right amount of...
4	I wish that this night would never be over. Th...

In [9]:

```
# Dataframe shape to show how many columns and rows the dataframe have a  
fter last cell  
df.shape
```

Out[9]:

```
(700, 1)
```

Now, to get Statistical information, the next cell is going to calculate the number of words per row, this is going to help us to determine the Frequency Distribution of number of words for each text extracted

In [10]:

```
df['Number_of_words'] = df['Lyric'].apply(lambda x:len(str(x).split()))
df.head()
```

Out[10]:

	Lyric	Number_of_words
0	Hey, slow it down. What do you want from me. W...	314
1	Died last night in my dreams. Walking the stre...	224
2	I've paid my dues. Time after time. I've done ...	153
3	So I got my boots on,. Got the right amount of...	365
4	I wish that this night would never be over. Th...	295

Statistical information

In [11]:

```
df['Number_of_words'].describe()
```

Out[11]:

```
count    700.000000
mean     248.065714
std      97.323252
min      31.000000
25%     178.000000
50%     233.000000
75%     309.000000
max     589.000000
Name: Number_of_words, dtype: float64
```

- **count**: Number of rows to evaluate
- **mean**: Average of words per row in the dataset
- **std**: Word's standard deviation
- **min**: Minimum amount of words found in a lyric or row
- **max**: Maximum amount of words found.

In [12]:

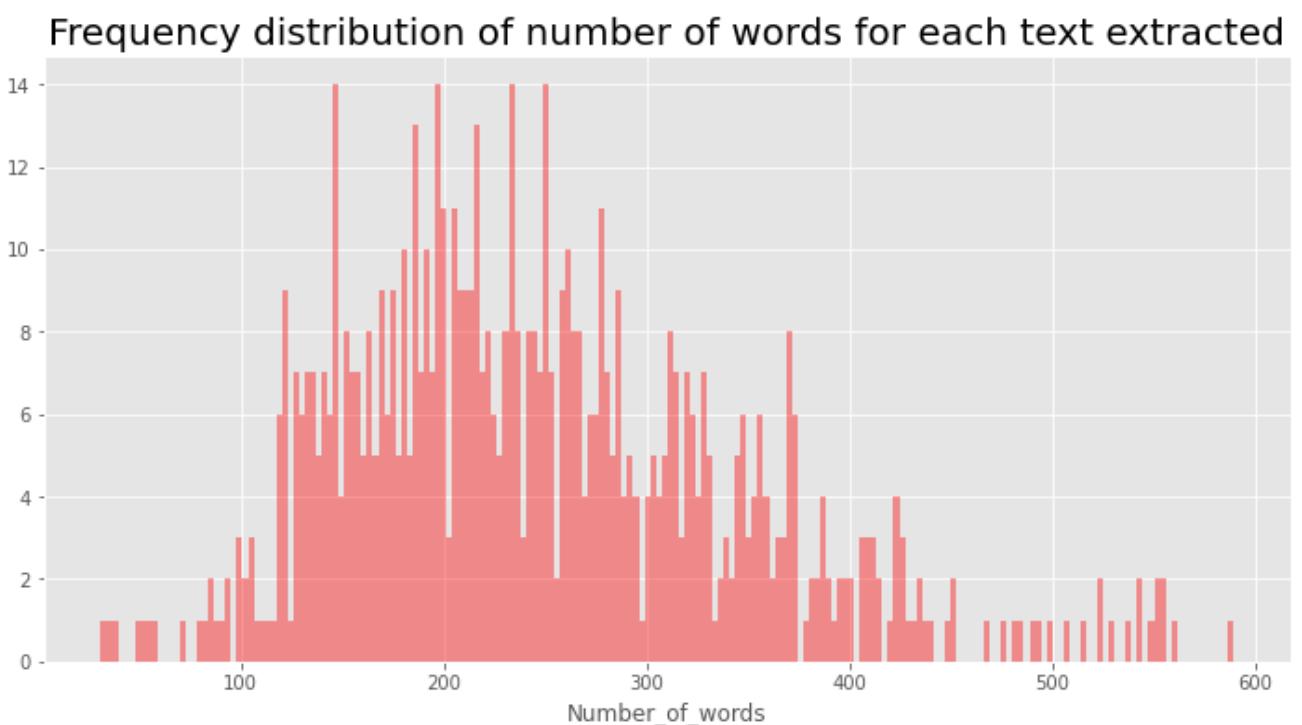
```
import matplotlib.pyplot as plt
plt.style.use('ggplot')
plt.figure(figsize=(12,6))
sns.distplot(df['Number_of_words'],kde = False,color="red",bins=200)
plt.title("Frequency distribution of number of words for each text extracted", size=20)
```

```
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
```

```
    warnings.warn(msg, FutureWarning)
```

Out[12]:

```
Text(0.5, 1.0, 'Frequency distribution of number of words for each text extracted')
```



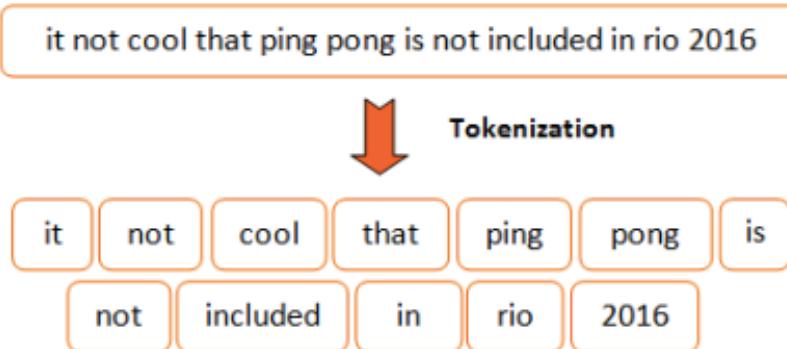
- **Y axis:** Represent how many times a word was found in the dataset
- **X axis:** Represent the amount of different words found;

Tokenization

The data from the column named **Lyric** must be preprocessed and to achieve these, the words will have to be converted into numbers. This process is called "tokenization".

Keras uses the class **Tokenizer()** to do this job, this class has two important methods to be used:

- **_fit_on_text()** : Update the internal vocabulary based on a given list of texts or in this case, "Lyrics" column, where each entry of the list is going to be a token.
- **_texts_to_sequences()** : Transform each text within the list of texts supplied to a sequence of integers; only words known by the tokenizer will be considered.



In [13] :

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(df['Lyric'].astype(str).str.lower())

total_words = len(tokenizer.word_index)+1
tokenized_sentences = tokenizer.texts_to_sequences(df['Lyric'].astype(str))
tokenized_sentences[0]
```

Out[13] :

```
[128,
 338,
 9,
 48,
```

31,
33,
1,
45,
72,
6,
31,
33,
1,
45,
72,
6,
35,
13,
459,
31,
33,
1,
45,
72,
6,
31,
33,
1,
45,
72,
6,
75,
387,
50,
95,
7,
43,
2,
108,
88,
232,

70,
202,
368,
653,
7,
43,
2,
290,
88,
7,
1023,
23,
40,
80,
25,
52,
18,
31,
33,
1,
45,
72,
6,
31,
33,
1,
45,
72,
6,
28,
16,
88,
32,
13,
1073,
9,
49,

139,
16,
88,
10,
2,
92,
66,
1,
48,
9,
1878,
6,
32,
69,
7,
784,
4,
372,
28,
125,
204,
112,
128,
31,
33,
1,
45,
72,
6,
31,
33,
1,
45,
72,
6,
35,
26,

1074,
4,
61,
11,
47,
39,
379,
5,
26,
124,
193,
42,
1,
26,
6,
2456,
13,
7,
3010,
23,
2116,
22,
832,
6,
109,
39,
527,
9,
2117,
75,
387,
50,
95,
7,
43,
30,
2,

108,
66,
1,
506,
70,
2,
593,
198,
155,
23,
2,
82,
1,
85,
340,
8,
81,
28,
16,
88,
32,
13,
1073,
9,
49,
139,
16,
88,
10,
2,
92,
66,
1,
48,
9,
1878,
6,

32,
69,
7,
784,
4,
372,
28,
125,
204,
112,
128,
31,
33,
1,
45,
72,
6,
31,
33,
1,
45,
72,
6,
28,
16,
88,
32,
14,
6,
2,
92,
66,
1,
48,
34,
2,
92,

66,
1,
48,
18,
28,
16,
88,
32,
13,
1073,
9,
49,
139,
16,
88,
10,
2,
92,
66,
1,
48,
9,
1878,
6,
32,
69,
7,
784,
4,
372,
28,
125,
204,
112,
128,
31,
33,

1,
45,
72,
6,
28,
16,
88,
32,
13,
1073,
9,
49,
139,
16,
88,
10,
2,
92,
66,
1,
48,
9,
1878,
6,
32,
69,
7,
784,
4,
372,
28,
125,
204,
112,
128,
2118,
45,

```
72,  
6,  
2118,  
45,  
72,  
6,  
2118,  
45,  
72,  
6,  
2118,  
45,  
72,  
6]
```

Slash sequences into a n-gram sequence

After the text is tokenized, the words will be sorted to represent them numerically by creating an input sequence using the created tokens.

In [14]:

```
input_sequences = list()  
for i in tokenized_sentences:  
    for t in range(1, len(i)):  
        n_gram_sequence = i[:t+1]  
        input_sequences.append(n_gram_sequence)
```

Padding

Before the model generation, is necessary to normalize all sentences to the same standard lenght, to avoid the memory overflow and to get the layers of the model way more deep, this is a simple process to add a 0's in the beginning of the text, resulting in layers of the same size.

_padsequences Transform a list of sequences that is a lists of integers into a 2D array, in this case the list is called input_sequences.

Sequences resulting shorter than **maxlen** are padded with 0's until they have the same length.

The position where the zeros will be added is determined by the argument **padding**, in this case, it will be done at the beginning of the sequence.

sequence before padding

```
[21, 4, 2, 12, 22, 23, 13, 2, 24, 6, 2, 7, 2, 4, 25],  
[ 2, 26, 7, 27, 14, 9, 1, 4, 28 ],  
[15, 25, 1, 29, 6, 15, 30 ],  
[ 1, 16, 17, 27, 30, 1, 5, 2 ],  
[31, 2, 28, 6, 32, 9, 33 ]]
```

sequence after padding
(padding and truncate in front/pre)

```
[23, 13, 2, 24, 6, 2, 7, 2, 4, 25],  
[ 0, 2, 26, 7, 27, 14, 9, 1, 4, 28 ],  
[ 0, 0, 0, 15, 25, 1, 29, 6, 15, 30 ],  
[ 0, 0, 1, 16, 17, 27, 30, 1, 5, 2 ],  
[ 0, 0, 0, 31, 2, 28, 6, 32, 9, 33 ],
```

MAX_SEQUENCE_LENGTH = 10

In [15]:

```
max_sequence_len = max([len(x) for x in input_sequences])  
input_sequences = np.array(pad_sequences(input_sequences, maxlen=max_se  
quence_len, padding='pre'))
```

In [16]:

```
input_sequences[:20]
```

Out[16]:

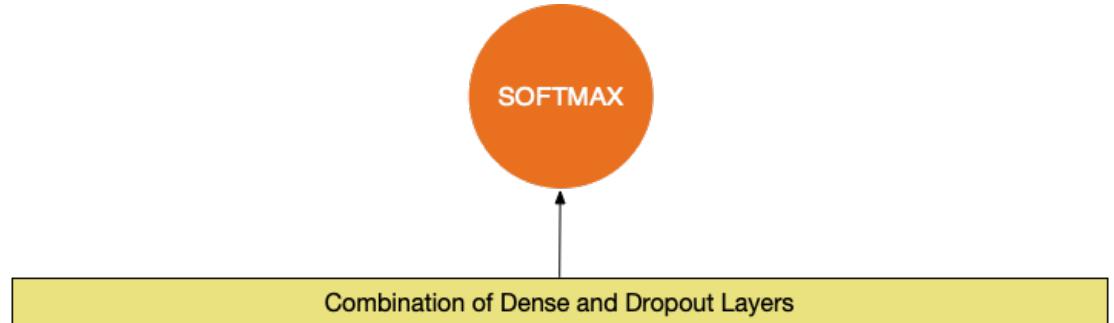
```
array([[ 0,  0,  0, ...,  0, 128, 338],
       [ 0,  0,  0, ..., 128, 338,   9],
       [ 0,  0,  0, ..., 338,   9,  48],
       ...,
       [ 0,  0,  0, ..., 35, 13, 459],
       [ 0,  0,  0, ..., 13, 459, 31],
       [ 0,  0,  0, ..., 459, 31, 33]], dtype=int32)
```

In [17]:

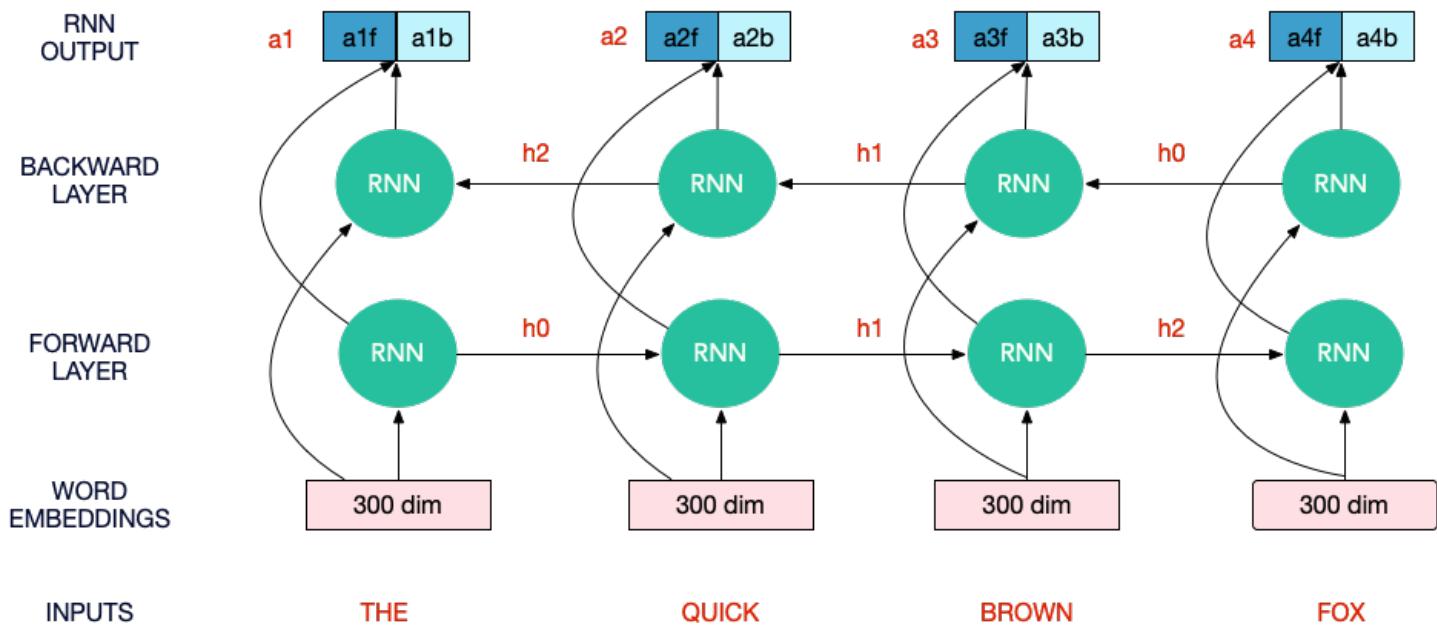
```
# create predictors and label
X, labels = input_sequences[:, :-1], input_sequences[:, -1]
y = tf.keras.utils.to_categorical(labels, num_classes=total_words)
```

Creating the Model

This test will use the Bidirectional LSTM model, this kind of neural networks run as the name says: in two ways. This is from past to future and vice versa, this is how the model preserves the information of both states at any point. LSTM neural networks are mostly used where context is involved.



Dimension of a_i = size of hidden state vector h
In our code we define h dimension as 64



In []:

```
model = Sequential()
```

Models in Keras are defined as a sequence of layers, and the Sequential model is about adding layers one at a time. Layers are the basic building block of neural network.

Layers:

Embedding

- Is a core layer, can only be used as the first layer in a model, turns positive integers into dense vectors of fixed size (first parameter is the size of the vocabulary, second parameter is the dimension of the dense embedding, and third parameter is about the length of sequences which is required because we will use a Dense layer later)

In []:

```
model.add(Embedding(total_words, 40, input_length=max_sequence_len-1))
```

Bidirectional:

- Is a recurrent layer, a bidirectional wrapper for RNN's which will receive a layer as an input being LSTM layer the one we chose, it will receive a positive integer as its input which refers to the amount of output nodes that should be returned.

In []:

```
model.add(Bidirectional(LSTM(250)))
```

Dropout:

- Is a regularization layer. This layer randomly sets input units to 0, with a frequency of the value we pass it, at each step during training time which helps prevent overfitting.

In []:

```
model.add(Dropout(0.1))
```

Dense:

- Is a core layer, and a densely-connected neural network layer. Receives as first parameter a positive integer which refers to the amount of output nodes that should be returned. Second parameter is the one named **activation** which defines the type of predictions the model can make; for the kind of problem we are abording the one which suits the better is softmax which outputs a vector of values (input) that can be interpreted as probabilities of being used.

In []:

```
model.add(Dense(total_words, activation='softmax'))
```

Compile method.

Takes two relevant parameters:

- **loss**: Also known as a cost function; works during the optimization process and it's role is to calculate the error of the model. Cross-entropy is used to estimate the difference between an esimated and predicted probability distributions. categorical_crossentropy will be used because it's best suited for this kind of problems and is almost universally used to train deep learning neural networks due to the results it produces.
- **optimizer**: Is responsible for reducing the losses and to provide the most accurate results possible. Adam is the option chosen because is the best choice offered by Keras to train the neural network in less time and more efficiently. Earlystop will stop the training if the model has stopped improving, this will be checked at the end of every epoch.

Optimization algorithm and Performance metrics

The optimization algorithm added in the configuration layer will be **Adam**, because of the good performance and results in other projects it has.

In this case, the 'accuracy' will be measured as performance metric, which gives closeness of calculated value to the actual value.

In []:

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

fit method trains the model for the fixed number of epochs given (iterations on the dataset).

In []:

```
earlystop = EarlyStopping(monitor='loss', min_delta=0, patience=3, verbose=0, mode='auto')
history = model.fit(X, y, epochs=20, verbose=1, callbacks=[earlystop])
```

Evaluating the Model

In this block a new graphic will be generated and displayed, the Y axis will stand for accuracy and X axis will stand for the amount of epochs; as shown, in order to increase the accuracy, you have to increase the number of epochs during the training.

In []:

```
# plot the accuracy

plt.plot(history.history['accuracy'], label='train acc')
plt.legend()
plt.show()
plt.savefig('AccVal_acc')
```

Import the Trained Model

Once the training process is completed it only remains the import of our model so we can test how does it work, in our case the trained model is called 'song_lyrics_generator'.

In [18]:

```
import matplotlib.pyplot as plt
import seaborn as sns
from keras.models import load_model
model = load_model('../input/songlyricmodel/song_lyrics_generator.h5')
```

Function to Generate the song

This step is in charge of preparing the function that will be used to complete a song given the model previously trained, it will predict the next words based on the input words suministrated as 'seed_text'. For this to work a tokenization must be applied to the seed_text, then a padding will be applied to the sequences generated and passed into the trained model so the next word can be predicted.

In [19]:

```
def complete_this_song(seed_text, next_words):  
    for _ in range(next_words):  
        token_list = tokenizer.texts_to_sequences([seed_text])[0]  
        token_list = pad_sequences([token_list], maxlen=max_sequence_length-1, padding='pre')  
        predicted = model.predict_classes(token_list, verbose=0)  
        output_word = ""  
        for word, index in tokenizer.word_index.items():  
            if index == predicted:  
                output_word = word  
                break  
        seed_text += " " + output_word  
    return seed_text
```

Examples

In [20]:

```
complete_this_song("I am missing you", 200)
```

```
/opt/conda/lib/python3.7/site-packages/tensorflow/python/keras/engine/s  
equential.py:450: UserWarning: `model.predict_classes()` is deprecated  
and will be removed after 2021-01-01. Please use instead:  
* `np.argmax(model.predict(x), axis=-1)`  
, if your model does multi-class classification  
(e.g. if it uses a `softmax` last-layer activation).  
* `(model.predict(x) > 0.5).astype("int32")`  
, if your model does binary classification  
(e.g. if it uses a `sigmoid` last-layer activation).  
warnings.warn(``model.predict_classes()` is deprecated and '
```

Out[20]:

"I am missing you i believe in the things you were so far away from me
i could not see the words that i was so cold and i never need to know y
our love is good i don't know how to tell you i'm not a fortune teller
i never change but i want you to stay i don't know what to tell you i'm
not a fortune teller i never change but i want you to stay i don't know
what to tell you i'm not a fortune teller i never change but i want you
to stay i don't know what to tell you i'm not a fortune teller i never
change but i want you to stay i don't know what to tell you i'm not a f
ortune teller i never change but i want you to stay i don't know what t
o tell you i'm not a fortune teller i never change but i want you to st
ay i don't know what to tell you i'm not a fortune teller i never chang
e but i want you to stay i don't know what to tell you i'm not a fortun
e teller i never change but i want you to"

In [21]:

```
complete_this_song("It's a cruel and random world", 80)
```

Out[21]:

"It's a cruel and random world is not much between how how could you do
and i see your life is a ghost and my heart is a ghost town my heart is
a ghost town my heart is a ghost town my heart is a ghost town my heart
is a ghost town my heart is a ghost town my heart is a ghost town my he
art is a ghost town my heart is a ghost town my heart is a ghost town m
y heart"

In [22]:

```
complete_this_song("I want a piece of pizza", 80)
```

Out[22]:

"I want a piece of pizza your eyes the bad are burning hot the sea you know you feel it up you are calling to you you look at me tonight you think of me of us how how how do how do we do how or now and now you couldn't see the ocean of hate your life you reply if"

In [23]:

```
complete_this_song("Love flowers", 80)
```

Out[23]:

"Love flowers get to her love and you make it okay but i'm not here i still still make you wanna be the one for you ooh oh oh oh baby baby baby baby baby baby baby baby you're gonna get you home and you go standing in your way and i don't want to miss a thing because i'm happy clap along if you know what happiness is is enough for me and you can do i know that"

In [24]:

```
complete_this_song("Hate flowers", 80)
```

Out[24]:

"Hate flowers love me love me love me how do you do you know what you do you love me if you love me if you love me love you don't want me to me and maybe i want you to trust me again i want to be your fantasy maybe i'm gonna be the one who understands you through me i know you understand me i want you to take me back to find to take to take you the sweetest"

Bibliografía

- [1] Real Academia Española (2020, noviembre), Diccionario de la lengua española, [En línea]. Disponible: <https://dle.rae.es> [Último acceso: 10 de diciembre del 2020].
- [2] Oxford Lexico (2020, noviembre), Definitions, Meanings, Synonyms, and Grammar by Oxford, [En línea]. Disponible: <https://www.lexico.com> [Último acceso: 2 de diciembre del 2020].
- [3] J. A. Gutiérrez Orozco, Escuela Superior de Cómputo (2008, septiembre 15), Máquinas de Estados Finitos, [En línea]. Disponible: http://delta.cs.cinvestav.mx/~mcintosh/cellularautomata/Summer_Research_files/maquinasef.pdf [Último acceso: 15 de diciembre del 2020].
- [4] O. Jaramillo, Universidad Nacional Autonoma de México (2007, mayo 03), El concepto de Sistema, [En línea]. Disponible: <https://www.ier.unam.mx/~ojs/pub/Termodinamica/node9.html> [Último acceso: 2 de diciembre del 2020].
- [5] Chaney, D. (2012). The Music Industry in the Digital Age: Consumer Participation in Value Creation. International Journal of Arts Management, (1), pp. 15.
- [6] Sutskever, I., Martens, J., Hinton, G. E. (2011, January). Generating text with recurrent neural networks. In ICML.
- [7] Kim, Y. (2014). Convolutional neural networks for sentence classification. arXiv preprint arXiv:1408.5882.
- [8] Monteith, K., Martinez, T. R., & Ventura, D. (2012, May). Automatic Generation of Melodic Accompaniments for Lyrics. In ICCC, pp. 87-94.
- [9] Nikolov, N. I., Malmi, E., Northcutt, C. G., Parisi, L. (2020). Conditional Rap Lyrics Generation with Denoising Autoencoders. arXiv preprint arXiv:2004.03965.

- [10] Baker, F. A. (2015). What about the music? Music therapists' perspectives on the role of music in the therapeutic songwriting process. *Psychology of Music*, 43(1), pp. 122-139.
- [11] Guerrero, J. (2012). El género musical en la música popular: algunos problemas para su caracterización. *Trans. Revista transcultural de música*, (16), pp. 1-22.
- [12] Wang, A., & Cho, K. (2019). Bert has a mouth, and it must speak: Bert as a markov random field language model. *arXiv preprint arXiv:1902.04094*.
- [13] Honnibal, M.,& Montani, I. (2017). spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing.
- [14] V. Advani (2021, febrero 11), What is Artificial Intelligence? How does AI work, Types and Future of it?, [En línea]. Disponible:<https://www.mygreatlearning.com/blog/what-is-artificial-intelligence/> [Último acceso: 6 de abril del 2021].
- [15] IBM Corporation (2021, marzo 31), Artificial Intelligence (AI), [En línea]. Disponible: <https://www.ibm.com/cloud/learn/what-is-artificial-intelligence> [Último acceso: 6 de abril del 2021].
- [16] L. Hardesty (2017, abril), Explained: Neural networks, [En línea]. Disponible: <https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414> [Último acceso: 15 de noviembre del 2020].
- [17] amBientech (2019, julio 30), ¿Qué es la neurona?, [En línea]. Disponible: <https://ambientech.org/la-neurona> [Último acceso: 6 de abril del 2021].
- [18] National Cancer Institute (2021), Neurons & Glial Cells, [En línea]. Disponible: <https://training.seer.cancer.gov/brain/tumors/anatomy/neurons.html> [Último acceso: 30 de abril del 2021].
- [19] C. Gershenson (2012, octubre), Artificial Neural Networks for Beginners, [En línea]. Disponible: <https://www.uv.mx/mia/files/2012/10/Artificial-Neural-Networks-for-Beginners.pdf> [Último acceso: 6 de abril del 2021].
- [20] Rubik's Code (2018, febrero), How do Artificial Neural Networks learn?, [En línea]. Disponible: <https://rubikscode.net/2018/01/15/how-artificial-neural-networks-learn/> [Último acceso: 7 de abril del 2021].

- [21] S. Leijnen, F. van Veen (2020, mayo), The Neural Network Zoo, [En línea]. Disponible: https://www.researchgate.net/publication/341373030_The_Neural_Network_Zoo [Último acceso: 20 de noviembre del 2020].
- [22] IBM Cloud Education (2020, septiembre 14), What are Recurrent Neural Networks?, [En línea]. Disponible: <https://www.ibm.com/cloud/learn/recurrent-neural-networks> [Último acceso: 26 de marzo del 2021].
- [23] ref IBM Cloud Education (2020, julio 2), Natural Language Processing [En línea]. Disponible: <https://www.ibm.com/cloud/learn/natural-language-processing> [Ultimo acceso: 20 de abril del 2021].
- [24] ref A. Vaca (2020, mayo), Transformers en Procesamiento del Lenguaje Natural, [En línea]. Disponible: <https://www.iic.uam.es/innovacion/transformers-en-procesamiento-del-lenguaje-natural/> [Ultimo acceso: 15 de abril del 2021].
- [25] ref A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser & I. Polosukhin (2017, diciembre), Attention Is All You Need, [En línea]. Disponible: <https://arxiv.org/pdf/1706.03762.pdf> [Ul-timo acceso: 15 de abril del 2021].
- [26] ref J. Devlin, M. Chang, K. Lee, K. Toutanova (2019, mayo), BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, [En línea]. Disponible: <https://arxiv.org/pdf/1810.04805.pdf> [Ul-timo acceso: 11 de abril del 2021].
- [27] ref R. Horev (2018, noviembre), BERT Explained: State of the art language model for NLP, [En línea]. Disponible: <https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270> [Ultimo acceso: 11 de abril del 2021].
- [28] ref B. Lutkevich (2020, enero), BERT language model, [En línea]. Disponible: <https://searchenterpriseai.techtarget.com/definition/BERT-language-model> [Ultimo acceso: 12 de abril del 2021].
- [29] ref A. Radford, J. Wu, D. Amodei, D. Amodei, J. Clark, M. Brundage & I. Sutskever (2020, enero), Better Language Models and Their Implications, [En línea]. Disponible: <https://openai.com/blog/better-language-models/> [Ultimo acceso: 14 de abril del 2021].

- [30] ref M. Edward (2019, febrero), Too powerful NLP model (GPT-2), [En línea]. Disponible: <https://towardsdatascience.com/too-powerful-nlp-model-generative-pre-training-2-4cc6afb6655> [Ultimo acceso: 14 de abril del 2021].
- [31] ref A. Radford, K. Narasimhan, T. Salimans & I. Sutskever (2018), Improving Language Understanding by Generative Pre-Training, [En línea]. Disponible: https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf [Ultimo acceso: 14 de abril del 2021].
- [32] ref J. Montantes(2019, mayo), Examining the Transformer Architecture, [En línea]. Disponible: <https://towardsdatascience.com/examining-the-transformer-architecture-part-1-the-openai-gpt-2-controversy-feceda4363bb> [Ultimo acceso: 14 de abril del 2021].
- [33] ref J. Vig(2019, marzo), GPT-2: Understanding Language Generation through Visualization, [En línea]. Disponible: <https://towardsdatascience.com/openai-gpt-2-understanding-language-generation-through-visualization-8252f683b2f8> [Ultimo acceso: 14 de abril del 2021].
- [34] Oracle México (2020, noviembre), ¿Qué es una base de datos?, [En línea]. Disponible: <https://www.oracle.com/mx/database/what-is-database/> [Último acceso: 20 de noviembre del 2020].
- [35] Escribir Canciones (2008), Estructura y elementos de una canción, [En línea]. Disponible: <https://dle.rae.es> [Último acceso: 2 de diciembre del 2020].
- [36] Swing this Music (2008), ¿QUÉ SECCIONES PUEDE TENER UNA CANCIÓN?española, [En línea]. Disponible: <https://sites.google.com/site/swingthismusiccast/interpretacio/estructura-cancion/secciones-de-una-cancion> [Último acceso: 2 de diciembre del 2020].
- [37] J. R. Norris (1997), Markov Chains, [En línea]. Disponible: <https://cape.fcfm.buap.mx/jdzf/cursos/procesos/libros/norris.pdf> [Último acceso: 15 de diciembre del 2020].
- [38] NGINx (2016, mayo 17), What is NGINX?, [En línea]. Disponible: <https://www.nginx.com/resources/glossary/nginx/> [Último acceso: 27 de mayo del 2021].

- [39] Mozilla.org (2021, febrero 19), HTML basics, [En línea]. Disponible: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics [Último acceso: 15 de mayo del 2021].
- [40] Mozilla.org (2021, mayo 14), HTML5, [En línea]. Disponible: <https://developer.mozilla.org/es/docs/Web/Guide/HTML/HTML5> [Último acceso: 15 de mayo del 2021].
- [41] W3C (2016), HTML & CSS, [En línea]. Disponible: [Último acceso: 15 de mayo del 2021].
- [42] Mozilla.org (2021, abril 27), What is JavaScript?, [En línea]. Disponible: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript [Último acceso: 15 de mayo del 2021].
- [43] OpenSSL (2018), OpenSSL, [En línea]. Disponible: <https://www.openssl.org> [Último acceso: 15 de mayo del 2021].
- [44] Documentation Group. (n.d.). Welcome! - The Apache HTTP Server Project. Apache. Retrieved April 6, 2021, from <https://httpd.apache.org/>
- [45] G., D. (2021, March 9). What is Apache? An In-Depth Overview of Apache Web Server. Hostinger Tutorials. <https://www.hostinger.com/tutorials/what-is-apache>
- [46] What is a web server? - Learn web development — MDN. (2021, January 27). MDN Web Docs. https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server
- [47] What is SSL? (2017, May 15). SSLSHOPPER. <https://www.sslshopper.com/what-is-ssl.html>
- [48] Verisign. (2015, August 30). ¿Qué es un certificado SSL? – Verisign. Verisign NameStudio. https://www.verisign.com/es_LA/website-presence/online/ssl-certificates/index.xhtml
- [49] Bavati, I. (2020, September 22). Data Science in the Cloud - Towards Data Science. Medium. <https://towardsdatascience.com/data-science-in-the-cloud-239b795a5792> (September 2020)

- [50] G. (2021, March 5). Comparing Google Cloud Platform, AWS and Azure - Georgian Impact Blog. Medium. <https://medium.com/georgian-impact-blog/comparing-google-cloud-platform-aws-and-azure-d4a52a3adbd2>
- [51] I., A. (2015, October 2). Jupyter/IPython Notebook Quick Start Guide. Jupyter Notebook Beginner Guide. https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html
- [52] Jones, E. (2021, March 25). Google Cloud vs AWS in 2021 (Comparing the Giants). Kinsta. <https://kinsta.com/blog/google-cloud-vs-aws/>
- [53] Copyright.gov (2021), Copyright in General, [En línea]. Disponible: <https://www.copyright.gov/help/faq/faq-general.html#what> [Último acceso: 04 de mayo del 2021].
- [54] World Intellectual Property Organization (2007), THE ARTS AND COPYRIGHT, [En línea]. Disponible: https://www.wipo.int/edocs/pubdocs/en/copyright/935/wipo_pub_935.pdf [Último acceso: 04 de mayo del 2021].
- [55] CÁMARA DE DIPUTADOS DEL H. CONGRESO DE LA UNIÓN (2014, julio 14), LEY FEDERAL DEL DERECHO DE AUTOR, [En línea]. Disponible: https://www.ucol.mx/content/cms/13/file/federal/LEY_FED_DEL_DERECHO_DEL_AUTOR.pdf [Último acceso: 05 de mayo del 2021].
- [56] Aaron Tay (2021, Abril 23), Data Cleaning Techniques [En línea]. Disponible: <https://www.digitalvidya.com/blog/data-cleaning-techniques/> [Último acceso: 30 Mayo 2021]
- [57] Python (2021), What is Python? Executive Summary, [En línea]. Disponible: <https://www.python.org/doc/essays/blurb/> [Último acceso: 24 de abril del 2021].
- [58] MongoDB. (2019), ¿Qué es MongoDB?, [En línea]. Disponible: <https://www.mongodb.com/es>. [Último acceso: 26 Marzo 2021].
- [59] Noteworthy Programming Masterpiece (2019), openssl-nodejs [En línea]. Disponible: <https://www.npmjs.com/package/openssl-nodejs> [Último acceso: 26 Marzo 2021]
- [60] The Apache Software Foundation (2019), Apache Tomcat [En línea]. Disponible: <http://tomcat.apache.org/index.html> [Último acceso: 26 Marzo 2021]

- [61] OneLook Dictionary Search (2016), Privacy Policy [En línea]. Disponible: <https://www.onelook.com/about.shtml> [Último acceso: 19 Mayo 2021]
- [62] MongoDB Inc (2018), MongoDB Licensing [En línea]. Disponible: <https://www.mongodb.com/community/licensing> [Último acceso: 19 Mayo 2021]
- [63] Google Research: BERT (2018), LICENSE [En línea]. Disponible: <https://github.com/google-research/bert/blob/master/LICENSE> [Último acceso: 19 Mayo 2021]
- [64] Apache (2004), The Apache License, Version 2.0 [En línea]. Disponible: <https://httpd.apache.org/docs/current/license.html> [Último acceso: 19 Mayo 2021]
- [65] Facebook: React (2018), LICENSE [En línea]. Disponible: <https://github.com/facebook/react/blob/master/LICENSE> [Último acceso: 19 Mayo 2021]
- [66] Python (2001), History and License [En línea]. Disponible: <https://docs.python.org/3/license.html> [Último acceso: 19 Mayo 2021]
- [67] Elastic (2021), Elastic License 2.0 [En línea]. Disponible: <https://www.elastic.co/es/licensing/elastic-license> [Último acceso: 19 Mayo 2021]
- [68] Google Cloud Platform (2021), Descripción general de Google Cloud [En línea]. Disponible: <https://cloud.google.com/docs/overview> [Último acceso: 25 Mayo 2021]
- [69] Amazon Web Services (2021), Informática en la nube con AWS [En línea]. Disponible: <https://aws.amazon.com/es/what-is-aws/> [Último acceso: 25 Mayo 2021]
- [70] Azure (2021), Conozca Azure [En línea]. Disponible: <https://azure.microsoft.com/es-mx/overview/> [Último acceso: 25 Mayo 2021]
- [71] ITM Platform (2010), Matriz de Evaluación de Riesgos [En línea]. Disponible: <https://www.itmplatform.com/es/recursos/matriz-de-evaluacion-de-riesgos/#RiskSetId=rh25q/6IQWFcfvYIwSUF6A==> [Último acceso: 26 Mayo 2021]

- [72] Kaggle (2010), How to Use Kaggle [En línea]. Disponible: <https://www.kaggle.com/docs/datasets> [Último acceso: 25 Mayo 2021]
- [73] Kaggle (2019, Noviembre 17), Song lyrics from 6 musical genres [En línea]. Disponible: <https://www.kaggle.com/neisse/scrapped-lyrics-from-6-genres> [Último acceso: 25 Mayo 2021]
- [74] Google (2018, Septiembre 5), Dataset Search [En línea]. Disponible: <https://datasetsearch.research.google.com> [Último acceso: 25 Mayo 2021]
- [75] Amazon, Open Data on AWS [En línea]. Disponible: <https://aws.amazon.com/es/opendata/?wwps-cards.sort-by=item.additionalFields.sortDate&wwps-cards.sort-order=desc> [Último acceso: 25 Mayo 2021]
- [76] Music4All (2020), A New Music Database and Its Applications [En línea]. Disponible: <https://sites.google.com/view/contact4music4all> [Último acceso: 25 Mayo 2021]
- [77] Vagalume (2002), Vagalume: Music e tudo [En línea]. Disponible: <https://www.vagalume.com.br/> [Último acceso: 25 Mayo 2021]
- [78] TopTenAI (2020, Mayo 9), Top 10 Lyric Generator Review 2021 [En línea]. Disponible: <https://topten.ai/lyric-generator-review/> [Último acceso: 30 Mayo 2021]
- [79] Kingma, D. P., Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.
- [80] Sanjeevi, M. (2018, June 21). Chapter 10.1: DeepNLP — LSTM (Long Short Term Memory) Networks with Math. Medium. <https://medium.com/deep-math-machine-learning-ai/chapter-10-1-deepnlp-lstm-long-short-term-memory-networks-with-math-21477f8e4235>
- [81] Foote, K. D. (2019, June 17). A Brief History of Natural Language Processing (NLP). DATAVERSITY. <https://www.dataversity.net/a-brief-history-of-natural-language-processing-nlp/>
- [82] Amazon (2021), Amazon EC2 [En línea]. Disponible: <https://aws.amazon.com/es/ec2/?ec2-whats-new.sort-by=item.additionalFields.postDateTime&ec2-whats-new.sort-order=desc> [Último acceso: 1 Junio 2021]

- [83] Amazon (2021), Amazon SageMaker [En línea]. Disponible: <https://aws.amazon.com/es/sagemaker/> [Último acceso: 1 Junio 2021]
- [84] Amazon (2021), Amazon S3 [En línea]. Disponible: <https://aws.amazon.com/es/s3/> [Último acceso: 1 Junio 2021]
- [85] Bansal, S. (2018, May 27). Language Modelling and Text Generation using LSTMs — Deep Learning for NLP. Medium. <https://medium.com/@shivambansal36/language-modelling-text-generation-using-lstms-deep-learning-for-nlp-ed36b224b275>
- [86] B. Lee (2019, Noviembre 17), How to 10x Response Rates on Surveys [En línea]. Disponible: <https://bryankmlee3.medium.com/conducting-surveys-with-nlp-d38df4c29e39> [Último acceso: 5 Junio 2021]