UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

APLICACIÓN DE REDES NEURONALES PARA SISTEMAS DE RECOMENDACIÓN USANDO AUTOCODIFICADORES

TESIS
PARA OBTENER EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

PRESENTA:

FERNANDO ÁVALOS GARCÍA

DIRECTOR DE TESIS:

M.T.C.A. MOISÉS EMMANUEL RAMÍREZ GUZMÁN

CO-DIRECTOR DE TESIS:

M.T.C.A. ERIK GERMÁN RAMOS PÉREZ

Huajuapan de León, Oaxaca. Septiembre de 2019.

Índice general

Resumen

Los sistemas de recomendación se han desarrollado desde finales de los años 90s y al día de hoy se siguen encontrando nuevas formas de implementarlos en diversas industrias como son comercio electrónico, turismo, redes sociales, por mencionar algunas. Cada industria tiene sus propias necesidades y la información con la que cuenta es diferente, es por eso que existe gran cantidad de investigación desarrollando nuevas técnicas para cubrir estas necesidades.

Una de las tecnologías en investigación para el desarrollo de sistemas de recomendación es el aprendizaje automático. El aprendizaje automático contiene algoritmos que permiten identificar patrones, tendencias, y relaciones que se encuentran en los datos. Existen diversos algoritmos de aprendizaje automático y gracias a la capacidad de cómputo con que se cuenta hoy en día las arquitecturas profundas están revolucionando el estado del arte en varios aspectos como identificación y localización de objetos en imágenes, traducción de textos, conversión de audio a texto, entre muchas otras aplicaciones. Entre las arquitecturas profundas se encuentran los auto-codificadores, estos permiten generar una representación de los datos de entrada generalmente en una dimensión menor, y a partir de esta representación generar una salida similar a la entrada.

En el presente trabajo se realizó la implementación de un algoritmo de sistemas de recomendación utilizando auto-codificadores con arquitecturas profundas. Se analizaron diferentes arquitecturas de redes neuronales hasta encontrar una configuración que genera los resultados esperados. Se realizó el entrenamiento de la red utilizando el conjunto de datos JESTER y se analizaron los resultados. De la misma forma, se realizó la implementación del método *Recall* para la validación de las recomendaciones generadas y los resultados se comparan con implementaciones realizadas anteriormente con el mismo conjunto de datos.

Por ultimo, se sugieren implementaciones o mejoras que se pueden realizar a futuro.

Capítulo 1

Introducción

Los sistemas de recomendación representan una parte esencial para las empresas de diversos sectores. Actualmente, empresas como Netflix, Google, Youtube, Facebook, Amazon, etc., ejecutan continuamente algoritmos de recomendación para persuadir al usuario a visitar y/o realizar alguna compra o utilizar algún servicio a través de su sitio.

Existen diferentes tipos de sistemas de recomendación, algunos ejemplos de las aplicaciones son: SmartCity es un sistema que crea un plan para visitar puntos de interés y eventos interesantes basado en las preferencia personales del perfil de usuario [Luberg et al., 2011]. PORE es un sistema de recomendación para bibliotecas digitales basado en ontologías personalizadas [Liao et al., 2009]. Sistemas de recomendación que facilitan la comprensión del estado de salud de una persona basado en su historial médico como es el caso de CoHRA (Collabarative Health Recommender System) [Wiesner and Pfeifer, 2014] o sistemas de recomendación de problemas de programación como Hacker Rank [1], UVa Hunting[2] o Jutge.org[3] [Caro Martínez, 2017].

Recientemente se han desarrollado diferentes arquitecturas y modelos en sistemas de recomendación basados en técnicas de aprendizaje computacional. Como resultado de este avance se ha visto un número creciente de publicaciones académicas y por parte de grupos de investigación del sector privado utilizando diferentes técnicas de redes neuronales para el análisis de los datos y la generación de recomendaciones que buscan ser más

[1] `https://www.hackerrank.com`
[2] `https://uhunt.onlinejudge.org/`
[3] `https://jutge.org/`

eficientes y mejorando la exactitud [Zhang et al., 2019]. Las principales técnicas de redes neuronales usadas son: Perceptrón Multicapa (MLP, del inglés *Multilayer Perceptron*), Autocodificadores (*Autoencoders*), Redes Neuronales Convolucionales (CNN, del inglés *Convolutional Neural Network*), Redes Neuronales Recurrentes (RNN, del inglés *Recurrent Neural Network*), Modelos de Similaridad Semántica Profunda (del inglés *Deep Semantic Similarity Model*), Máquinas restringidas de Boltzman (RBM, del inglés *Restricted Boltzmann Machine*), Estimación Neuronal de Distribución (ANADE, del inglés *Autoregresiva Neural Autoregressive Distribution Estimation*) y Redes Generativas de Adversarios (GAN, del inglés *Generative Adversarial Network*).

Los sistemas de clasificación utilizan diferentes tecnologías pero estas se pueden clasificar en 2 grandes grupos:

- Sistemas de recomendación basados en contenido [Jafarkarimi et al., 2012], utilizan los atributos almacenados del perfil de usuario para establecer una relación con los atributos de los productos y/o servicios, de esta forma se realizan recomendaciones al usuario de nuevos productos.
- Sistemas de recomendación basados en filtros colaborativos [Trivago, 2005a], estos sistemas capturan y analizan grandes cantidades de información relacionados a la actividad, preferencia y comportamiento de los usuarios realizando una predicción basada en la similitud con otros usuarios.

Los sistemas de recomendación basados en contenido pueden caracterizar cada usuario, pero los sistemas de filtros colaborativos tienen algunas ventajas [Melville et al., 2002]. En primer lugar, los sistemas de Filtros Colaborativos se pueden utilizar en dominios donde no existe mucha información asociada con los productos o servicios, o donde el contenido es difícil de analizar usando una computadora . En segundo, los sistemas de Filtros Colaborativos tienen la habilidad de generar recomendaciones inesperadas -es decir, el sistema puede realizar recomendaciones relevantes al usuario, sin que estén de alguna forma relacionadas al perfil del usuario.

1.1. Trabajo relacionado

Los sistemas de recomendación han experimentado un gran auge en tiempos recientes en diferentes industrias y al día de hoy es un tema en el cual se continúa invirtiendo

en investigación y desarrollo de nuevas metodologías debido a los retos que aún existen en el tema [Schedl et al., 2018, Elahi et al., 2016, Rubens et al., 2015]. Algunos de estos retos son inicialización en frío, el problema con conjuntos de datos dispersos, estrategias de evaluación de sistemas de recomendación. El problema de inicialización en frío [Schein et al., 2002] se da cuando existe un nuevo usuario del sistema o se introduce un nuevo producto o servicio al sistema, en este caso no existe suficiente información en el sistema relacionado al usuario o al producto/servicio que permita inducir una recomendación personalizada al usuario.

En el día a día, empresas como LinkedIn, Amazon, Google, Facebook, etc., utilizan sistemas de recomendación para generar valor agregado a los usuarios a partir del análisis de los datos recolectados. LinkedIn es una comunidad orientada a las empresas, a los negocios y al empleo, la empresa ha desarrollado un Sistema de Recomendación de Talento [Geyik et al., 2018] este sistema requiere calcular un interés compartido entre el reclutador, el candidato y los requerimientos de la posición de trabajo y basado en esta información el sistema provee al reclutador con una lista de candidatos potenciales, por otra parte la empresa ha desarrollado un Sistema de Recomendación de Trabajo Personalizado [Kenthapadi et al., 2017] el cual permite realizar recomendaciones de las mejores oportunidades laborales para cada perfil.

Amazon es una empresa multinacional enfocada en comercio electrónico, cómputo en la nube e inteligencia artificial, la empresa ha trabajando con sistemas de recomendación por más de 20 años [Smith and Linden, 2017], usando y perfeccionando su algoritmo de filtros colaborativos que permite hacer conjuntos de productos para hacer recomendaciones, en lugar de hacer conjuntos de personas [Linden et al., 2003b]. Google, otro de los grandes en la industria tecnológica sigue invirtiendo en sistemas de recomendación en diferentes áreas [Covington et al., 2016, Cheng et al., 2016, Chen et al., 2019, Beutel et al., 2018], su área de desarrollo de *Youtube* [Covington et al., 2016] debido a la naturaleza de su información utiliza Redes Neuronales Profundas para su sistema de recomendación. Por otra parte, la empresa ha desarrollado fusiones de redes neuronales como lo muestra [Cheng et al., 2016], o como lo ha publicado la compañía en su área especializada en Inteligencia Artificial *Google Brain*, investigando nuevas técnicas de desarrollo de sistemas de recomendación utilizando Redes Neuronales Recurrentes [Zheng et al., 2018] o Aprendizaje Reforzado [Chen et al., 2019]. Facebook es otra compañía liderando la carrera de la Inteligencia Artificial, también está invirtiendo en nuevas tecnologías en el ámbito de los Sistemas de Recomendación utilizando Redes Neuronales Profundas [Zheng et al., 2018] y como lo ha publicado su departamento especializado en Inteligencia Artificial FAIR (*Facebook Artificial Intelligence Research*) usando nuevas tecnologías como Aprendizaje Reforzado [Warlop et al., 2018].

En un intento por mejorar los Sistemas de Recomendación y que el público en general se beneficie de esta investigación se han generado conjuntos de datos que son abiertos al público en plataformas capaces de hacer la evaluación de los resultados [Kaggle, 2010, CrowdAI, 2018]. Uno de los casos más sonados fue el Premio Netflix [Bennett et al., 2007, Netflix, 2006] donde se ofrecían premios a los equipos que crearan un algoritmo que venciera en exactitud al algoritmo Cinematch por ciertos montos [Netflix, 2006], concluyendo con una mejora de 10.10 % a las predicciones del algoritmo Cinematch. Al momento de escribir este documento existe el RecSys Challenge [RecSys, 2012], un evento anual donde el reto es diseñar un sistema de recomendación para un caso de uso en específico. El cual en 2019 es organizado por Trivago [Trivago, 2005b] y tiene como reto desarrollar un sistema de recomendaciones [RecSys, 2019] basado en sesiones y contexto para proveer una lista de hospedajes que cumplan con las necesidades de los usuarios.

La investigación en sistemas de recomendación no se limita los grandes corporativos, existen investigaciones desarrolladas por universidades alrededor del mundo en este ámbito. Sólo por citar algunos ejemplos, Universidades de Alemania y Japón han implementado sistemas de recomendación de documentos científicos (*papers*), en conjunto las Universidades de Magdeburgo y Berlín en Alemania han implementado un híbrido de filtros colaborativos, métodos estadísticos y Máquinas de soporte vectorial (SVM, del inglés *Support Vector Machines*) [Gipp et al., 2009], mientras que la Universidad de Kyoto en Japón realizó su desarrollo usando HCF-IDF(*Hierarchical Concept Frequency - Inverse. Document Frequency*) sólo en los títulos de los documentos científicos para no infringir los derechos de autor [Nishioka and Ogata, 2018]. Por otra parte, el Instituto de Tecnología Aeronáutica de Nanchang en China ha implementado Sistemas de Recomendación usando Filtros Colaborativos para Comercio Electrónico [Wen and Shui-Sheng, 2006].

En el ámbito local, también se ha desarrollado investigación en el área de sistemas de recomendación. La Universidad de Yucatán, México en conjunto con la Universidad de Castilla la Mancha y la Universidad de Córdoba, España publicaron un sistema híbrido de recomendaciones de Objetos de Aprendizaje [Zapata-Gonzalez et al., 2011] usando la técnica de vecinos más cercanos para encontrar similitudes entre Objetos de Aprendizaje y Usuarios, y reglas de minado para descubrir Objetos de Aprendizaje que fueron descargados en conjunto. De la misma forma, el Instituto de Instalaciones Eléctricas(IIE) en conjunto con el Instituto Nacional de Astrofísica, Óptica y Electrónica(INAOE) publicaron un sistema de recomendación para la operación de plantas generadoras [Reyes et al., 2009], este sistema de recomendación se basa un modelo de refinamiento iterativo para la resolución de procesos de Markov. Y más recientemente,

el Instituto Tecnológico de Orizaba en conjunto con el Instituto Tecnológico de Culiacán presentaron EmoRemSys [Lopez et al., 2016], un Sistema de recomendación de recursos educativos basado en detección de emociones, este sistema se basa en la plataforma Apache Mahout [Lyubimov and Palumbo, 2016] y el API de SkyBiometry [sky, 2019], y utiliza la técnica de vecinos más cercanos en conjunto con el coeficiente de correlación de Pearson para realizar las recomendaciones. De la misma forma en la industria existen esfuerzos en el desarrollo de nuevas técnicas en sistemas de recomendación, la empresa Beeva con presencia en España y México ha desarrollado un sistema de recomendación basado en filtros colaborativos para la empresa BBVA [Bee, 2018], además de otros esfuerzos de desarrollo como lo muestra la presentación de la Investigadora Blanca Vargas Govea colaboradora de la empresa OCC Mundial para la revista Software Gurú [Vargas Govea, 2018].

Para concluir, los sistemas de recomendación no son algo nuevo en la industria, sin embargo, se han diversificado las aplicaciones de estos sistemas y esto repercute en la necesidad de desarrollar nuevas técnicas y tecnologías para su implementación. Es por esto que durante la ultima década se han realizado estudios utilizado fuentes de datos estructuradas [Lee et al., 2009]. Sin embargo, en los años mas recientes las investigaciones se han desviado al uso de conjuntos de datos implícitos, en esta categoría entran las redes neuronales [Zhang et al., 2017] y este es el tema principal para el desarrollo del trabajo presente.

1.2. Planteamiento del problema

Existen diferentes algoritmos de inteligencia artificial para proveer soluciones que permiten realizar recomendaciones personalizadas, cada uno enfocado en un conjunto de datos con características específicas. Para efectos de este trabajo se considera utilizar un conjunto de datos no disperso para implementar una arquitectura que permita generar recomendaciones utilizando un autocodificador con eliminación de ruido (DA, del inglés *Denoising Autoencoder*).

En este trabajo de tesis se plantea el uso de una red neuronal con autocodificadores para la generación de recomendaciones, utilizando como datos de entrenamiento un conjunto de datos no disperso de calificaciones de productos.

1.3. Objetivo General

Implementar una red neuronal para la generación de recomendaciones utilizando un conjunto de datos no dispersos.

1.4. Objetivos Particulares

- Revisar el estado del arte en sistemas de recomendación y redes neuronales.
- Explorar diversos modelos de redes neuronales usados como base para sistemas de recomendaciones.
- Revisar bibliotecas para implementación de redes neuronales.
- Diseñar una arquitectura de red neuronal que permita la generación de recomendaciones a partir de una entrada parcial de datos.
- Implementar la arquitectura propuesta utilizando alguna biblioteca de redes neuronales.
- Analizar y comparar resultados de la arquitectura.

1.5. Metas

A continuación se listan las metas a seguir durante la elaboración del proceso de investigación:

- Reporte del estado del arte de los sistemas de recomendación modernos.
- Revisión de los marcos de trabajo (*Frameworks*) para ejecutar modelos de redes neuronales.
- Selección de un conjunto de datos que esté documentado para realización de pruebas con algoritmos de recomendación basaos.

- Implementación y ajuste de los parámetros de los modelos de redes neuronales sobre el conjunto de datos seleccionado.
- Reporte de ejecución y análisis de los modelos de arquitecturas de Redes Neuronales implementadas.
- Documentación de los resultados de la pruebas.
- Elaboración del documento de tesis.

1.6. Hipótesis

Con el uso de una arquitectura profunda es posible implementar un Sistema de Recomendación basado en Filtros Colaborativos que permita generar recomendaciones de productos o servicios a los usuarios basadas en los patrones implícitos en la información.

1.7. Limitaciones

El alcance de la tesis se limita a la implementación un algoritmo que permita generar recomendaciones a partir del análisis de un conjunto de datos no disperso. Analizar las arquitecturas propuestas en trabajos relacionados, el diseño de una arquitectura que cumpla con los objetivos planteados y la presentación de los resultados.

Para la realización de este trabajo, se usan conjuntos de datos disponibles en Internet que se han utilizado en el pasado para trabajos similares.

1.8. Metodología

La metodología ocupada en el desarrollo de este proyecto contiene los pasos que se muestran en la figura 1.1:

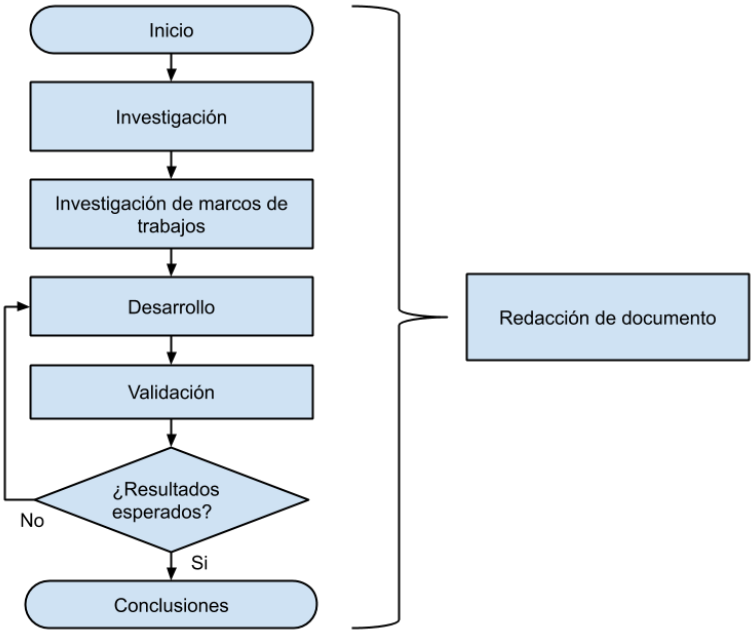

Figura 1.1: Representación de la metodología a usar durante el desarrollo de la tesis.

- **Investigación y especificación de los algoritmos.** En esta fase se revisan y seleccionan los algoritmos a implementar basados en sus características.
- **Revisión de las bibliotecas sobre los marcos de trabajo de desarrollo**. Se revisan las bibliotecas que permiten la ejecución de los algoritmos seleccionados. Se analizan el preprocesamiento necesario a los datos de entrada y los parámetros de ajuste.
- **Codificación y ejecución.** Se realiza la implementación en la biblioteca seleccionada considerando el preprocesamiento y ajuste de los parámetros para el procesamiento de cada conjunto de datos seleccionado.
- **Verificación y validación.** Se comprueba que los resultados de ejecución sean consistentes y se hace una comparación con otras implementaciones sobre los conjuntos de datos seleccionados para obtener una referencia de la validez.
- **Escritura de la tesis.** Se realiza durante todo el desarrollo del trabajo de investigación.

Capítulo 2

Marco Teórico

2.1. Introducción

Los sistemas de recomendación [Adomavicius and Tuzhilin, 2005] son herramientas de software y técnicas que se enfocan en filtrar sobre una cantidad muy grande de productos y/o servicios aplicando alguna función de utilidad para generar una lista de sugerencias reducida y personalizada. Estas sugerencias están relacionadas a diversos procesos de toma de decisiones, tales como qué productos comprar, qué musica escuchar, qué comer, qué lugares visitar, incluso qué noticias leer en línea. Un sistema de recomendación se enfoca en un tipo de producto o servicio, por lo tanto el diseño, la experiencia gráfica del usuario y la técnica utilizada por el generador de recomendaciones deben ser adecuados para proveer recomendaciones efectivas y asegurar una alta probabilidad de uso [Linden et al., 2003a].

Para tener sistemas de recomendación altamente efectivos es necesario la recopilación y el procesamiento continuo de diferentes tipos de información. Estos datos principalmente son acerca del producto o servicio a recomendar y el usuario que recibe las recomendaciones [Adomavicius and Tuzhilin, 2005], aunque esto puede variar en función de la técnica de recomendación a utilizar.

En general, existen técnicas de recomendación que con solo recopilar información limitada (como pueden ser calificaciones y/o evaluaciones) son capaces de generar recomendaciones. Por otra parte, existen sistemas que requieren un conocimiento más

profundo, en este caso es posible usar descripciones ontológicas del usuario y del producto o servicio, información acerca de sus limitantes e incluso las relaciones sociales o actividades del usuario. En cualquiera de los casos, a partir de la información recopilada por el sistema es posible identificar 3 tipos de objetos: Usuarios, Productos o Servicios y Transacciones.

Como se menciona previamente, para poder generar recomendaciones personalizadas es necesario almacenar un rango amplio de información acerca del usuario, de cada uno de los productos o servicios y de las transacciones que ha llevado a cabo el usuario dentro del sistema. La información del producto recopilada puede ser muy diversa, por ejemplo un sistema de recomendación de películas puede almacenar el genero, la duración, el lenguaje, los actores principales, el país de origen, etc. Lo mismo aplica para el usuario, se pueden recopilar datos como género, rango de edad, localización, nacionalidad, etc. Esta información se puede estructurar de diversas formas y la cantidad se dicta por el tipo de técnica a utilizar. Con respecto a las transacciones se pueden listar: el número de visitas, regresos y el tiempo que los usuarios (o algún usuario o grupo de interés) ocupan visualizando un producto, las compras realizadas o los elementos que hay en el carrito de compras, por mencionar algunos.

La aplicación de los sistemas de recomendación es muy amplia, su uso destaca en diversas industrias como son [Ricci et al., 2010] :

- Entretenimiento: recomendaciones de películas, música y sistemas pagados de televisión.
- Contenido: recomendaciones de noticias, documentos, páginas Web, aplicaciones de aprendizaje en línea y filtros de correo electrónico.
- Comercio Electrónico: recomendaciones de productos a comprar por los consumidores.
- Servicios: recomendación de destinos de viaje, consulta de expertos, casas en renta o incluso citas en línea.

2.2. Redes neuronales Artificiales

Existen escenarios complejos en los que no es posible abstraer un algoritmo que generalice una respuesta correcta para un conjunto grande de datos. Para proveer una

solución a estos escenarios se han desarrollado sistemas computacionales inspirados en redes neuronales biológicas. Estos sistemas cuentan con unidades de cómputo no lineales llamadas neuronas, las cuales activan a otras neuronas adyacentes por medio de conexiones modeladas a través de un peso. El valor de los enlaces hacia las neuronas adyacentes y una función de activación son las que determinan si la neurona será activada o continua inactiva.

El proceso de aprendizaje o entrenamiento en las redes neuronales consiste en optimizar el valor de los pesos adjuntos a cada enlace de la red neuronal hasta que generalice de manera acertada para la mayor parte de los casos de entrenamiento que se proveen a la red neuronal durante el entrenamiento y para casos que no se proveen en el conjunto de datos de entrenamiento pero que pueden ser inferidos a partir de estos datos.

Es importante recalcar la importancia de la función de activación de las neuronas, ya que esta función es esencial para proveer una solución a problemas no lineales. Existen diversas funciones de activación, entre ellas Sigmoide, ReLU, Leaky Relu, por citar algunas [Khan et al., 2018].

2.2.1. Clasificación de la Redes Neuronales

La arquitectura en que las neuronas se conectan tiene una relación directa con los algoritmos que pueden ocuparse para su entrenamiento. La clasificación de las redes neuronales basada en su estructura permite identificar tres arquitecturas principalmente [Haykin, 2009]:

Redes Unicapa *feed-forward*

En la forma más simple de redes neuronales se tiene una capa de entrada de datos que envía datos directamente a una capa de neuronas de salida, pero no viceversa, es decir la capa es estrictamente del tipo hacia adelante (*feed forward*), como lo muestra la figura 2.1. Este tipo de redes neuronales se denomina unicapa haciendo referencia solamente a la capa de salida, ya que la capa de entrada no realiza ningún cómputo durante la ejecución.

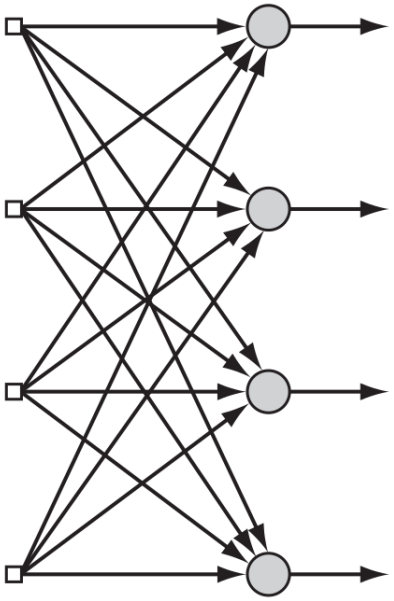

Figura 2.1: Red Neuronal Unicapa: en esta imagen se puede observar la capa de entrada y una capa de procesamiento (neuronas de salida) en la red neuronal (fuente: [Haykin, 2009]).

Redes Multicapa *feed-forward*

Un segundo tipo de Redes Neuronales son las Redes Neuronales Multicapa. Este tipo de Redes Neuronales se caracteriza por tener una o más capas ocultas que realizan algún cómputo que influye en la salida, el término capas ocultas se refiere al hecho de que estas capas no son visibles directamente desde la capa de entrada o la capa de salida (ver figurta 2.2). Las capas ocultas habilitan la red neuronal para ser capaz de abstraer patrones complejos embebidos en los datos de entrada.

En las redes neuronales multicapa cada una de las capas son ejecutadas secuencialmente, esto significa que los datos de la capa de entrada se envían a la capa oculta, la capa oculta realiza el cómputo y la salida es enviada a la siguiente capa oculta, es decir, se realizan los cálculos capa por capa sucesivamente hasta llegar a la capa de salida.

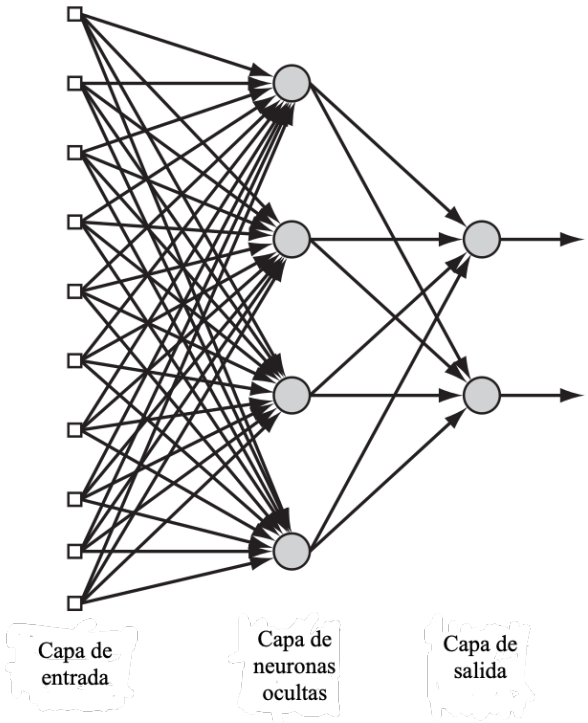

Figura 2.2: Red completamente conectada con una capa oculta. El cómputo se realiza en la capa oculta y en la capa de salida (fuente [Haykin, 2009]).

Redes Neuronales Recurrentes

Las redes neuronales recurrentes se distinguen de las redes feed forward porque al menos una de las salidas de la red es utilizada como entrada en la siguiente iteración, esto se ilustra en la figura 2.3. Otra característica es que este tipo de redes no tienen una capa de entrada y una capa de salida claramente definida, es decir la salida de la capa se utiliza como entrada de la siguiente iteración de forma recurrente hasta que se cumple alguna condición.

Estas redes pueden ser clasificadas en redes neuronales recurrentes directas y redes neuronales recurrentes indirectas.

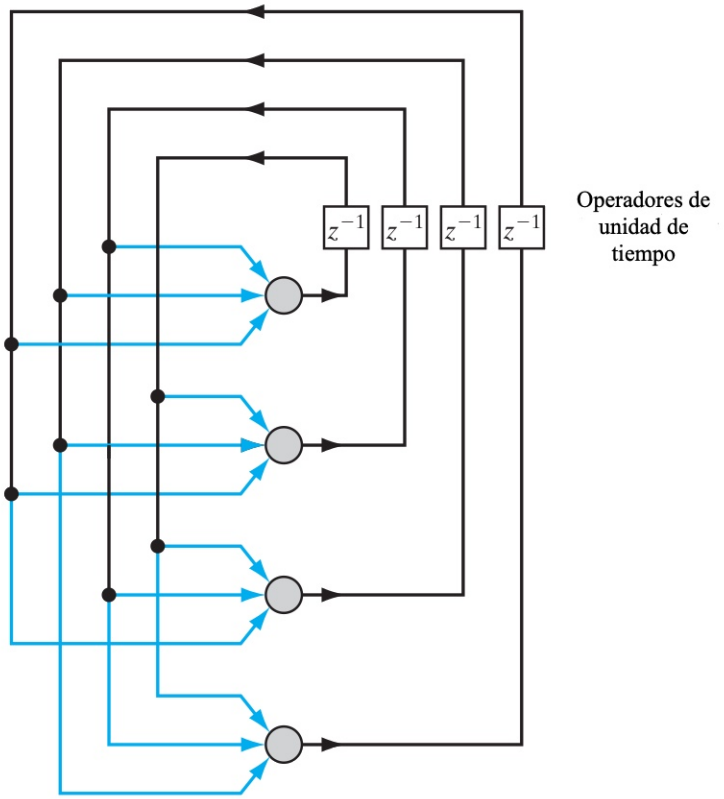

Figura 2.3: Red Neuronal Recurrente: Esta imagen ilustra como las salidas de las neuronas pueden ser usadas como entradas de las misma neurona (fuente: [Haykin, 2009]).

2.2.2. Arquitecturas profundas

Las arquitecturas profundas permiten construir modelos computacionales que son compuestos de múltiples capas de procesamiento para aprender representaciones de los datos con diferentes niveles de abstracción. Estos modelos han mejorado significativamente el estado del arte en reconocimiento del lenguaje, reconocimiento de objetos visuales, detección de objetos y muchos otros dominios tales como descubrimiento de fármacos y genomas.

2.3. Aprendizaje Automático

A lo largo de la historia las computadoras han sido diseñadas para realizar el cómputo de tareas repetitivas que faciliten la vida de los seres humanos. Sin embargo, cuando un sistema es capaz de modificar su estructura, programa o datos con la finalidad de mejorar su rendimiento en los resultados a futuro, en este caso existe una justificación para decir que el sistema está aprendiendo. El tipo de tareas más comunes que relacionan con el aprendizaje automático son: reconocimiento, diagnóstico, planeación, control de robots, predicción, entre otros [Alpaydin, 2010].

Basado en esto, es posible definir el aprendizaje automático como la programación de computadoras para generalizar comportamientos y reconocer patrones, tendencias y relaciones entre datos. Al proponer un modelo con parámetros definidos, el aprendizaje es la ejecución de un programa para optimizar los parámetros del modelo usando datos de ejemplo o experiencias pasadas. El modelo podría ser predictivo para tomar decisiones a futuro, o descriptivo para obtener aprendizaje de los datos, o ambos [Watt et al., 2016].

Ante la posibilidad de diseñar sistemas y algoritmos que realicen las tareas de forma eficiente y con un alto de grado de exactitud, existen justificantes para la investigación y desarrollo de sistemas de aprendizaje automático [Alpaydin, 2010, Watt et al., 2016]:

- La Investigación y desarrollo del aprendizaje automático, es un medio para ayudar a comprender la forma en que los seres humanos y los animales aprenden.
- Existen algunas tareas que no pueden ser definidas claramente más que por ejemplos. En este caso se proveen datos de entrada al sistema y los resultados esperados pero no una relación concisa entre ellos, el sistema debe ser capaz de modificar su estructura a través del procesamiento de una cantidad grande de estos ejemplos y generalizar la relación implícita en los ejemplos.
- El aprendizaje automático puede ser usado para extraer relaciones y correlaciones ocultas entre inmensas cantidades de datos.
- Las condiciones de los ambientes van cambiando. Existe la necesidad de que los sistemas se adapten a estos cambios sin la necesidad de realizar un rediseño.

2.3.1. Medición del error

La capa de medición de pérdida [Zhao et al., 2015] de una red neuronal realiza la comparación de los valores de salida después de realizar el cómputo y los valores reales esperados durante el entrenamiento. Por lo tanto, esta capa es considerada el factor principal del aprendizaje de la red neuronal. El cálculo de la medición del error que regularmente se emplea es la Desviación Cuadrática Media (DCM) que se define por la ecuación 2.1:

$$DCM = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2 \tag{2.1}$$

Esto es entendible debido al numero de propiedades deseables que esta ecuación posee, entre estas propiedades destacan que es una ecuación convexa y diferenciable.

2.3.2. Optimización

Los algoritmos de arquitectura profunda requieren realizar optimización en muchos contextos[Goodfellow et al., 2016], de todos estos problemas en donde la optimización es requerida el mas difícil es el entrenamiento de las redes neuronales. Aun en la actualidad es común invertir días o incluso meses y cientos de computadoras para lograr el entrenamiento de una red neuronal que resuelve un problema especifico. Se han desarrollado algoritmos básicos para el resolver este problema por ejemplo Descenso de Gradiente Estocástico[Robbins and Monro, 1951], Momentum [Polyak, 1964], etc. pero a lo largo de los años se han desarrollado nuevos métodos entre ellos RMSProp [Tieleman and Hinton, 2012] que es el algoritmo utilizado para este trabajo.

Un problema común durante el entrenamiento con los métodos de Descenso de Gradiente Estocástico es que los gradientes tienden a desaparecer. Root Mean Square Propogation(RMSProp) es una propuesta para resolver este problema, usa el promedio de los gradientes al cuadrado para normalizar al propio gradiente. Esto tiene un efecto de penalización en función del tamaño de los pasos, decrementa el gradiente para pasos largos e incrementa el gradiente para pasos cortos para evitar que desaparezca.

2.3.3. Autocodificadores

Se denomina auto-codificador (del inglés, Autoencoder) a una red neuronal que ha sido entrenada para recibir una entrada, generar una representación de esa entrada y generar una salida similar al vector de entrada [Ramos-Pérez, 2016].

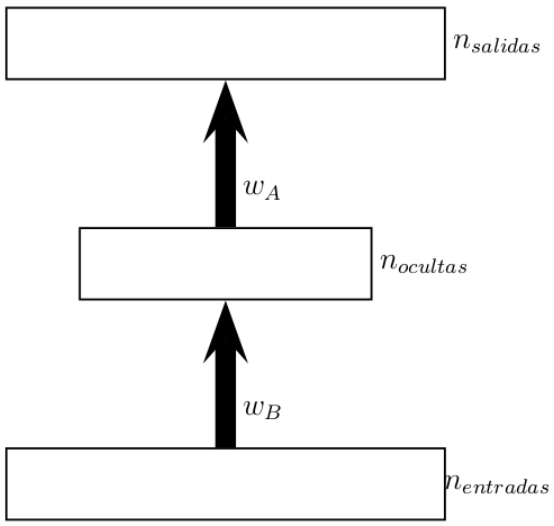

Figura 2.4: Arquitectura de un auto-codificador (fuente [Ramos-Pérez, 2016]).

Esta red neuronal puede tener h capas ocultas y está conformada por dos componentes: un codificador $r = f(x)$ que genera una representación interna de los datos de entrada y un decodificador $x = g(r)$, que a partir de la representación generada por el codificador es capaz de regenerar el vector de entrada 2.4.

Cabe mencionar que una vez que se realiza el entrenamiento del auto-codificador, el codificador es capaz de generar una representación de la entrada en el espacio latente z, con una dimensión reducida en comparación con la capa de entrada, misma que es posible revertir al aplicar el decodificador.

Sin embargo, entrenar una función $x = g(f(x))$ para toda x no tiene mucho sentido, es por eso que existen diversas técnicas para restringir a los auto-codificadores para que copien sólo los rasgos que son mas relevantes para los datos de entrada.

2.3.4. Clasificación de aprendizaje automático

El aprendizaje automático se puede clasificar basado en el tipo de entrenamiento que se da al algoritmo y se divide en tres rubros: aprendizaje automático supervisado, aprendizaje automático no supervisado y aprendizaje automático semi supervisado, a continuación se hace una descripción de cada clase:

- **Aprendizaje Automático Supervisado**: es una de las técnicas más ampliamente estudiadas, debido a que se han desarrollado diversos métodos para su implementación. Ésta consiste en tomar un conjunto de datos de entrenamiento etiquetados y el algoritmo es capaz de determinar una función de mapeo entre las diferentes permutaciones de las entradas y una posible salida. Esto permite al algoritmo generar resultados acertados incluso en escenarios nunca antes vistos, es decir escenarios que no han sido incluidos en los datos etiquetados de entrenamiento. Algunos de los usos de este tipo de aprendizaje es clasificación de correo electrónico (*Spam/No-spam*), predicción del valor de inmuebles, detección de transacciones fraudulentas, etc.

- **Aprendizaje Automático No Supervisado**: al igual que en el Aprendizaje Automático Supervisado recibe un conjunto de datos de entrenamiento, pero no se proveen etiquetas que retro alimentan el algoritmo al momento de su entrenamiento. Sin embargo, después de procesar los datos es posible encontrar similitudes o patrones que permiten hacer representaciones del estado latente de los datos de entrada, lo cual hace posible la detección de anomalías en los datos, la predicción de estados futuros, patrones de agrupamiento, entre otras cosas. Entre los usos más comunes está la clasificación de noticias por tema, agrupamiento de moléculas, clasificación de comunidades en Internet, etc.

- **Aprendizaje Automático Semi-supervisado** hace referencia un punto intermedio entre Aprendizaje Automático Supervisado y No Supervisado, donde los datos de entrenamiento pueden incluir etiquetas que proveen retroalimentación al algoritmo pero al mismo tiempo existen datos donde las etiquetas no están presentes. En este caso el algoritmo comienza con el procesamiento de los datos etiquetados, y ocupa las propiedades de los datos etiquetados y no etiquetados para realizar una clasificación en un segmento los datos no etiquetados [Chapelle et al., 2006]. Entre sus usos están problemas relacionados al procesamiento de lenguaje natural y la clasificación de textos.

Por otra parte, es posible clasificar el aprendizaje automático basado en la estructura de las fuentes de entrada del algoritmo, en este caso se puede hacer una clasificación [Dietterich et al., 2008] entre aprendizaje automático estructurado y aprendizaje automático no estructurado:

- **Aprendizaje Automático Estructurado**: se refiere al aprendizaje de hipótesis generadas a partir de datos con una estructura interna que contiene una o más relaciones. En general, los datos pueden contener entradas y salidas estructuradas de las cuales algunos segmentos pueden ser inciertos, con ruido o totalmente ausentes. Aplicaciones de este tipo de métodos incluye segmentación y traducción de enunciados, predicción de propiedades farmacológicas de moléculas e interpretación de escenas visuales. Con la implementación del aprendizaje en el contexto de datos altamente relacionados que permitan inferencia sofisticada, el aprendizaje automático estructurado tiene un mayor potencial de proveer las herramientas para construir sistemas integrados de Inteligencia Artificial.
- **Aprendizaje Automático No Estructurado**: se denomina de esta forma cuando el conjunto de datos tiene una estructura interna, pero la estructura no está definida por un modelo de datos o un esquema. En este caso los datos pueden ser generados por humanos por ejemplo conversaciones generadas en mensajeros electrónicos, correos electrónicos, fotos, archivos de audio o de vídeo, sitios web, textos, etc., o pueden ser generadas por computadoras por ejemplo imágenes satélitales, sensores de clima, monitoreo de tráfico, sistemas de vigilancia, etc. Implementando este tipo de técnicas los sistemas adquieren la capacidad de encontrar patrones en cantidades inmensas de datos.

2.4. Sistemas de recomendación

Un sistema de recomendación es una implementación de software totalmente funcional que aplica por lo menos un algoritmo para realizar recomendaciones [Beel et al., 2015]. Además de incluir una interfaz de usuario, un acervo de recomendaciones posibles y un operador que administra el sistema.

Éste se desenvuelve en un escenario de recomendación que describe por completo el contexto incluyendo el sistema de recomendación y el ambiente de recomendación,

por ejemplo, los productos o servicios a recomendar y las características de los usuarios. Y su objetivo es lograr una alta efectividad en las recomendaciones, es decir, proveer recomendaciones "buenasz útiles, que generen usuarios felices porque se satisfacen sus necesidades.

2.4.1. Funciones de los Sistemas de Recomendación

Los Sistemas de Recomendación se pueden utilizar para diferentes fines, algunos ejemplos son la plataforma de vídeos Youtube usa sistemas de recomendación para sugerir vídeos e incrementar el tráfico en la plataforma. Otro caso es la plataforma de ventas en línea Amazon, que hace sugerencias de productos con la finalidad de incrementar sus ventas. A continuación se presentan algunos ejemplos de uso de esta tecnología [Ricci et al., 2010]:

- **Incrementar el número de productos vendidos:** esta es probablemente la función comercial más importante de un sistema de recomendación. Su objetivo se logra cuando las recomendaciones del sistema son productos o servicios que tienden a cubrir las necesidades actuales del usuario. En general, el objetivo del sistema es incrementar la probabilidad de que una recomendación se convierta en una venta de un producto o servicio.
- **Vender productos más diversos:** otra de las principales funciones del sistema de recomendación es ayudar al usuario a encontrar productos o servicios que sería complicado encontrar sin una recomendación específica. En otras palabras, el sistema recomienda no solo los productos o servicios más populares, sino los que más se adecuan a sus necesidades.
- **Incrementar la satisfacción del usuario:** el usuario encontrará las recomendaciones interesantes, relevantes y complementado con un diseño apropiado de interfaz humano-computadora disfrutará más el uso de la plataforma. La combinación de recomendaciones efectivas y una interfaz adecuada puede incrementar la evaluación subjetiva del sistema por parte del usuario.
- **Incrementar la fidelidad del usuario:** un usuario puede ser fiel a un sitio Web el cual, cuando es visitado reconoce un cliente frecuente y lo trata como tal. Esta es una de las funciones de un sistema de recomendación, ya que estos normalmente realizan el cómputo de recomendaciones utilizando la información adquirida en interacciones previas del usuario.

- **Entender mejor qué es lo que el usuario quiere:** otra de la funciones importantes de los sistemas de recomendación, la cual puede ser compartida con otras aplicaciones es las descripción de las preferencias del usuario. Éstas pueden ser recolectadas explícitamente o implícitamente: se dice que los datos son recolectados explícitamente cuando el sistema provee una interfaz exclusivamente para este propósito, por ejemplo: encuestas, formularios, etc., por otra parte, la recolección de datos implícita se realiza durante la navegación habitual del usuario, por ejemplo: se recolecta información de los clics ejecutados por el usuario, los productos visitados, las calificación que el usuario da a un producto o servicio, comentarios del usuario, etc.

2.4.2. Clasificación estrategias para sistemas de recomendación

Existen diferentes clasificaciones de los sistemas de recomendación, a continuación se describe las principales [Beel et al., 2015]:

- **Sistemas de recomendación por estereotipos:** este es uno de los primeros modelos de sistemas de recomendación, es inspirado en la psicología que permite a los psicólogos rápidamente juzgar a los personas basándose solo en algunas características.
- **Sistemas de recomendación basados en filtros de contenido:** es unos de los sistemas de recomendación mas estudiados e investigados. Uno de sus componentes principales es el proceso de modelo del usuario, en el cual el interés del usuario es inferido a partir de los atributos de los productos o servicios con los cuales el usuario tuvo interacción.
- **Sistemas de recomendación basado en filtros colaborativos:** el principio detrás de este método asume que a los usuarios les gusta lo que a personas con gustos similares les gusta (figura 2.5), para esto se considera que dos personas tienen gustos similares cuando califican de manera similar el mismo producto o servicio.
- **Sistemas de recomendación por co-ocurrencia:** este se basa en las relaciones que existen entre los productos o servicios al ser consumidos por los usuarios. El caso de uso más popular de este modelo es la implementación de Amazon [Schafer et al., 1999, Linden et al., 2003b] “Comprados juntos habitualmente”.

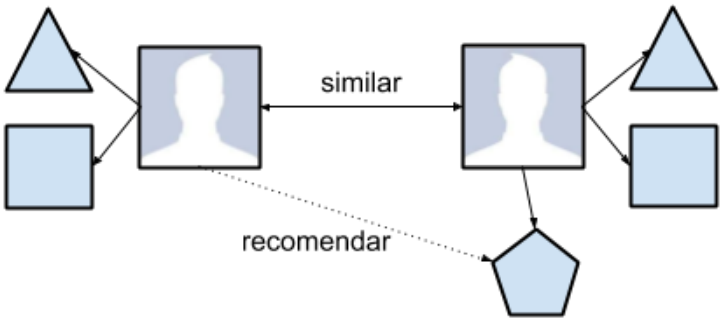

Figura 2.5: Principio de filtros colaborativos.

- **Sistemas de recomendación por relevancia global:** en este caso el modelo adopta un modelo único no personalizado y realiza recomendaciones de productos o servicios basado en la relevancia. Algunos casos de uso de este modelo son en Youtube "Subidos recientemente" y Amazon "Los mas vendidos".
- **Sistemas de recomendación híbridos:** el modelo de sistemas de recomendación híbrido se refiere a sistemas de recomendación que integra 2 ó más estrategias.

2.4.3. El problema cold-start

Se investigan dos vertientes del problema cold-start [Gonard, 2018], en primer lugar tenemos problema relacionado al producto o servicio: éste se presenta cuando no existen calificaciones para el producto o servicio, un ejemplo de esto es cuando se agrega un nuevo producto o servicio al catálogo. Por otra parte, se tiene el problema relacionado al usuario: en este caso el usuario acaba de registrarse en el sistema y no ha proporcionado ninguna calificación a productos o servicios, por lo tanto no es posible para el algoritmo establecer una relación con productos o servicios que probablemente le gusten.

Ambas vertientes tienen sus complejidades, pero la vertiente relacionada al usuario representa un reto mayor debido a que el comportamiento es distinto de una persona a otra.

Capítulo 3

Desarrollo del proyecto

En esta sección se presentan las herramientas, biblioteca y lenguaje de programación utilizado durante el desarrollo del presente trabajo. Por otra parte, se describen las características de la solución propuesta, y del conjunto de datos utilizado.

3.1. Biblioteca de Aprendizaje Profundo

Existe un amplio número de bibliotecas y herramientas para la implementación de algoritmos de aprendizaje profundo [Erickson et al., 2017], cada una de ellas se respalda en una comunidad de usuarios y desarrolladores diferente, y ofrece sus propias ventajas y desventajas.

En el cuadro 3.1, [Synced, 2019] se muestra un comparativo del rendimiento y la utilización de recursos de las bibliotecas TensorFlow 1.13, PyTorch 1.1.0 y MXNet 1.4.0. Como se puede observar MXNet realiza un mejor manejo de los recursos del sistema y tiene un rendimiento similar a TensorFlow y PyTorch, debido a esto y a otras características que se mencionan en la siguiente sección se eligió a MXNet como la biblioteca para el desarrollo del proyecto.

Batch size 256 train	GPU			CPU	Memory		Performance
	GPU utilization	Memory utilization Time	Memory Used/Total (MiB)	CPU Utilization	Memory Utilization	Memory Used/Total	Speed (steps /sec)
TF 1.13	21.62%	4.79%	23077/24190	4.36%	21.57%	3387/15703	3.09
PyTorch 1.1.0	11.76%	1.88%	1472/24190	4.79%	29.00%	4554/15703	3.13
MXNet 1.4.0	20.22%	1.40%	1311/24190	3.63%	26.27%	4126/15703	3.06

Cuadro 3.1: Rendimiento y utilización de recursos durante el entrenamiento de Redes Neuronales de Filtros Colaborativos (Fuente: [Synced, 2019]).

3.1.1. MXNet

MXNet [Chen et al., 2015] es una biblioteca de aprendizaje automático multi-lenguaje que facilita el desarrollo de algoritmos, en particular de aprendizaje profundo. MXNet ofrece interfaces de alto nivel para diversos lenguajes, una simbólica y otra imperativa. De la misma forma, realiza cómputo y un manejo de memoria eficiente, y se puede ejecutar en varios sistemas heterogéneos, desde dispositivos móviles hasta arreglos de GPU's distribuidos.

Además del rendimiento y manejo de los recursos del sistema que se muestran en el cuadro 3.1, a continuación se listan otras características que favorecen a MXNet como la biblioteca adecuada para el presente proyecto:

- Es ligero y eficiente.
- Ofrece una interfaz declarativa simbólica de alto nivel, pero también cuenta con otra imperativa a nivel de tensores.
- Ofrece una interfaz políglota.
- Existe una comunidad de usuarios extensa y activa que dan soporte.

MXNet ofrece APIs en diferentes lenguajes para la implementación de algoritmos de aprendizaje automático. Esta lista de lenguajes incluye Python, Julia, R, C++, Clojure, Java, Perl y Scala. Evaluar cada uno de estos lenguajes no está dentro de los límites de este trabajo, simplemente se exponen algunas de las razones por las cuales fue seleccionado Scala como lenguaje de programación de desarrollo.

Scala combina programación orientada a objetos y programación funcional en un lenguaje de alto nivel de forma concisa. Scala tiene tipos estáticos lo cual evita errores en aplicaciones complejas, y su máquina virtual de Java permite construir sistemas de alto rendimiento con fácil acceso a un enorme ecosistema de bibliotecas.

Según Odersky [Odersky et al., 2004], algunos de los aspectos más sobresalientes del lenguaje son:

- Scala tiene un modelo de objetos uniforme, en este sentido cada valor es un objeto y cada llamada a operación es un método.
- Scala es también un lenguaje funcional, lo que implica que las funciones son valores de primera-clase.
- Permite la descomposición de objetos mediante la evaluación de patrones.
- Scala cuenta con abstracciones de conceptos uniformes y eficientes para tipos y valores.
- Tiene una composición de *mixins* (combinación de clases que proveen funcionalidad pero no pueden ser instanciadas) simétrica y flexible para la composición de clases y *traits* (similares a las interfaces pero permiten una implementación parcial).

3.2. Hardware

El desarrollo del presente trabajo se realizó en una computadora Lenovo Legion con las siguientes características:

- Lenovo Legion Y730.
- Ubuntu 16.04.
- Procesador 2.9 GHz Intel Core i7.
- Memoria 16 GB 2133 MHz LPDDR3.
- 1 Terabyte de disco duro + Intel Optane 16gb.
- GPU NVIDIA GTX 1080 TI.

3.3. Conjunto de datos

El conjunto de datos usado para este trabajo es Jester de Movie Lens [Goldberg et al., 2001]. Este conjunto de datos fue concebido y está disponible libremente para la investigación de filtros colaborativos. Contiene 4.1 millones de calificaciones de 100 chistes de 73,421 usuarios recabados entre Abril de 1999 y Mayo del 2003.

El formato del archivo de entrada Figura 3.1 es el siguiente:

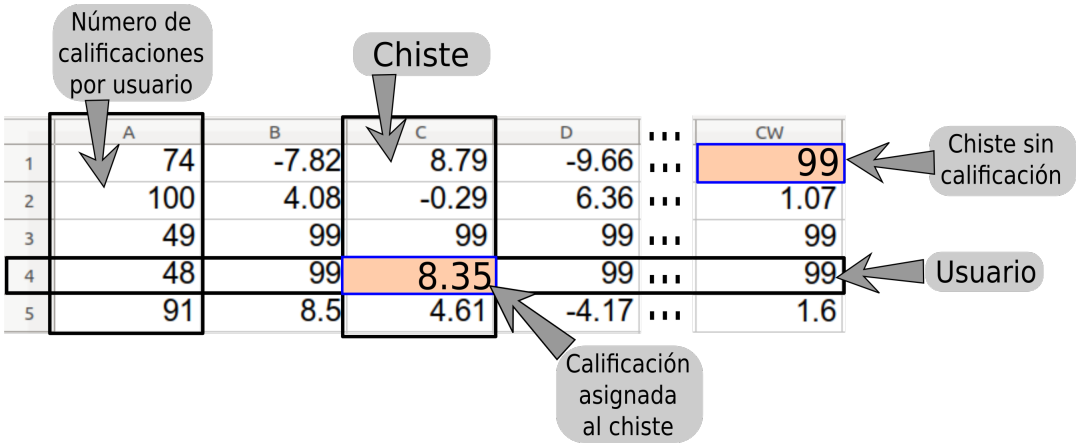

Figura 3.1: Formato del conjunto de datos Jester.

- 3 Archivos que contienen datos de calificaciones anónimas de un total de 73,421 usuarios.
- Los datos se encuentran en un archivo formato .zip que contiene archivos en formato excel (.xls).
- Las calificaciones son valores reales en el rango de -10.00 a 10.00 (el valor 99 hace referencia un valor "sin calificación").
- Existe una fila por usuario.
- La primera columna representa el número de calificaciones del usuario.
- La sub-matriz incluyendo las columnas 5, 7, 8, 13, 15, 16, 17, 18, 19, 20 es densa.

3.4. Pre-procesamiento de los datos

Desafortunadamente los datos no siempre se presentan en un formato que puede ser utilizado por los algoritmos desarrollados, esta necesidad de adaptar los datos a un formato específico es el trabajo que se describe en esta sección.

3.4.1. Pre-procesamiento

El conjunto de datos de Jester de Movielens está contenido en un archivo Excel (.xls) que no contiene encabezado, la primera columna contiene el numero de calificaciones que el usuario ha capturado y las calificaciones se muestran en un rango de -10 a 10 (ver figura 3.1).

Para poder procesar el conjunto de datos por la red neuronal de arquitectura profunda es necesario convertir los valores a una representación matricial donde se elimine la primera columna que contiene el número de calificaciones capturadas por el usuario y se normalicen los valores de las calificaciones en un rango de 0 a 1. Una vez completado este proceso, cada una de las columnas de la matriz representa un chiste y cada unas de las filas a un usuario. La intersección de una fila y una columna representa la calificación proporcionada por el usuario.

El pre-procesamiento de los datos(figura 3.2) realizado se describe a continuación:

- Crear una matriz para el procesamiento de los datos.
- Asignar una columna a cada chiste en el conjunto de datos.
- Agregar una fila por cada usuario y realizar un mapeo de las calificaciones a la celda a la que pertenece.
- Aplicar la formula C = (C + 10) / 20 a cada celda de la matriz para su normalización.
- Realizar un re-ordenamiento aleatorio de los datos de entrada.
- Segmentar el conjunto de datos para el entrenamiento(70 %), la validación(15 %) y las pruebas(15 %). Esto es necesario para poder validar y probar la capacidad de generalización de la red con datos no usados durante el entrenamiento.

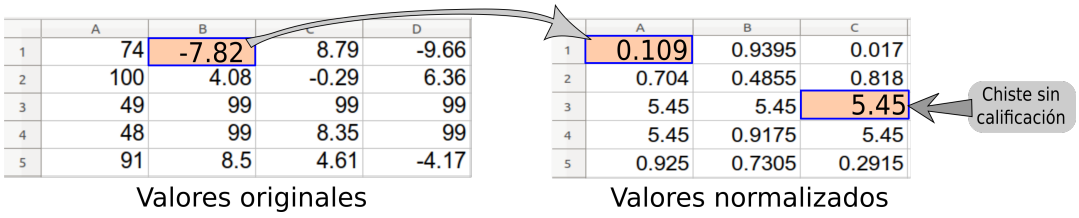

Figura 3.2: Datos de entrada pre-procesados.

3.5. Arquitectura de la red neuronal

En esta sección se describe la arquitectura de la red neuronal desarrollada. En la figura 3.3 se hace una representación de la Red Neuronal en la forma de un grafo acíclico dirigido en el cual cada nodo representado es una capa.

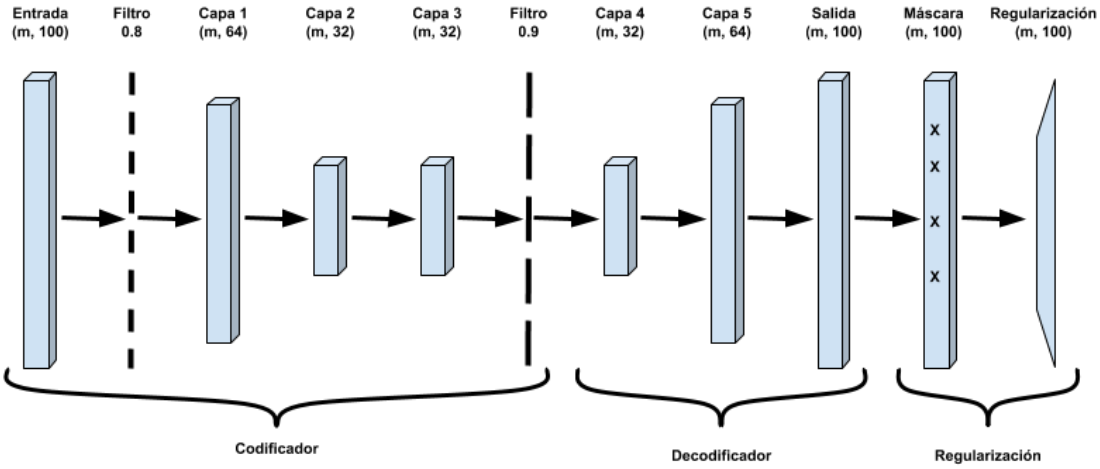

Figura 3.3: Arquitectura profunda implementada para generar recomendaciones.

- La primera capa es llamada la capa de entrada, esta recibe como entrada los datos previamente pre-procesados (figura 3.2) y su dimensión es 100, que es el número de chistes que existen el conjunto de datos.
- La segunda capa es un filtro, el propósito de esta capa es ignorar neuronas durante la fase de entrenamiento de forma aleatoria (*dropout*). Esta es una técnica que

permite a los algoritmos generalizar de una forma más efectiva cuando se aplica una entrada no incluida en el conjunto de datos de entrenamiento.

- La tercera capa genera una representación de la entrada con una dimensión de 64, este es el primer paso en la generación de una representación del auto-codificador.
- La cuarta y la quinta capa generan una representación con una dimensión de 32. Como se puede observar, se genera una representación de la mitad del tamaño de la salida de la tercera capa. Al generar una representación de menor dimensión se asignan valores de acuerdo a patrones identificados en los datos de la capa de entrada.
- La sexta capa es un filtro aplicado a la representación de 32 celdas, en este caso se ocultan neuronas aleatoriamente para que cuando el decodificador realice la expansión a la dimensión de entrada, el algoritmo generalice patrones comunes en la salida que serán utilizados como recomendaciones.
- La séptima capa es la primera parte del decodificador, una vez que el codificador genera una representación de los datos de entrada en una dimensión menor, el decodificador utiliza esta representación como entrada y genera una salida de la misma dimensión que la capa de entrada. En este caso la primera capa del decodificador genera una salida de dimensión 32.
- La octava capa realiza una expansión de la salida de la capa siete de 32 a 64. El decodificador replica los capas del codificador pero en sentido inverso, debido a esto se genera una salida de dimensión 64 como entrada de la ultima capa del decodificador.
- La novena capa de la red expande la salida de la octava capa a la dimensión de la capa de entrada. En este punto el auto-codificador ha generado una representación de la capa de entrada basada en la abstracción de las características de los datos de entrenamiento, incluyendo otros rasgos comunes usados como recomendaciones.
- La décima capa hace un enmascaramiento de los datos, de esta forma se remueven las calificaciones realizadas por el usuario obteniendo patrones de los datos aún no vistos, estos datos serán utilizados como recomendaciones.
- En la última capa se realiza una selección de los valores más sobresalientes, estos serán enviados como recomendaciones al usuario final.

En resumen, la arquitectura presentada en este trabajo es un auto-codificador. Las primeras cinco capas representan el codificador, éste genera una representación de la capa de entrada en un dimensión menor, además de ignorar neuronas aleatoriamente de la entrada, de esta manera el algoritmo abstrae características comunes y con alta relevancia en el conjunto de datos de entrenamiento.

Posteriormente, a partir de la capa seis a la nueve se tiene la representación del decodificador. En la primera capa del decodificador se ignoran algunas celdas de la representación reducida, con esto se logra que el algoritmo generalice la salida para escenarios no mostrados durante el entrenamiento. Al terminar este proceso se obtiene una salida similar a los datos de entrada, que incluye rasgos generalizados encontrados por el algoritmo.

Las ultimas dos capas son las capas de regularización, en estas capas se remueven los datos que ya son conocidos por el usuario y se maximizan los rasgos comunes generados por el algoritmo los cuales son usados como recomendaciones para el usuario final.

Capítulo 4

Resultados y conclusiones

En este capítulo se presentan los resultados y las conclusiones de las pruebas realizadas durante el desarrollo de la tesis, se exponen las diferentes arquitecturas y configuraciones de parámetros e hiper-parámetros así como los resultados de las pruebas.

4.1. Parámetros e hiper-parámetros

Para la selección del número de capas ocultas a incluir en la red neuronal y sus dimensiones, se realizaron una serie de pruebas incluyendo diversas configuraciones.

En la figura 4.1 se ilustra el procedimiento utilizado. Básicamente se realizaron dos fases durante el entrenamiento. La fase 1 fue puramente experimental, se realizó un entrenamiento con diferentes configuraciones variando el número de capas y de neuronas por cada capa, así mismo, se analizaron los resultados. Los modelos con mejores resultados de la fase 1 se usaron para un entrenamiento con diferentes parámetros en la fase 2. Por ultimo se validaron y se documentaron los resultados. A continuación se describe el proceso y las configuraciones utilizadas.

Los hiper-parámetros que se utilizaron durante la fase 1 del desarrollo fueron los siguientes:

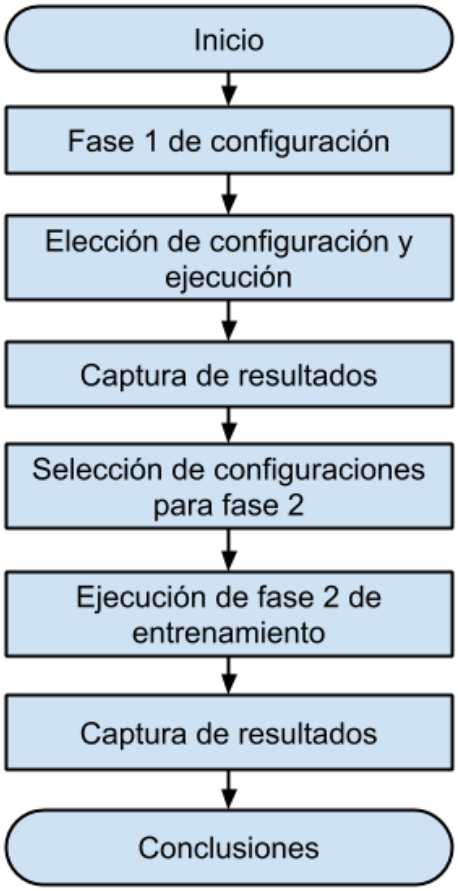

Figura 4.1: Fases ejecutadas para el entrenamiento de la red neuronal.

- Inicialización de los parámetros: utilizando una función uniforme.
- Constante de aprendizaje: 0.001.
- Algoritmo de optimización: RMSProp.
- Número de iteraciones: 500.
- Tamaño de la muestra: 128.

En la tabla 4.1 se describen las configuraciones con las que se experimentó, se realizaron variaciones tanto al número de capas usadas como al numero de neuronas por

Identificador	**Codificador**	**Decodificador**
1	100, 8, 16	16, 8, 100
2	100, 32, 64	64, 32, 100
3	**100, 64, 16**	**16, 64, 100**
4	100, 64, 32	32, 64, 100
5	100,64,16,16	16,64,100
6	**100, 64, 32, 32**	**32, 64, 100**
7	100, 64, 32, 32, 16	16, 32, 64, 100

Cuadro 4.1: Configuraciones de red neuronal usadas en la fase 1 de entrenamiento.

Identificador	**Reducción de error (%)**
1	4.303560111
2	3.191240186
3	**22.15933228**
4	-0.375970316
5	2.61896765
6	**17.48418626**
7	5.957444904

Cuadro 4.2: Porcentajes de reducción del error para las configuraciones en la fase 1.

capa:

Durante el entrenamiento se realizó la captura de los resultados y el cálculo del porcentaje de reducción del error que se obtuvo para cada uno de los modelos. En la tabla 4.2 se muestran los resultados del cálculo de reducción del error:

Una vez que se realizaron la serie de experimentos que se muestran en los cuadros 4.1 y 4.2, se seleccionaron las 2 configuraciones con mejores resultados para una segunda fase de entrenamiento.

En la fase 2 de entrenamiento se realizó una afinación a los parámetros y los hiper-parámetros del modelo. Al incrementar el tamaño de la muestra el error se incrementó, esto permitió entrenar durante una cantidad mayor de iteraciones antes de que llegar a un punto estable en los resultados.

La lista de los parámetros para la fase 2 de entrenamiento son:

- Inicialización de los parámetros: utilizando una función uniforme.
- Constante de aprendizaje: 0.0001.
- Algoritmo de optimización: RMSProp.
- Numero de iteraciones: 5000.
- Tamaño de la muestra: 1024.

En la figura 4.2 se puede observar la evolución del entrenamiento de las 2 configuraciones que se seleccionaron en la fase 1. Aún cuando en la primera fase del entrenamiento la configuración de la red con el identificador #3 obtuvo una reducción del error mayor (22.15 %) comparado con la configuración con el identificador #6(17.48 %) que es una configuración con más capas y un mayor número de neuronas. Se puede observar que a largo plazo la configuración #6 tiene una mayor capacidad de aprendizaje y por lo tanto tiene un mejor rendimiento.

Concluyendo, la configuración de la red neuronal con un codificador con dimensiones **100, 64, 32, 32** y un decodificador con dimensiones **32, 64, 100** fue la que se implementó debido a su mayor exactitud.

4.1.1. Tiempo de entrenamiento y predicción

Realizar el entrenamiento de un modelo es una tarea que requiere invertir tiempo considerable, cada una de las pruebas a las diferentes arquitecturas propuestas necesita tiempo para mostrar resultados.

Una vez que el modelo fue entrenado los tiempos de inferencia para la generación de predicciones es sumamente corto. En el cuadro 4.3 se muestran los tiempos de entrenamiento e inferencia del presente trabajo:

La mejora en los tiempos de ejecución al utilizar el GPU (Unidad de Procesamiento Gráfico) se debe a la implementación de CUDA (Arquitectura Unificada de Dispositivos de Cómputo) y su capacidad de procesamiento en paralelo para la solución de operaciones complejas.

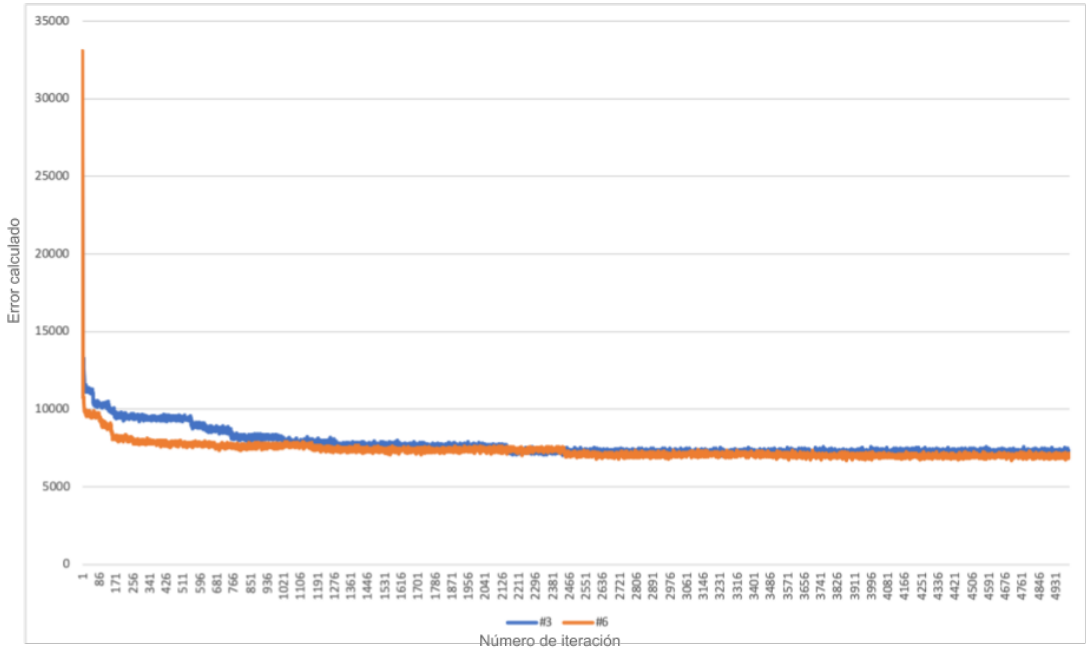

Figura 4.2: Gráfica del error en cada iteración.

4.2. Generación de recomendaciones

Para ilustrar la forma en que el algoritmo genera recomendaciones se presenta en el ejemplo del vector cero. Para esta prueba se generó un vector de dimensión 100 donde cada celda representa una calificación a un chiste. Este es el escenario donde el usuario no ha realizado calificación alguna (*Cold start problem*, subsección 2.4.3). A continuación se muestra la entrada a la red neuronal.

	CPU	**GPU**
Entrenamiento(5000 Iteraciones)	1 hora 42 minutos	17 minutos
Predicción	0.019	0.0034

Cuadro 4.3: Tiempos de entrenamiento y predicción.

$$\begin{vmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{vmatrix}$$

La red neuronal ejecuta las predicciones relacionadas al vector de entrada y genera una salida de la misma dimensión (100), debido a que en la entrada no se introdujo ninguna calificación, los valores generados son la probabilidad de que al usuario le guste un chiste, como ejemplo se puede observar que la probabilidad que al usuario le guste el chiste en la posición 1 es 38.69 % y la probabilidad de que le guste el chiste 4 es 0.00 %. El vector completo de las predicciones generadas es:

$$\begin{vmatrix} 0.3869 & 0.3980 & 0.3605 & 0.0000 & 0.4892 & 0.5050 & 0.4732 & 0.4599 & 0.3025 & 0.5055 \\ 0.5361 & 0.5436 & 0.4071 & 0.5291 & 0.3879 & 0.3479 & 0.4317 & 0.4519 & 0.5041 & 0.4244 \\ 0.5921 & 0.5022 & 0.4501 & 0.2678 & 0.4677 & 0.5276 & 0.6372 & 0.5358 & 0.6226 & 0.3691 \\ 0.5807 & 0.6423 & 0.3009 & 0.4976 & 0.6350 & 0.6378 & 0.3002 & 0.5376 & 0.5237 & 0.5146 \\ 0.3852 & 0.5721 & 0.3502 & 0.2671 & 0.5034 & 0.5376 & 0.5257 & 0.5677 & 0.6160 & 0.6648 \\ 0.3665 & 0.4197 & 0.6188 & 0.6068 & 0.4501 & 0.5712 & 0.2598 & 0.1888 & 0.3733 & 0.3711 \\ 0.5835 & 0.6214 & 0.4510 & 0.3462 & 0.5856 & 0.6018 & 0.3402 & 0.6107 & 0.6017 & 0.4659 \\ 0.0613 & 0.0966 & 0.0000 & 0.0646 & 0.0000 & 0.0977 & 0.0000 & 0.0878 & 0.0716 & 0.0821 \\ 0.0935 & 0.0846 & 0.0930 & 0.0000 & 0.0864 & 0.0885 & 0.0979 & 0.1006 & 0.1097 & 0.0859 \\ 0.1063 & 0.0987 & 0.1118 & 0.1057 & 0.1095 & 0.1196 & 0.1250 & 0.1232 & 0.0000 & 0.1108 \end{vmatrix}$$

Del vector de predicciones el sistema extrae las 10 mejores recomendaciones que el sistema puede generar. Para esto se toma el vector de salida, se hace una clasificación de los valores con una predicción más alta y se filtran solo las mejores.

En el cuadro 4.4 se muestran las recomendaciones generadas por el algoritmo al procesar el Vector 0.

Posición	Numero	Predicción
1	50	0.6648
2	32	0.6423
3	36	0.6378
4	27	0.6372
5	35	0.6349
6	29	0.6226
7	62	0.6213
8	53	0.6188
9	49	0.6160
10	68	0.6106

Cuadro 4.4: Las 10 mejores predicciones para el Vector 0.

Estas recomendaciones no son aleatorias, son rasgos comunes que el sistema aprendió a identificar durante el entrenamiento, en la Figura 4.3 se puede observar que los chistes generados por el sistema se encuentran entre los que más calificaciones tienen. Por ejemplo el chiste numero 50 que aparece como el más recomendado por el sistema tiene 23,499 recomendaciones en el conjunto de datos de entrenamiento, y segundo chiste más recomendado tiene 23,481.

El segundo parámetro que se consideró durante el análisis fue la moda en las calificaciones de los chistes como se puede observar en la Figura 4.4 la moda en el chiste más recomendado es 6.75 con una frecuencia de 148 calificaciones.

Por último se realizó una gráfica de la calificación promedio para cada chiste. Como puede observarse en la figura 4.5 la calificación promedio para el chiste en el índice numero 50 es 6.75.

A partir de las gráficas y la información estadística obtenida del conjunto de datos es posible sugerir un razonamiento del por qué el algoritmo está arrojando los resultados mostrados en el cuadro 4.4. El chiste con el índice 50 tiene un número de calificaciones alto, ni la moda de la calificación ni la frecuencia de la moda es la más alta pero cuando se observa la gráfica detenidamente la combinación moda y frecuencia de la moda es de las más altas. Por ultimo, el valor promedio también se aproxima al valor recomendado por el algoritmo.

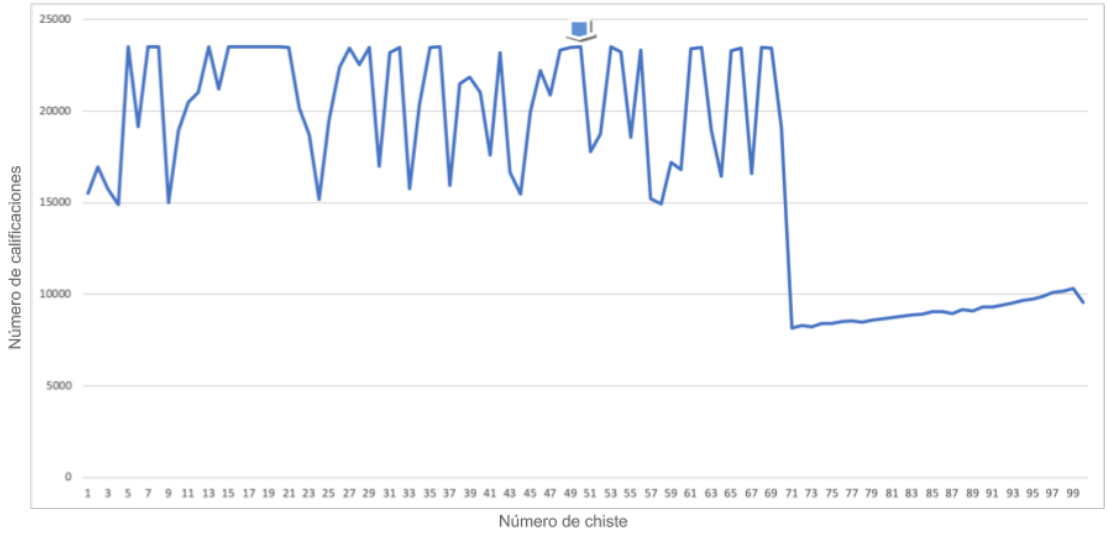

Figura 4.3: Numero de calificaciones por chiste.

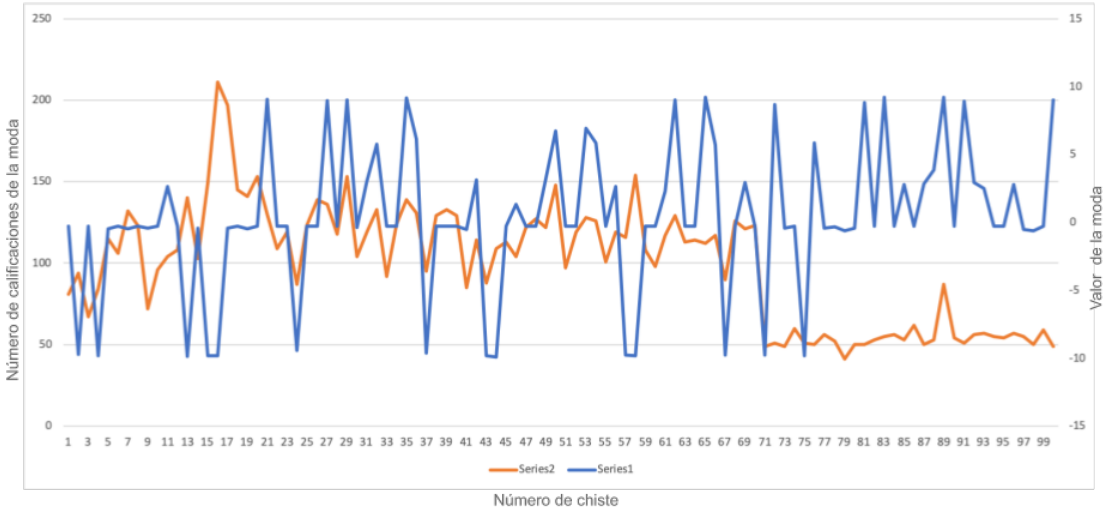

Figura 4.4: Moda(Color naranja) y frecuencia de la moda(Color azul) por cada chiste.

El razonamiento provee una visión del por qué el algoritmo está generando estos resultados, se analizó solo un caso ya que el número de permutaciones posibles es infinito; sin embargo, el algoritmo calculó los parámetros para generar recomendaciones para cada uno de los escenarios posibles.

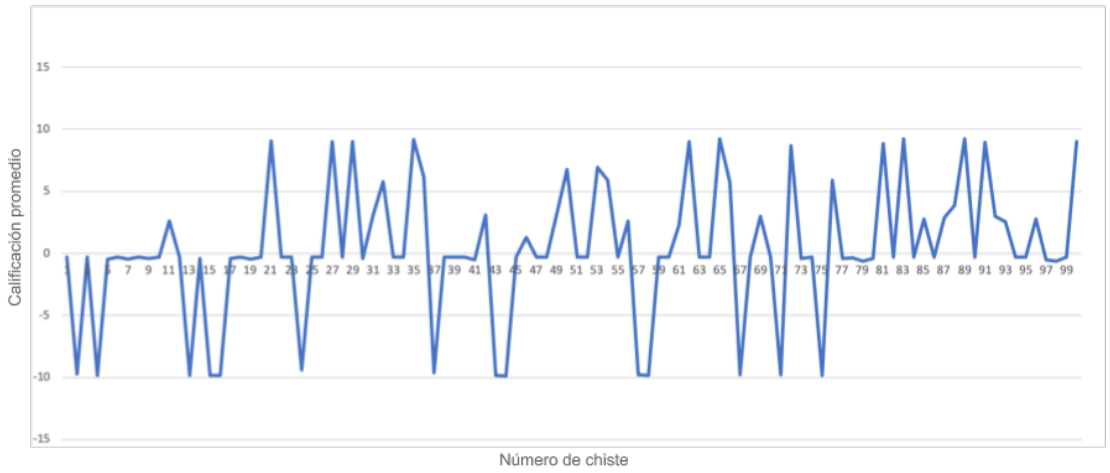

Figura 4.5: Calificaciones promedio para cada chiste.

4.3. Exactitud de las Recomendaciones

Para la evaluación de la exactitud de las recomendaciones generadas por el sistema, se utilizó el método Recall. Recall [Kai Yu et al., 2004] es el porcentaje de servicios o productos a los que el usuario le dio una calificación alta y que el algoritmo generó con una alta probabilidad de que al usuario le guste.

Con el objetivo de realizar una comparación de resultados, se realizó una implementación del método de evaluación Recall usando como referencia [Kai Yu et al., 2004]. A continuación se describe la implementación:

- Utiliza el conjunto de datos de validación(15 %).
- Ocultar 30 de las calificaciones de cada usuario activo aleatoriamente.
- Utilizar las calificaciones de cada usuario restantes como entrada al sistema de recomendación.
- Generar las recomendaciones.
- Extraer las N recomendaciones generadas con un mayor valor.
- Extraer las N recomendaciones con mayor valor ocultas en la entrada.
- Calcular el porcentaje de intersección en los conjuntos calculados previamente.

4.3.1. Comparación de resultados

Como referencia de comparación de resultados se utilizó la tabla de resultados de [Kai Yu et al., 2004], estos resultados se muestran en la tabla 4.5, y se resaltan los resultados más notorios que se obtuvieron con el método propuesto:

	TOP 5	**TOP 10**
Pearson correlations	0.251	0.454
Naıve Bayes	0.2350	0.4438
PMCF D	0.264	0.468
PMCF P	0.256	0.464
Autoencoder	**0.2784**	0.4407

Cuadro 4.5: Comparación de resultados usando el metodo Recall.

Como se muestra en en cuadro 4.5 de comparación de resultados, el método implementado en este trabajo fue más efectivo para el caso en el que se comparan los 5 resultados con mayor calificación.

4.4. Conclusiones

Existen diversas aplicaciones para los sistemas de recomendación en la industria y el objetivo de estas implementaciones puede ser variado. Por medio de los sistemas de recomendación es posible incrementar las ventas, generar tráfico en sitios web, crear clientes mas leales, etc., esto en el día a día representa una ventaja competitiva frente a otros competidores y es ahí donde reside su importancia.

En este trabajo se aplicaron técnicas de aprendizaje automático a los sistemas de recomendación, se empleó un auto-codificador con arquitectura profunda como una solución. Se demostró que el método aplicado es igual de efectivo e incluso en algunos casos supera otros resultados publicados, además de ofrecer recomendaciones a usuarios que interactúan con el sistema por primera vez (arranque en frío). De la misma forma, se demostró la viabilidad del proyecto, ya que aún cuando los tiempos de entrenamiento son largos, las predicciones son instantáneas obteniendo aún mejores resultados con el uso de una Unidad de Procesamiento Gráfico (GPU).

Estos resultados refuerzan la idea de implementar otras arquitecturas profundas en la búsqueda de soluciones en el ámbito de los sistemas de recomendación. Esto permitirá el desarrollo de sistemas más complejos y nuevas aplicaciones que no solo aporten a la industria, sino a nuestra sociedad.

4.5. Trabajo futuro

La investigación en sistemas de recomendación al día de hoy tiene un gran auge, entre los trabajos que se contemplan desarrollar a futuro están la implementación de otros casos de uso, la investigación y desarrollo aplicando otras técnicas de aprendizaje automático.

Algunas implementaciones importantes a desarrollar a futuro son:

- Desarrollo de un caso de uso real que genere un beneficio para la comunidad.
- Implementación de un conjunto de datos no denso, lo cual requiere diferentes hiper-parámetros y arquitectura.
- Sistema de recomendación basado en una Red Neuronal Recurrente.
- Sistema de recomendación basado usando Aprendizaje Reforzado.
- Sistema de recomendación basado en textos usando Procesamiento de Lenguaje Natural.

Bibliografía

[Bee, 2018] (2018). Sistema de recomendacion basado en filtros colaborativos.

[sky, 2019] (2019). Cloud based biometrics api as a service.

[Adomavicius and Tuzhilin, 2005] Adomavicius, G. and Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge & Data Engineering*, 1(6):734–749.

[Alpaydin, 2010] Alpaydin, E. (2010). *Introduction to Machine Learning*. The MIT Press, 2nd edition.

[Beel et al., 2015] Beel, J., Gipp, B., Langer, S., and Breitinger, C. (2015). Research paper recommender systems: A literature survey. *International Journal on Digital Libraries*, pages 1–34.

[Bennett et al., 2007] Bennett, J., Lanning, S., et al. (2007). The Netflix Prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, NY, USA.

[Beutel et al., 2018] Beutel, A., Covington, P., Jain, S., Xu, C., Li, J., Gatto, V., and Chi, E. H. (2018). Latent cross: Making use of context in Recurrent Recommender Systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 46–54. ACM.

[Caro Martínez, 2017] Caro Martínez, M. (2017). Sistemas de recomendación basados en técnicas de predicción de enlaces para jueces en línea. Master's thesis, Universidad Complutense de Madrid.

[Chapelle et al., 2006] Chapelle, O., Schlkopf, B., and Zien, A. (2006). *Semi-Supervised Learning (1st edition)*. The MIT Press, U.S.A.

[Chen et al., 2019] Chen, M., Beutel, A., Covington, P., Jain, S., Belletti, F., and Chi, E. H. (2019). Top-K Off-Policy Correction for a Reinforce Recommender System. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 456–464. ACM.

[Chen et al., 2015] Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C., and Zhang, Z. (2015). Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274.

[Cheng et al., 2016] Cheng, H.-T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., Anderson, G., Corrado, G., Chai, W., Ispir, M., et al. (2016). Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pages 7–10. ACM.

[Covington et al., 2016] Covington, P., Adams, J., and Sargin, E. (2016). Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198. ACM.

[CrowdAI, 2018] CrowdAI (2018). Crowdsourcing AI to solve real-world problems. `https://www.crowdai.org`. Fecha de consulta: 2019-02-26.

[Dietterich et al., 2008] Dietterich, T. G., Domingos, P., Getoor, L., Muggleton, S., and Tadepalli, P. (2008). Structured machine learning: the next ten years. *Machine Learning*, 73(1):3.

[Elahi et al., 2016] Elahi, M., Ricci, F., and Rubens, N. (2016). A survey of active learning in collaborative filtering recommender systems. *Computer Science Review*, 20:29–50.

[Erickson et al., 2017] Erickson, B. J., Korfiatis, P., Akkus, Z., Kline, T., and Philbrick, K. (2017). Toolkits and libraries for deep learning. *Journal of Digital Imaging*, 30(4):400–405.

[Geyik et al., 2018] Geyik, S. C., Guo, Q., Hu, B., Ozcaglar, C., Thakkar, K., Wu, X., and Kenthapadi, K. (2018). Talent search and recommendation systems at linkedin: Practical challenges and lessons learned. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 1353–1354. ACM.

[Gipp et al., 2009] Gipp, B., Beel, J., and Hentschel, C. (2009). Scienstein: A research paper recommender system. In *Proceedings of the international conference on emerging trends in computing (icetic'09)*, pages 309–315.

[Goldberg et al., 2001] Goldberg, K., Roeder, T., Gupta, D., and Perkins, C. (2001). Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151.

[Gonard, 2018] Gonard, F. (2018). *Cold-start recommendation:from Algorithm Portfolios to JobApplicant Matching*. Tesis de doctorado, l'Universit e Paris-Sud.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

[Haykin, 2009] Haykin, S. S. (2009). *Neural Networks and Learning Machines (1st edition)*. Prentice Hall.

[Jafarkarimi et al., 2012] Jafarkarimi, H., Sim, A. T. H., and Saadatdoost, R. (2012). A naive recommendation model for large databases. *International Journal of Information and Education Technology*, 2(3):216.

[Kaggle, 2010] Kaggle (2010). Kaggle is the place to do data science projects. `https://www.kaggle.com`. Fecha de consulta: 2019-02-26.

[Kai Yu et al., 2004] Kai Yu, Schwaighofer, A., Tresp, V., Xiaowei Xu, and Kriegel, H. . (2004). Probabilistic memory-based collaborative filtering. *IEEE Transactions on Knowledge and Data Engineering*, 16(1):56–69.

[Kenthapadi et al., 2017] Kenthapadi, K., Le, B., and Venkataraman, G. (2017). Personalized job recommendation system at linkedin: Practical challenges and lessons learned. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 346–347. ACM.

[Khan et al., 2018] Khan, S., Rahmani, H., Shah, S. A. A., and Bennamoun, M. (2018). A guide to convolutional neural networks for computer vision. *Synthesis Lectures on Computer Vision*, 8(1):1–207.

[Lee et al., 2009] Lee, C.-S., Chang, Y.-C., and Wang, M.-H. (2009). Ontological recommendation multi-agent for tainan city travel. *Expert Systems with Applications*, 36(3):6740–6753.

[Liao et al., 2009] Liao, S., Kao, K., Liao, I., Chen, H., and Huang, S. (2009). Pore: a personal ontology recommender system for digital libraries. *The Electronic Library*, 27(3):496–508.

[Linden et al., 2003a] Linden, G., Smith, B., and York, J. (2003a). Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, (1):76–80.

[Linden et al., 2003b] Linden, G., Smith, B., and York, J. (2003b). recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80.

[Lopez et al., 2016] Lopez, M. B., Montes, A. J. H., Ramirez, R. V., Hernandez, G. A., Cabada, R. Z., and Estrada, M. L. B. (2016). Emoremsys: an educational recommender system by using emotions detection. *RISTI (Revista Iberica de Sistemas e Tecnologias de Informacao)*, 1(17):80–96.

[Luberg et al., 2011] Luberg, A., Tammet, T., and Järv, P. (2011). Smart city: A rule-based tourist recommendation system. In Law, R., Fuchs, M., and Ricci, F., editors, *Information and Communication Technologies in Tourism 2011*, pages 51–62, Vienna. Springer Vienna.

[Lyubimov and Palumbo, 2016] Lyubimov, D. and Palumbo, A. (2016). *Apache Mahout: Beyond MapReduce*. CreateSpace Independent Publishing Platform, USA, 1st edition.

[Melville et al., 2002] Melville, P., Mooney, R. J., and Nagarajan, R. (2002). Content-boosted collaborative filtering for improved recommendations. *Aaai/iaai*, 23:187–192.

[Netflix, 2006] Netflix (2006). The Netflix Prize. `https://www.netflixprize.com`. Fecha de consulta: 2019-02-26.

[Nishioka and Ogata, 2018] Nishioka, C. and Ogata, H. (2018). Research paper recommender system for university students on the e-book system. *JCDL '18 Proceedings of the 18th ACM/IEEE on Joint Conference on Digital Libraries*.

[Odersky et al., 2004] Odersky, M., Micheloud, S., Mihaylov, N., Schinz, M., Stenman, E., Zenger, M., and et al. (2004). An overview of the scala programming language. Technical report.

[Polyak, 1964] Polyak, B. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1 – 17.

[Ramos-Pérez, 2016] Ramos-Pérez, E. (2016). Aprendizaje de representaciones de secuencia de aminoácidos utilizando arquitecturas profundas. Master's thesis, Universidad Tecnológica de la Mixteca.

[RecSys, 2012] RecSys (2012). ACM Recommender Systems community Challenge. `https://recsys.acm.org/challenges/`. Fecha de consulta: 2019-02-26.

[RecSys, 2019] RecSys (2019). ACM Recommender Systems community Challenge 2019. `http://www.recsyschallenge.com/2019`. Fecha de consulta: 2019-02-26.

[Reyes et al., 2009] Reyes, A., Enrique Sucar, L., and Morales, E. F. (2009). Asisto: A qualitative mdp-based recommender system for power plant operation. *Computación y Sistemas*, 13(1):5–20.

[Ricci et al., 2010] Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B. (2010). *Recommender Systems Handbook*. Springer-Verlag, Berlin, Heidelberg, 1st edition.

[Robbins and Monro, 1951] Robbins, H. and Monro, S. (1951). A stochastic approximation method. *Ann. Math. Statist.*, 22(3):400–407.

[Rubens et al., 2015] Rubens, N., Elahi, M., Sugiyama, M., and Kaplan, D. (2015). Active learning in recommender systems. In *Recommender systems handbook*, pages 809–846. Springer.

[Schafer et al., 1999] Schafer, J. B., Konstan, J., and Riedl, J. (1999). Recommender systems in e-commerce. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 158–166. ACM.

[Schedl et al., 2018] Schedl, M., Zamani, H., Chen, C.-W., Deldjoo, Y., and Elahi, M. (2018). Current challenges and visions in music recommender systems research. *International Journal of Multimedia Information Retrieval*, 7(2):95–116.

[Schein et al., 2002] Schein, A. I., Popescul, A., Ungar, L. H., and Pennock, D. M. (2002). Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 253–260. ACM.

[Smith and Linden, 2017] Smith, B. and Linden, G. (2017). Two decades of recommender systems at amazon. com. *Ieee internet computing*, 21(3):12–18.

[Synced, 2019] Synced (2019). Tensorflow, pytorch or mxnet? a comprehensive evaluation on nlp cv tasks with titan rtx.

[Tieleman and Hinton, 2012] Tieleman, T. and Hinton, G. (2012). Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning.

[Trivago, 2005a] Trivago (2005a). Online recommender systems: How does a website know what i want? `https://blogs.ams.org/mathgradblog/2015/05/25/online-recommender-systems-website-want/`. Fecha de consulta: 2019-02-26.

[Trivago, 2005b] Trivago (2005b). Trivago Naamloze Vennootschap. `http://www.trivago.com`. Fecha de consulta: 2019-02-26.

[Vargas Govea, 2018] Vargas Govea, B. (2018). Sistemas de recomendacion: Lecciones aprendidas y mejores practivas.

[Warlop et al., 2018] Warlop, R., Lazaric, A., and Mary, J. (2018). Fighting Boredom in Recommender Systems with Linear Reinforcement Learning. In *Advances in Neural Information Processing Systems*, pages 1764–1773.

[Watt et al., 2016] Watt, J., Borhani, R., and Katsaggelos, A. K. (2016). *Machine learning refined: foundations, algorithms, and applications*. Cambridge University Press.

[Wen and Shui-Sheng, 2006] Wen, Y. and Shui-Sheng, Y. (2006). A survey of collaborative filtering algorithm applied in e-commerce recommender system [J]. *Computer technology and development*, 16(9):70–72.

[Wiesner and Pfeifer, 2014] Wiesner, M. and Pfeifer, D. (2014). Health recommender systems: Concepts, requirements, technical basics and challenges. *International Journal of Environmental Research and Public Health*, 11(3):2580–2607.

[Zapata-Gonzalez et al., 2011] Zapata-Gonzalez, A., Menendez, V., Prieto-Méndez, M., and Romero, C. (2011). Using data mining in a recommender system to search for learning objects in repositories. In *Proceedings of the 4th international conference on educational data mining*, pages 321–322.

[Zhang et al., 2017] Zhang, S., Yao, L., and Sun, A. (2017). Deep learning based recommender system: A survey and new perspectives. *arXiv preprint arXiv:1707.07435*.

[Zhang et al., 2019] Zhang, S., Yao, L., and Sun, A. (2019). Deep learning based recommender system: A survey and new perspectives. *ACM Comput. Surv.*, 52:5:1–5:38.

[Zhao et al., 2015] Zhao, H., Gallo, O., Frosio, I., and Kautz, J. (2015). Loss functions for neural networks for image processing. *arXiv preprint arXiv:1511.08861*.

[Zheng et al., 2018] Zheng, L., Tan, Z., Han, K., and Mao, R. (2018). Collaborative Multi-modal deep learning for the personalized product retrieval in Facebook Marketplace. *arXiv preprint arXiv:1805.12312*.

Apéndice A

Código del auto-codificador para el sistema de recomendaciones.

A.1. Procesamiento del conjunto de datos

El código A.1 se ocupa para el pre-procesamiento de los datos. Este método toma como entrada el nombre del archivo con el conjunto de datos, los otros parámetros son opcionales y describen cómo se dividen los datos para el entrenamiento, las pruebas y la validación.

```
1  def convertMovieLens(fileName: String, trainPercent: Int = 70,
     validPercent: Int = 15, testPercent: Int = 15 ): Unit = {
2
3      //Definici n del prefijo de los archivos de salida.
4      val outputName = "Data_Tensor"
5
6      assert(trainPercent + validPercent + testPercent == 100, "Sum of
     percentages must be equal to 100.")
7
8      //Lectura de archivo de entrada.
9      val dataRaw = scala.io.Source.fromFile(new File(fileName))
10     val splittedData = dataRaw.getLines().map(_.split(",").map( str => {
11       val r = str.toFloat
12
13       //Normalizaci n del archivo de entrada.
14       if( r != 99) {
```

```
15          (r + 10) / 20
16        } else {
17          0
18        }
19
20      //Remover la primera columna.
21      }).drop(1))
22
23      val splittedDataList = splittedData.toList
24
25      //Re-ordenamiento aleatorio del conjunto de datos.
26      val (trainData, data) = Random.shuffle(splittedDataList).splitAt(
        splittedDataList.size * trainPercent / 100)
27      val (validData, testData) = data.splitAt(data.size * validPercent /(
        validPercent + testPercent))
28
29      //Almacenamiento de archivos.
30      saveToFile(trainData,  outputName + ".train" )
31      saveToFile(validData, outputName + ".valid" )
32      saveToFile(testData, outputName + ".test")
33
34    }
```

Código A.1: Código para el procesamiento de los datos.

El código A.2 muestra el método que permite la creación de la red, recibe como entradas el símbolo de la capa de entrada, un arreglo con las dimensiones del codificador, otro arreglo con las dimensiones del decodificador y un valor opcional para el filtro.

```
1   def buildDeepEncoder(input: Symbol, encoderDims: Option[Array[Int]],
2                        decoderDims: Option[Array[Int]], dropoutVal: Option
      [Float] = None): Symbol = {
3
4     val encoderSym = encoder(input, encoderDims)
5     val dropoutSym = dropout("enc", encoderSym, dropoutVal)
6     val decoderSym = decoder(dropoutSym, decoderDims)
7
8     decoderSym
9   }
```

Código A.2: Método de ayuda para la creación de la red.

El código A.3 muestra el método para la creación de un codificador, recibe como parámetros un símbolo de inicio, un arreglo con las dimensiones de las capas del codificador y el tipo de función de activación a utilizar.

```
1  def encoder(input: Symbol, dimsOpt: Option[Array[Int]], activation:
     String = "relu"): Symbol = {
2
3    dimsOpt match {
4      case Some(dims) => {
5        val encoderNetwork = createNetwork("enc", dims, input, activation)
6        encoderNetwork
7      }
8      case _ => throw new RuntimeException("Missing dims for encoding.")
9    }
10 }
```

Código A.3: Método para la creación de un codificador.

El código A.4 tiene un método para la creación de un decodificador, recibe como parámetros un símbolo de inicio, un arreglo con las dimensiones de las capas del codificador y el tipo de función de activación a utilizar.

```
1  def decoder(input: Symbol, dimsOpt: Option[Array[Int]], activation:
     String = "relu"): Symbol = {
2
3    dimsOpt match {
4      case Some(dims) => {
5        val decoderNetwork = createNetwork("dec", dims, input, activation)
6        decoderNetwork
7      }
8      case _ => throw new RuntimeException("Missing dims for decoding.")
9    }
10 }
```

Código A.4: Método para la creación del decodificador.

El código A.5 contiene un método para agregar un filtro a la red, recibe como parámetros un prefijo para el símbolo, el símbolo de inicio y un valor opcional con la probabilidad de ocultar la neurona durante el entrenamiento y permitir una mejor generalización.

```
1  def dropout(prefix: String, input: Symbol, percent: Option[Float]) :
     Symbol = {
2
3    percent match {
4      case Some(p) =>
5        Symbol.Dropout(s"${prefix}_dropout")()(Map("data" -> input, "p" ->
     p))
6      case _ =>
```

```
7           input
8       }
9   }
```

Código A.5: Método para implementar el *dropout*.

El método privado de ayuda para la creación de una red y anexar al símbolo de entrada que se muestra en el código A.6, recibe como parámetros un prefijo para el nombre de las capas, un arreglo con las dimensiones, el símbolo de inicio y el tipo de activación de la capa.

```
1   private[autoencoder] def createNetwork(prefix: String, dims: Array[Int],
2                                          input: Symbol, activation: String
    ): Symbol = {
3
4     val network = dims.zip(1 to dims.length).foldLeft(input){
5
6       (layer, idx) =>
7         val fc = Symbol.FullyConnected(s"${prefix}_dl${idx._2}")()(Map("
    data" -> layer, "num_hidden" -> idx._1))
8         val act = Symbol.Activation(s"${prefix}_act${idx._2}")()(Map("data
    " -> fc, "act_type" -> activation))
9         act
10    }
11
12    network
13  }
```

Código A.6: Método privado de ayuda para la creación de una red

El método que ejecuta el entrenamiento de la red neuronal se muestra en el código A.7, este método no requiere entradas. Realiza la carga del conjunto de datos de entrenamiento, hace uso de los métodos auxiliares que se definieron anteriormente para le creación del auto-codificador y realiza el entrenamiento durante 5000 iteraciones. Durante el entrenamiento se almacenan los parámetros del modelo periódicamente y al final se realiza el almacenamiento del modelo entrenado.

```
1   def trainRecommender() = {
2
3     //Definici n de las 5000 iteraciones de entrenamiento.
4     val epoch = 5000;
5     val ctx = Context.gpu();
6
7     //Carga de archivo de entrenamiento.
```

```
8     val dataIter = DataLoader.loadIterCSV("Data_Tensor.train", "(100)",
      batchSize = "1024")
9
10    //Definici n de las dimensiones de las capas del auto-decodificador.
11    val encoderDims = Some(Array(100, 64, 32, 32))
12    val decoderDims = Some(Array(32, 64, 100))
13    val dropOut = Some(0.95F)
14
15    val dataSymbol = Symbol.Variable("data")
16    val dataDropOut = Symbol.Dropout("dataDropOut")()(Map("data" ->
      dataSymbol, "p" -> 0.9F))
17
18    //Creaci n del s mbolo del auto-codificador.
19    val decoderSymbol = SymbolBuilder.buildDeepEncoder(dataDropOut,
      encoderDims, decoderDims, dropOut)
20
21    //Definici n de modulo auto-codificador.
22    val module = new AutoEncoderModule(decoderSymbol,
23      ctx,
24      dataIter.provideData,
25      Map("label" -> dataIter.provideData("data")),
26      //new SGD(learningRate = 0.0001F),
27      new RMSProp(learningRate = 0.0001F, wd = 0.0001f),
28      loss)
29
30    var mse = 0F;
31
32    //Iteraciones de entrenamiento.
33    for(i <- 0 until epoch) {
34
35      dataIter.reset()
36
37      while (dataIter.hasNext) {
38
39        val dataBatch = dataIter.next()
40        module.forward(dataBatch.data, dataBatch.data)
41        module.backward()
42        module.updateGradients()
43
44        val loss = module.getOutputs()(0)
45
46        mse = NDArray.sum(loss).toArray(0)
47
48      }
49
50      if(i > 0 && i %10 == 0) {
```

```
51            module.saveCheckpoint("DeepRecommenderDropout", i)
52          }
53
54          println(s"Iteration: ${i} MSE metric is: ${mse}")
55        }
56
57        module.saveCheckpoint("DeepRecommenderDropout_Time", epoch)
58      }
```

Código A.7: Método para el entrenamiento de la red.

El método para probar del auto-codificador utilizando el vector 0 se muestra en el código A.8. Éste realiza la carga de las dimensiones del conjunto de datos, carga el modelo pre-entrenado para su ejecución y realiza la predicción. Al final se realiza el ordenamiento para obtener los 10 resultados con más alta probabilidad y se realiza la impresión de los resultados.

```
1   def testRecommender(): Unit = {
2
3     //Carga las dimensiones del conjunto de datos.
4     val dataIter = DataLoader.loadIterCSV("TrainData.csv", "(1,1,100)",
      batchSize = "20")
5     val dataShape = dataIter.provideData("data")
6
7     //Carga el modelo pre-entrenado para su ejecuci n.
8     val module = AutoEncoderModule.loadModel(dataIter.provideData("data"),
      "DeepRecommenderDropout", 5000)
9
10    //Creaci n de un vector 0 de dimensi n 100 e inicializa el lote con
      solo un elemento.
11    val zeros = NDArray.zeros(Shape(1,1,1,100))
12    val dataBatch = new DataBatch(IndexedSeq(zeros),IndexedSeq(zeros),
      IndexedSeq(0),0)
13
14    val startTime = Calendar.getInstance().getTime().getTime()
15
16    //Realiza la predicci n
17    val predicted = module.predict(dataBatch)
18    val s = predicted.map(n => Shape(n.shape(0), n.shape(1))).head
19
20    //Realiza el enmascaramiento de la salida.
21    val masked = NDArray.reshape(dataBatch.data(0), s).<=(0)
22    val outputMasked = masked * predicted(0)
23
```

```
24      print(s"Prediction time(ms): ${Calendar.getInstance().getTime().
        getTime - startTime}")
25
26      //Indeza los resultados.
27      val output = outputMasked.toArray.grouped(100).toArray
28      val idxed = output(0).zipWithIndex
29
30      //Ordena los valores basado en la probabilidad en orden descendente.
31      val res = idxed.sortWith((v1 , v2) => v1._1 > v2._1).take(10)
32
33      //Imprime los resultados.
34      res.foreach( v => println(s"Probabilidad: ${v._1} Indice: ${v._2}"))
35
36    }
```

Código A.8: Método para probar del auto-codificador utilizando el vector 0.

Finalmente, el método de verificación de resultados usando el método Recall se muestra en el código A.9, solo se reciben como parámetros el número de resultados que se desea comparar. Realiza la carga del conjunto de datos de validación, se carga el modelo pre-entrenado y ejecuta el método Recall para la verificación.

```
1  . lastline. lastline
2    def validateRecall(topValues: Int): Unit = {
3
4      //Carga los datos de validaci n.
5      val dataIter = DataLoader.loadIterCSV("Data_Tensor.test", "(1,1,100)",
       batchSize = "20")
6
7      //Carga el modelo pre-entranado.
8      val module = AutoEncoderModule.loadModel(dataIter.provideData("data"),
       "DeepRecommenderDropout", 5000)
9
10     var arrResult = List[Float]()
11
12     while(dataIter.hasNext) {
13
14       //Obtiene el primer lote de elementos.
15       val batch = dataIter.next()
16
17       //Remueve los valores aleatorios y almacena el nuevo arreglo, los
       calificaciones mas altas que se
18       //removieron, y los indices de los valores removidos.
19       val output = batch.data.head.toArray.grouped(100).map(
       removeRandomValues(topValues)).toArray
```

```
20        val array = output.map(_._3).flatten
21        val topIdx = output.map(_._2)
22        val idxs = output.map(_._1)
23
24        //Se crea el lote con los nuevos valores.
25        val ndArray = NDArray.array(array, Shape( 20,1,1,100))
26        val dataBatch = new DataBatch(IndexedSeq(ndArray), batch.data,
    IndexedSeq(0),0)
27
28        val predicted = module.predict(dataBatch)
29
30        val result = predicted(0).toArray.grouped(100).toArray
31
32        //Se realiza el calculo del metodo Recall.
33        val matched = result.zipWithIndex.map {
34          v =>
35            val topTen = v._1.zipWithIndex
36            val rec = topTen.filter(t => idxs(v._2).contains(t._2))
37            val topRec = rec.sortBy(-_._1).take(topValues)
38            (topRec.map(_._2).toSet.intersect(topIdx(v._2)).size.toFloat /
    topValues)
39        }
40
41        arrResult = arrResult ++ matched
42
43      }
44
45      println("Recall: " + arrResult.sum / arrResult.length)
46
47
48    }
49
50    def removeRandomValues(topValues: Int)(arr: Array[Float]) = {
51
52      val arrIdx = arr.zipWithIndex
53      val top = Random.shuffle(arrIdx.filter( _._1 > 0).toList).take(30)
54      val topIdx = top.map(_._2).toSet
55
56      val resp = arrIdx.map(v => if(topIdx.contains(v._2)) (0.0.toFloat, v.
    _2) else v).map(_._1)
57
58      (topIdx, top.sortBy(-_._1).take(topValues).map(_._2).toSet, resp)
59    }
60
61
62    def main(args: Array[String]): Unit = {
```

```
63
64      //Extraer el archivo y separar el conjunto en prueba, validacion y
        pruebas
65      DataLoader.convertMovieLens("/home/favalos/Downloads/jester-data-2.csv
        ", trainPercent = 90, validPercent = 5, testPercent = 5)
66
67      //medida del tiempo de entrenamiento
68      print(s"Training stat time: ${Calendar.getInstance().getTime()}")
69      trainRecommender()
70      print(s"Training end time: ${Calendar.getInstance().getTime()}")
71
72      //Probar el vector cero
73      testRecommender()
74
75      //Validar el metodo recall
76      validateRecall(10)
77
78    }
79
80  }
```

Código A.9: Método de verificación de resultados usando el método Recall