UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA

La Universidad Católica de Loja

ÁREA TÉCNICA

TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Implementación de algoritmos de clasificación con Deep Learning a través del uso de la librería TensorFlow.

TRABAJO DE TITULACIÓN.

AUTOR: Vivanco Castro, Juan Sebastian

DIRECTORA: Cabrera Loayza, María del Carmen, Mgtr.

LOJA - ECUADOR

2018

2018

APROBACIÓN DEL DIRECTOR DEL TRABAJO DE TITULACIÓN

Magister.
María del Carmen Cabrera Loayza
DOCENTE DE LA TITULACIÓN

De mi consideración:

El presente trabajo de titulación: Implementación de algoritmos de clasificación con Deep Learning a través del uso de la librería TensorFlow realizado por Juan Sebastian Vivanco Castro, ha sido orientado y revisado durante su ejecución, por cuanto se aprueba la presentación del mismo.

Loja, diciembre del 2018

f).....

DECLARACIÓN DE AUTORÍA Y CESIÓN DE DERECHOS

Yo Juan Sebastian Vivanco Castro declaro ser autor (a) del presente trabajo de titulación: Implementación de algoritmos de clasificación con Deep Learning a través del uso de la librería TensorFlow, de la Titulación de Informática, siendo María del Carmen Cabrera Loayza director (a) del presente trabajo; y eximo expresamente a la Universidad Técnica Particular de Loja y a sus representantes legales de posibles reclamos o acciones legales. Además certifico que las ideas, conceptos, procedimientos y resultados vertidos en el presente trabajo investigativo, son de mi exclusiva responsabilidad.

Adicionalmente declaro conocer y aceptar la disposición del Art. 88 del Estatuto Orgánico de la Universidad Técnica Particular de Loja que en su parte pertinente textualmente dice: "Forman parte del patrimonio de la Universidad la propiedad intelectual de investigaciones, trabajos científicos o técnicos y tesis de grado o trabajos de titulación que se realicen con el apoyo financiero, académico o institucional (operativo) de la Universidad."

f.....
Autor: Juan Sebastian Vivanco
Castro: Cédula 1104786817

DEDICATORIA

Dedico este trabajo a quienes me acompañaron en este viaje en la vida universitaria, a mis padres por guiarme y brindarme su apoyo en cada momento guiándome en mi vida y formación profesional, porque gracias a su esfuerzo he tenido la dicha de culminar con mis estudios. A todos mis seres queridos que gracias a su presencia he crecido y he podido formarme personalmente.

Juan Sebastian

AGRADECIMIENTO

Agradezco primeramente a Dios por sus darme bendiciones, salud en todo momento y permitirme dar un paso más en mis metas, a mis padres, quienes me brindaron apoyo económico y moral durante el transcurso de mi carrera profesional, a mi familia en general que compartieron sus conocimientos y experiencias conmigo, de manera muy especial a mi directora de tesis que sin su paciencia, consejos y compromiso permitió culminar con el presente trabajo.

Juan Sebastian

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

En este trabajo se realizó pruebas con algoritmos utilizando el método de Redes Neuronales de Deep Learning con la librería TensorFlow sobre un conjunto de datos de 6 millones de logs de acciones ejecutadas por estudiantes en la plataforma OpenCampus basada en OpenEdx de la Universidad, la depuración de datos fue realizada con la librería Pandas por la necesidad del algoritmo de tratar los datos de manera manual al ser semi-supervisado, para las pruebas la evaluación del algoritmo con la Red Neuronal Convolucional fue ejecutado con 1 millón de logs. Se examinó dos clasificaciones una por tipo de evento en relación al estudiante y curso, otra por el estado de aprobación de los estudiantes por cada curso. Los datos analizados sugieren que la mayor cantidad de los estudiantes presentan dificultades al aprobar los cursos. La metodología utilizada fue adaptada de la metodología original de KDD para aprovechar su uso con Deep Learning.

PALABRAS CLAVES: Aprendizaje Automático; Inteligencia Artificial; TensorFlow; algoritmos de clasificación; Redes Neuronales; Red Neuronal Convolucional; OpenEdx; OpenCampus.

ABSTRACT

In this work we performed tests with algorithms using the Deep Learning Neural Networks method with the TensorFlow library on a data set of 6 million logs of actions executed by students in the OpenCampus platform based on OpenEdx of the University, the data debugging was performed with the Pandas library for the need of the algorithm to treat data manually to be semi-supervised, for tests the evaluation of the algorithm with the Neuronal Network Convolucional was executed with 1 million logs. Two classifications were examined, one by type of event in relation to the student and course, the other by the approval status of the students for each course. The analyzed data suggest that the greatest number of students have difficulties in passing the courses. The methodology used was adapted from KDD's original methodology to take advantage of its use with Deep Learning.

KEY WORDS: Deep Learning; Artificial Intelligence; TensorFlow; classification algorithms; Neural Networks; Convolutional Neural Network; OpenEdx; OpenCampus.

CAPÍTULO 1

INTRODUCCIÓN

1.1 Introducción

El presente Trabajo de Titulación se centra en la implementación de algoritmos de clasificación con metodos de Deep Learning, que pueden hacer uso de la librería TensorFlow y, más concretamente, en conjunto de datos de texto contribuyendo en el campo de Deep Learning. Los datos se obtuvieron de la plataforma OpenCampus de la Universidad que contiene logs sobre acciones que realizan estudiantes sobre los cursos con los que interactúan. Para ello se aborda, desde una metodología adaptada de KDD, partiendo de la depuración del conjunto de datos de los últimos 3 años, luego se analiza por pruebas el algoritmos más apto y, de manera particular, el proceso de entrenamiento y evaluación con la implementación de este algoritmo con el uso de Redes Neuronales, el proceso constituye en el aprendizaje automático semi-supervisado.

El potencial de estas clasificaciones depende de diversas variables, entre ellas los tipos de eventos de las acciones sobre la plataforma y de la relación que se produce entre el estado de aprobación que se encuentran los estudiantes. Profundizar la clasificación y los métodos de clasificación de Deep Learning junto a la librería TensorFlow constituye el motivo central de este Trabajo de Titulación.

1.2 Problemática

El tratamiento de grandes volúmenes de datos de diferentes categorías en la actualidad, traen una necesidad al momento de ser organizados, los algoritmos de clasificación tienen la característica de trabajar junto a la rama de Machine Learning llamada Deep Learning, que a través de su metodología de redes neuronales puedan tomar una muestra de datos previamente clasificados, trabajar en los datos no organizados e ir evolucionando dentro de un rango especifico y de esta manera automatizar el proceso de clasificación de grandes cantidades de datos que hasta ahora comúnmente se lo realiza de manera manual.

1.3 Justificación

TensorFlow es una biblioteca de código abierto del lenguaje de programación Python que se basa precisamente en la metodología o sistema de redes neuronales pudiendo relacionar varios datos en red simultáneamente, como un cerebro humano, permitiendo reconocer y relacionar los datos de una manera más rápida de lo que comúnmente se hace. De esta manera se puede vincular los procesos investigativos acerca de los grandes volúmenes de datos con el uso de nuevas tecnologías, que es lo que el presente trabajo de investigación busca.

1.4 Objetivos

El presente Trabajo de Titulación ha definido el cumplimiento de los siguientes objetivos:

General

- Implementar algoritmos de clasificación con Deep Learning a través del uso de la librería TensorFlow.

Específicos

- Comprender los fundamentos teóricos de Deep Learning en el ámbito de algoritmos de clasificación.
- Analizar y seleccionar los métodos y propiedades vinculados a algoritmos clasificación de la librería TensorFlow.
- Analizar, especificar y depurar un conjunto de datos orientado a Deep Learning para generar clasificaciones aplicando la librería TensorFlow.
- Implementar y realizar pruebas sobre los algoritmos seleccionados a aplicar en el conjunto de datos.

- Analizar y comparar los resultados obtenidos de cada prueba obtenida a través de los algoritmos de clasificación.

1.5 Metodología

Para desarrollo del presente Trabajo de Titulación se van a considerar los siguientes aspectos:

- Búsqueda de información de la temática en fuentes relevantes.
- Análisis de trabajos relacionados.
- Especificación de fuentes de información.
- Implementación de un entorno Python con librerías TensorFlow.
- Pruebas de algoritmos de clasificación de Deep Learning.
- Análisis e interpretación de resultados.

1.6 Estuctura del documento

En el Capítulo 1 se hace la presentación formal de la problemática que da raíz al tema, la relación entre el objetivo general, los objetivos específicos y la metodología utilizada en el Trabajo de Titulación.

En el Capítulo 2 constituye el encuadre del problema que se va a tratar dentro de limitantes teóricas y constituyen el punto de partida para orientar el desarrollo de la investigación.
Se incluyen teorías relacionadas, estudios relacionados, problemas relacionados, resultados obtenidos y situación actual del problema de los algoritmos de clasificación, la librería de Deep Learning Tensorflow y las Redes Neuronales.

En el Capítulo 3 incluye los aspectos metodológicos aplicados en el desarrollo del Trabajo de Titulación, los algoritmos que se seleccionó, el dataset y plataforma a utilizar,

tipo de Red Neuronal y herramientas como el leguanje de programación Python que junto a la librería Pandas y la implementación del algoritmo en TensorFlow permiten hacer congruente y coherente la medición de las variables, se describen de forma clara la clasificación y dentro de esta parte se muestran todas las pruebas realizadas que determinan como terminado el desarrollo del tema de Trabajo de Titulación y por lo tanto, el cumplimiento de los objetivos propuestos.

En el Capítulo 4 se integra los hallazgos principales de la investigación realizada y los pone en el contexto del conocimiento actual o plataformas actuales como openEdx. Esta sección incluye dentro del texto, las figuras, tablas, y otros que sean relevantes con la temática tratada de la implementación del algoritmo de clasificación.

En la presentación de resultados se encuentran dos principales secciones, la primera es referida a clasificación por eventos la cual permite tener una visión panorámica de cuáles fueron las iteraciones sobre la plataforma (tendencias a aprobar y reprobar los cursos), mientras que en la segunda sección ya orientada la clasificación de estudiantes aprobados y reprobados.
Finalmente se presentan las conclusiones y recomendaciones resultado del desarrollo del Trabajo de Titulación.

CAPÍTULO 2

MARCO TEÓRICO

En este capítulo se presenta el marco teórico que soporta el Trabajo de Titulación a desarrollar. En la sección 2.1 se hace referencia a los fundamentos teóricos de Deep Learning, desde su historia hasta las Redes Neuronales y sus arquitecturas. En la sección 2.2 se hace una introducción a los algoritmos de predicción y clasificación que maneja DL. La sección 2.4 se enfoca en describir las librerías de DL. Al finalizar este capítulo en la sección 2.5 se demuestra trabajos que tengan relación con el proyecto a desarrollar.

2.1 Fundamentos Teóricos de Deep Learning

En esta sección se observa una breve síntesis histórica de Deep Learning (DL), principalmente desde la perspectiva de cómo varias tecnologías han sido impactadas por DL, y cómo empezó a tener mayor impulso y valor durante la última década.

2.1.1 Historia

“Históricamente, el concepto de DL se originó a partir de la investigación de Redes Neuronales artificiales. Pero DL no pudo entrar a arquitecturas profundas antes de 2006” (Bengio, 2009).

“Desde Machine Learning (ML) hasta DL, perceptron fue descubierto por Rosenblatt en 1957” (Rosenblatt, 1958), pero en 1969, Minsky y Papert demostraron que una sola neurona artificial es incapaz de implementar algunas funciones como la función lógica XOR.

Posteriormente, algunos investigadores intentaron hacer que el algoritmo perceptron no fuera lineal, pero no se encontraron resultados hasta el descubrimiento del Multi Layer Perceptron (MLP) con los algoritmos Back Propagation (BP) de Rumelhart, Hinton y Williams en 1986 (LeCun et al., 1989; Rumelhart et al., 1986). “Desafortunadamente, BP por sí solo no fue poderosa en la práctica para una red de aprendizaje”

(Glorot and Bengio, 2010), y muy lenta cuando el número de capas ocultas es muy alto.

Muchos investigadores abandonaron las Redes Neuronales en la década de 1990 con múltiples capas ocultas adaptativas porque SVM (Vapnik y sus colaboradores desarrollaron Máquinas de Vectores de Soporte en 1993, una aquitectura superficial) (Cortes and Vapnik, 1995) funcionó mejor, y no hubo intentos exitosos de entrenar redes profundas.

A pesar de esto, LeCun, en 1998, tuvo cierto éxito con las Redes Neuronales convolucionales y ahora es ampliamente utilizado para el reconocimiento de imágenes y vídeo (Deng et al., 2013; Lecun et al., 1998a).

El avance de DL fue en 2006 con la aparición de máquinas más rápidas y nuevos métodos para el pre-entrenamiento sin supervisión (Chen et al., 2014; Hinton et al., 2006). Quienes lideraron el avance de DL fueron Hinton, Bengio y LeCun. Hinton utilizó Máquinas Restringidas de Boltzmann (RBM) como modelos generativos de muchos tipos diferentes de datos, incluidas las imágenes etiquetadas o no etiquetadas.

Los AutoEncoders (AE) fueron desarrollados por Bengio (Liou et al., 2014) y se han utilizado para aprender la codificación eficiente. Yun LeCun inventó las representaciones de sparce para la clasificación de imágenes y el reconocimiento de objetos (Jarrett et al.).

“Desde 2006, Deep Structured Learning, o más comúnmente llamado DL o Hierarchical Learning, se ha convertido en una nueva área de Machine Learning Research” (Bengio, 2011).

2.1.2 Deep Learning

DL es un subcampo del ML inspirado en la estructura y el funcionamiento del cerebro humano llamado Redes Neuronales artificiales.
DL es una nueva área de investigación de ML, que se ha introducido con el objetivo de acercar ML a uno de sus objetivos originales: la Inteligencia Artificial (IA). DL trata de aprender múltiples niveles de representación y abstracción que ayudan a dar sentido a datos tales como imágenes, sonido y texto.

DL es una familia de métodos que utilizan arquitecturas profundas para aprender representaciones de funciones de alto nivel, utilizando varias capas para crear un espacio de funciones mejoradas (Zhang, 2015).

La familia de los métodos de DL se ha ido enriqueciendo cada vez más, abarcando las Redes Neuronales, modelos probabilísticos jerárquicos y una variedad de algoritmos de aprendizaje supervisados y no supervisados. DL es una clase de técnicas de ML que explotan muchas capas de procesamiento de información no lineal para la extracción y transformación de funciones supervisadas o no supervisadas, y para el análisis y la clasificación de patrones (Deng, 2014).

El propósito del DL es calcular las características jerárquicas o representaciones de los datos de observación, donde las características o factores de nivel superior se definen a partir de los de nivel inferior.

El Hype Cycle para las tecnologías emergentes, 2017 proporciona información obtenida a partir de evaluaciones de más de 2.000 tecnologías que las firmas de investigación y asesoramiento siguen. A partir de esta gran base de tecnologías, las tecnologías que muestran el mayor potencial para ofrecer una ventaja competitiva en los próximos 5 a 10 años se incluyen en el Hype Cycle que se muestra en la Figura 2.1.

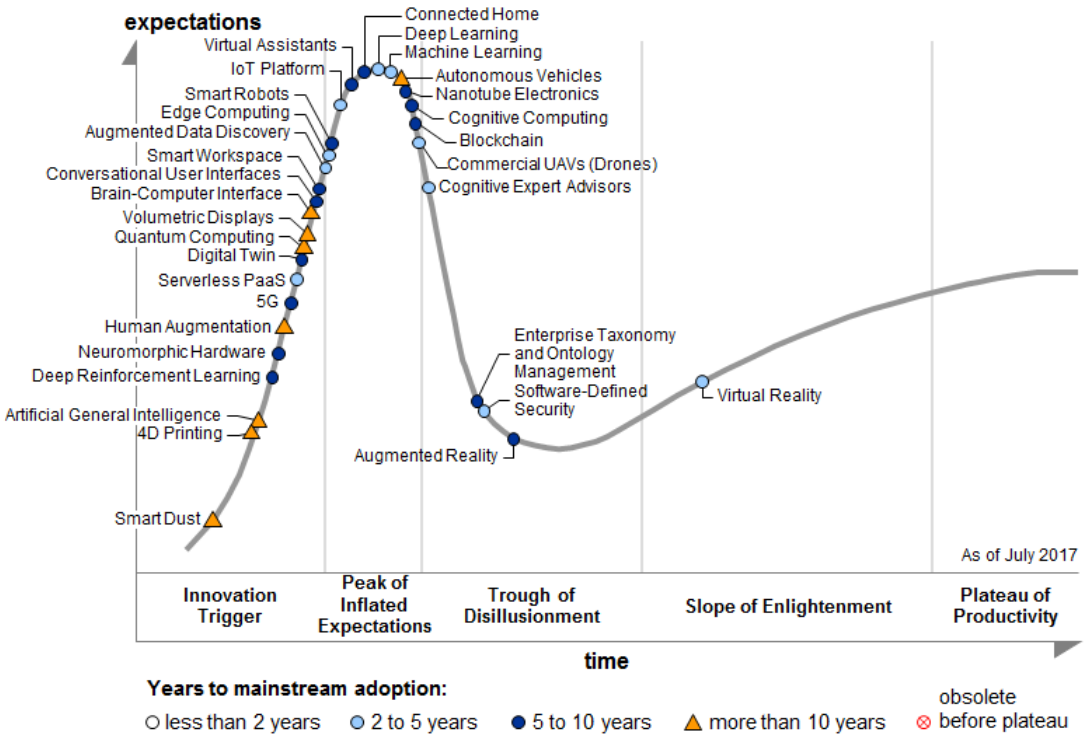


Figure 2.1: Hype Cycle de Gartner
Fuente: (Stamford, 2017)
Elaborado por: (Stamford, 2017)

Las ocho tecnologías agregadas al Hype Cycle de este año incluyen Deep Learning, 5G, Inteligencia Artificial General, Deep Refforcement Learning, Digital Twin, Edge Computing, Serverless PaaS y Cognitive Computing.

Las tres tendencias más dominantes incluyen la IA en todas partes, donde el gasto global destinado para el Desarrollo e Invetigación (R&D) de Amazon, Apple, Baidu, Google, IBM, Microsoft y Facebook está impulsando hoy una carrera para patentes DL y ML, y se acelerará en el futuro.
La carrera es para la Propiedad Intelectual de DL y ML que continuan hoy en día como las mayores tendencias. Gartner predice que las aplicaciones y herramientas de DL serán un componente estándar en el 80% de las cajas de herramientas de los científicos de datos para la siguiente década.

La Inteligencia Artificial General durante la próxima década se volverá penetrante, convirtiéndose en la base de la IA como servicio: Gartner predice que la IA como servicio será la tecnología central que posibilitará la convergencia de AI en todas partes, experiencias inmersivas de forma transparente y plataformas digitales (Stamford, 2017).

2.1.3 Relación con la Inteligencia Artificial

El objetivo principal de la Inteligencia Artificial (IA) es desarrollar una máquina que sea capaz de sentir, reconocer, aprender y recordar como un ser humano real. La IA está moviendo el diseño de más algoritmos correctos de aprendizaje de representación para implementar tales prioridades. Por lo tanto, la IA se movió hacia el método de ML, que depende en gran medida de la elección de la representación de datos en la que se aplican.
En esta tarea, gran parte del esfuerzo real en la implementación de algoritmos de ML se aplica al diseño de tuberías de pre-procesamiento y transformaciones de datos que da como resultado una representación de los datos que respaldan el ML efectivo (Bengio, 2009).

Dicha ingeniería de características es importante pero exige mucho trabajo y resalta los inconvenientes de los algoritmos de aprendizaje actuales: su incapacidad para organizar y extraer la información de los datos.

La ingeniería de funciones es una forma de aprovechar el ingenio humano, aumentar la facilidad y el alcance de la aplicabilidad de ML, permitiendo hacer que los algoritmos de aprendizaje estén menos sujetos a la ingeniería de características, de modo que las nuevas aplicaciones puedan construirse más rápido y, más significativamente, de esta manera se acerca cada vez más ML y cada una de sus tecnologías como es DL a la IA (Poonia et al., 2016).

El aprendizaje es la representación de los datos que hacen que sea más fácil ex-

traer información útil al construir clasificadores o predictores. DL es un método de aprendizaje con la arquitectura profunda y los algoritmos de aprendizaje, que pueden realizar el aprendizaje intelectual (Lee et al., 2009). Esta capacidad, señala una nueva dirección hacia la IA junto con los algoritmos de aprendizaje eficientes.

DL es una nueva área de investigación de ML, que se ha incluido con el objetivo de acercar ML a uno de sus objetivos: la IA. DL se trata de aprender múltiples niveles de representación que ayudan a dar sentido a los datos, como imágenes, sonido y texto (Bhatia and Rana, 2015).

2.1.4 Funcionamiento

En esta sección se analiza primero las Redes Neuronales y luego las arquitecturas profundas para entender cómo funciona DL.

2.1.4.1 *Redes Neuronales*

DL es solo un nombre dado a un conjunto de algoritmos que usan una Red Neuronal como arquitectura. Las Redes Neuronales también tienen una larga historia; tienen más éxito en los últimos años debido a la disponibilidad de hardware paralelo como GPU y clústeres de computadoras y gran cantidad de datos (Arel et al., 2010).

- **Perceptrones**

Es la primera generación de Redes Neuronales, que trata de implementar el reconocimiento de objetos mediante el aprendizaje de vectores de peso, que combina todas las características de Redes Neuronales de alguna manera. Este primer Perceptrón solo consiste en una capa de entrada, una capa de salida y una característica fija en el medio.
Su capacidad es de clasificar algunas formas básicas como triángulos y cuadrados. Fue desarrollado para que las personas vean el potencial que una máquina inteligente real que puede sentir, aprender, recordar y reconocer como seres humanos.

Una de sus limitaciones, es que la capa característica de este Perceptrón es fija y fue creada por seres humanos, lo que va en contra de la definición de una máquina "inteligente" real. Otra razón es su estructura de tener una capa que limita las funciones que puede aprender (Shalev-Shwartz and Singer, 2005).

- **Redes Neuronales con capas ocultas**

Luego de la aparición del Perceptrón, se reemplazó la capa de características fijas únicas de este, con varias capas ocultas, creando la Red Neuronal de segunda generación. Debido a sus muchas capas, también se lo denomina Perceptrón Multi-Capa (MLP).

La Red Neuronal puede aprender funciones más complicadas a través del algoritmo BP que propaga la señal de error computada en la capa de salida, para actualizar los vectores de peso hasta que se alcanza la convergencia (Le, 2015). El alcance de aprendizaje de esta Red Neuronal se extiende debido a las estructuras multi-capa, pero tiene algunas desventajas (Mo, 2012):

1. La Red Neuronal no pudo entrenar los datos no etiquetados mientras que en el entrenamiento la mayoría de los datos no están etiquetados.
2. La señal de corrección se debilitará cuando vuelva a pasar a través de múltiples capas.
3. Debido a su estructura (múltiples capas ocultas), el aprendizaje es demasiado lento.

- **Máquinas de Vectores de Soporte** (SVM)

Reemplazando la capa de características hechas a mano en los Perceptrones originales, se crea un nuevo modelo SVM, siguiendo una función llamada kernel, que mapea los datos de entrada en otro espacio de alta dimensión (Shalev-Shwartz and Singer, 2005).

Su estructura simple hace que el aprendizaje sea rápido y fácil. Sin embargo, su estructura simple tiene un peor rendimiento para los datos que a su vez contienen características complicadas.
Existe una forma de resolver este problema: agregar un conocimiento previo al modelo SVM para obtener una mejor capa de características. El SVM tiene tres desventajas (Mo, 2012):

1. El modelo anterior no aprende el conocimiento previo, sino que se lo explica al modelo.

2. Es difícil encontrar un conjunto general de conocimientos previos, que se puede aplicar incluso al mismo tipo de problema.

3. No hay garantía de que el conocimiento previo agregado al modelo sea correcto, en tal caso el conocimiento previo puede confundir los procedimientos de aprendizaje y dar como resultado un modelo deficiente.

   Por lo tanto, necesitamos una arquitectura que pueda aprender las características a partir de los datos que se le brindan, y también puede tratar con los datos no etiquetados.

Para entrenar una Red Neuronal, se utiliza un proceso llamado BP. Calcula el gradiente de derecha a izquierda(Bengio and LeCun, 2007). Si la red es más grande de lo que lleva demasiado tiempo entrenar y a menudo da resultados inexactos, se debe utilizar un método llamado RBM, que puede encontrar automáticamente patrones en los datos mediante la reconstrucción de la entrada.
Se denomina restringido porque ningún nodo de la misma capa está conectado entre sí (Lee et al., 2009). RBM es una capa visible de la red de dos capas y una capa oculta:

1. En el pase hacia adelante, RBM toma las entradas y las traduce en un conjunto de números que codifica la entrada.

2. En el pase hacia atrás, toma un conjunto de números y los traduce de vuelta para formar las entradas reconstruidas. En RBM, resulta ser muy importante los conjuntos de datos del mundo real, como fotos, voces, datos de sensores y todos los cuales tienden a estar sin etiqueta.

En la actualidad, muchas arquitecturas profundas de DL utilizan RBM para realizar tareas más precisas.

2.1.4.2 *Arquitecturas Profundas*

Existen diferentes arquitecturas profundas, pero Deep Belief Network (DBN) en español Red de Creencia Profunda y Convolutional Neural Network (CNN) en español Red Neuronal Convolucional son los dos hitos más importantes porque están bien establecidos en el campo de DL y muestran una gran promesa para el trabajo futuro en este campo según (Arel et al., 2010).

- **Red de Creencia Profunda**

Al igual que RBM, DBN también es concebido como una alternativa a BP. DBN se puede ver como una pila de RBM donde la capa oculta de un RBM es la capa visible de otra capa (Salakhutdinov and Hinton).

DBN es un modelo híbrido que consta de dos partes. Los dos niveles superiores son un modelo de gráfico no dirigido que forma la memoria asociativa, y las capas restantes son un modelo de gráfico dirigido que es una RBM.

En general, las DBN se pueden entrenar como un modelo discriminativo (problema de inferencia, problema de clasificación, etc.) o un modelo generativo (generar datos de entrenamiento, etc.). Las DBN se componen de varias capas de RBMs Figura 2.2. Estas redes están “restringidas” a una sola capa visible y a una sola capa oculta, donde las conexiones se forman entre las capas. las unidades ocultas capturan las correlaciones de datos de orden superior dentro de las unidades visibles.

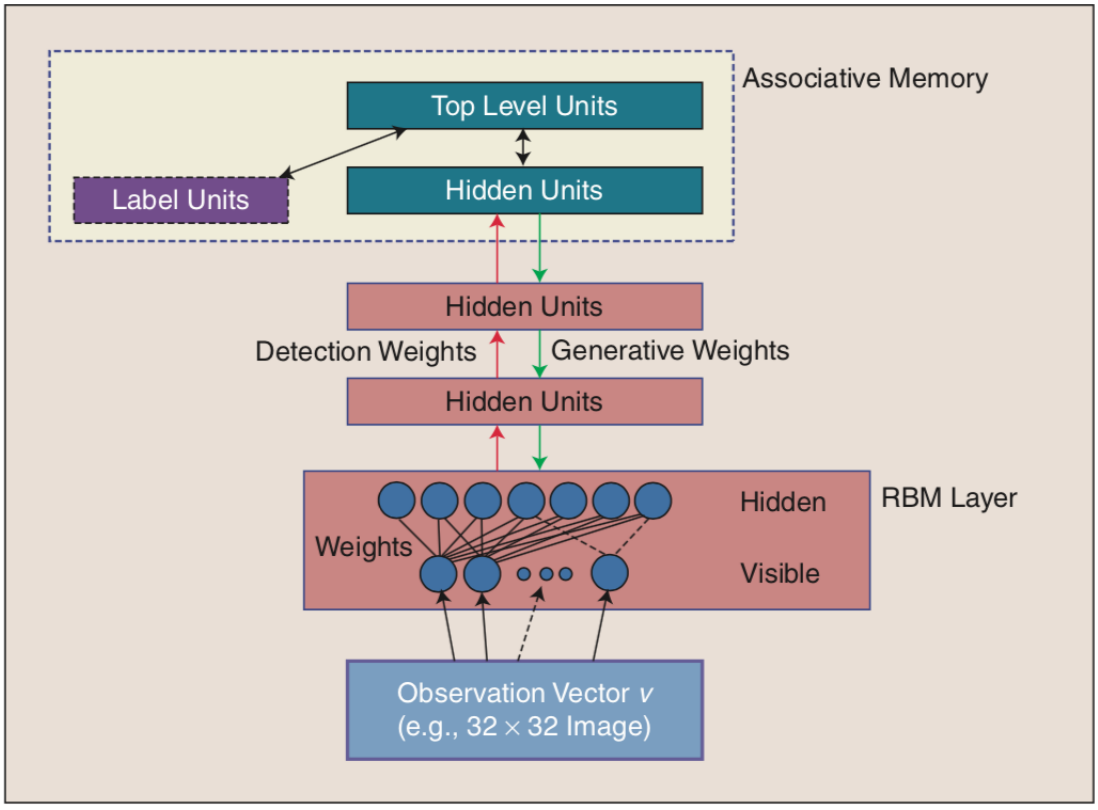


Figure 2.2: Procesamiento Arquitectura DBN
Fuente: (Arel et al., 2010)
Elaborado por: (Arel et al., 2010)

El modelo discriminativo es un procedimiento que trata de aprender los pesos de detección mediante la optimización de una probabilidad posterior. El modelo generativo es un procedimiento que trata de aprender los pesos generativos mediante la optimización de una probabilidad conjunta (Mo, 2012).

DBN tiene la capacidad de “aprender características”, esto se logra mediante una estrategia de aprendizaje “capa por capa” donde las características de nivel superior se han aprendido de las capas anteriores, y esta capacidad es la mayor ventaja de DBN (Lee et al., 2009).

- **Red Neuronal Convolucional**

CNN pertenece a la red de alimentación avanzada (Bengio and LeCun, 2007), pero combina tres ideas: campo receptivo local, pesos compartidos y, en ocasiones, submuestreo espacial o temporal.

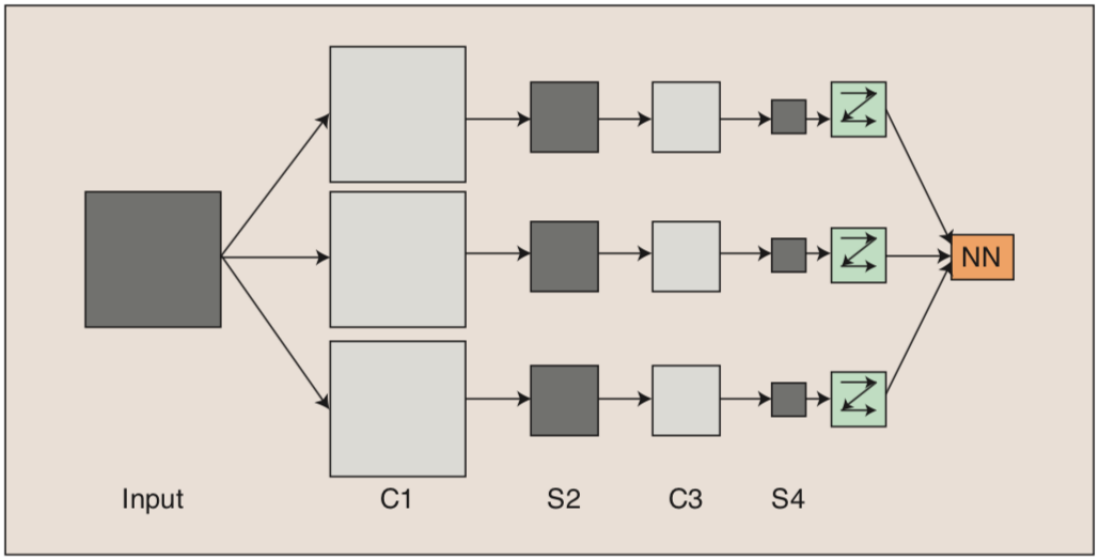


Figure 2.3: Procedimiento Arquitectura CNN
Fuente: (Arel et al., 2010)
Elaborado por: (Arel et al., 2010)

El campo receptivo local que también se denomina ventana de desplazamiento, es una pequeña porción de los datos; el mapeo de la totalidad de la materia se realiza mediante pesos compartidos; el submuestreo espacial y temporal (generalmente el subsumínio espacial) puede reducir los mapas de características por dimensiones, y forma un nuevo conjunto de mapas de características para que el proceso de convolución pueda aplicarse nuevamente (Ngiam et al., 2010).

la Figura 2.3 muestra que las dimensiones de los mapas de características disminuyen después de cada capa, y este proceso convierte a la CNN en un modelo útil para “reducir el número de parámetros que deben aprenderse y que mejora con el entrenamiento general de retro-propagación y de retro-alimentación” (Mo, 2012).

- **Red Neuronal Recurrente**

La Red Neuronal Recurrente (RNN) es un modelo de secuencia neural que logra un desempeño de vanguardia en tareas importantes que incluyen el modelado de lenguaje, reconocimiento de voz y la traducción automática.
Se sabe que las aplicaciones exitosas de Redes Neuronales requieren una buena regularización aunque desafortunadamente, se abandonó el método de regularización más poderoso para las Redes Neuronales de retroalimentación, este no funcionaba bien con las RNN.
Como resultado, las aplicaciones prácticas de RNN a menudo usan modelos que son demasiado pequeños porque los RNN grandes tienden a sobreajustar. Los métodos de regularización existentes proporcionan mejoras relativamente pequeñas para RNN (Graves, 2013).

Las RNN se han utilizado con éxito para el reconocimiento de voz y se han aplicado recientemente a la traducción automática, donde se usan para el modelado de lenguaje, el re-ranking o el modelado de frases (Devlin et al., 2014).

2.2 Algoritmos

En esta sección se observa una breve síntesis sobre el funcionamiento de los algoritmos de predicción y clasificación, como actúan estos en DL, ya en la sección 2.3 se hace un análisis profundo sobre algunos algoritmos de clasificación que es en lo que se enfoca el proyecto a desarrollar.

Los algoritmos de Deep Learning intentan transformar los datos en bruto o desorganizados en una forma a partir de la cual es más fácil realizar tareas de aprendizaje supervisado y no supervisado, como la clasificación y predicción de estos (Dauphin et al., 2012).

2.2.1 Predicción

Cuando a menudo se necesita predecir datos, naturalmente los algoritmos de predicción en DL se basan en variables observables de *Y* dada *X* más a menudo al algoritmo de aprendizaje que los subconjuntos aleatorios de variables observadas (Yoshua Bengio, 2013), es decir se está explotando la existencia de factores explicativos comunes subyacentes que son útiles para varias tareas.
Esto también es cierto para el aprendizaje semi-supervisado, que explota las conexiones entre la distribución de entrada *P (X)* y una distribución condicional objetivo *P (Y | X)*.

En general, estas dos distribuciones, vistas como funciones de *X*, pueden no estar relacionadas entre sí. Pero actualmente, a menudo ocurre que algunos de los factores que dan forma a la distribución de las variables de entrada *X* que logran la predicción o salida de datos en *Y*.

Los algoritmos de predicción tratan en su mayor parte con el aprendizaje profundo que depende en gran medida del aprendizaje no supervisado o semi-supervisado, en la fase de aprendizaje-representación no supervisada, se tiene acceso algunas de las clases.
De tal manera que algunos de los factores que explican *P (X | Y)* para *Y* en las clases de entrenamiento, y que serán capturados por la representación aprendida, serán útiles para predecir diferentes clases con estos algoritmos, a partir del conjunto de pruebas o dataset (Bengio, 2012).

2.2.2 Clasificación

Debido a la demanda de clasificar grandes cantidades de datos para poder ser procesados, surgen los algoritmos de clasificación que utilizan diversos métodos ya sean propios o implementan otros previamente utilizados.

Los algoritmos de clasificación tienden a simplificar de manera extrema este proceso que antes expertos realizaban de manera manual, donde un experto etiquetaba manualmente cada plantilla en la base de datos, luego asignaba a cada variable de entrada la misma etiqueta que se estableció manualmente para la plantilla correspondiente. DL permite el uso de modelos lineales relativamente más simples para las tareas de análisis de datos como la clasificación, que es importante al desarrollar modelos para manejar la escala de grandes cantidades de datos (Najafabadi et al., 2015).

Estos algoritmos en DL buscan la solidez de la clasificación, la capacidad de asignar la misma clase a diferentes variables en común, independientemente de la etiqueta manual.

Para la extracción de características y la clasificación misma, en estos algoritmos; primero, la variable se procesa para extraer características que conducen a una discernibilidad entre las clases que se representan con frecuencia en forma de un vector numérico, en segundo lugar el vector de características se utiliza para realizar la clasificación, ya sea mediante un conjunto de reglas fijas o entrenando un modelo de forma supervisada (Peralta et al., 2018).

En la siguiente sección se analizan los algoritmos de clasificación y su relación con DL más a detalle, debido a que son el objetivo principal del presente Trabajo de Titulación.

2.3 Algoritmos de clasificación

A continuación se explica de manera general todos los tipos de algoritmos de clasificación según Sifium (2017) y cuál es más conveniente de implementar de acuerdo a los propósitos del proyecto a desarrollar: Regresión logística, Redes Bayesianas, Maquina de soporte vectorial, Árboles de decisión, Árboles impulsados, Bosques de decisión aleatoria, Vecino más cercano, **Redes Neuronales**

2.3.1 Clasificador de Redes Bayesianas

Es una técnica de clasificación basada en el teorema de Bayes con un supuesto de independencia entre los predicadores. En términos simples, un clasificador de Redes Bayesianas supone que la presencia de una característica particular en una clase no está relacionada con la presencia de ninguna otra característica.
Incluso si estas características dependen unas de otras o de la existencia de otras características, todas estas propiedades contribuyen de forma independiente a la probabilidad final de clasificación.

El modelo Redes Bayesianas es fácil de construir y particularmente útil para conjuntos de datos simples y muy grandes.

2.3.2 Regresión logística

Es un método estadístico para analizar un conjunto de datos en el que hay una o más variables independientes que determinan un resultado. El resultado se mide con una variable dicotómica (en la que solo hay dos resultados posibles).

El objetivo de la regresión logística es encontrar el modelo más adecuado para describir la relación entre la característica dicotómica de interés (variable dependiente = respuesta o variable de resultado) y un conjunto de variables independientes (predicadoras o explicativas).

2.3.3 Árboles de decisión

El árbol de decisiones construye modelos de clasificación o regresión en forma de una estructura de árbol. Se divide en conjuntos de datos y a su vez en subconjuntos cada vez más pequeños, mientras que al mismo tiempo se desarrolla incrementalmente un árbol de decisiones asociado.

El resultado final es un árbol con nodos de decisión y nodos hoja. Un nodo de decisión tiene dos o más ramas y un nodo hoja representa una clasificación o decisión, el nodo de decisión más elevado en un árbol que corresponde al mejor predicador se llama nodo raíz. Los árboles de decisión pueden manejar datos categóricos y numéricos.

2.3.4 Bosques de decisión aleatoria

Los bosques de decisión aleatoria son un método de aprendizaje conjunto para clasificación, regresión y otras tareas, que operan construyendo una multitud de árboles de decisión en el tiempo de entrenamiento y generando la clase que es el modo de las clases (clasificación) o predicción media (regresión) de los árboles individuales.

Los bosques de decisión aleatoria corrigen el hábito de los árboles de decisión de ajustarse demasiado a su conjunto de entrenamiento.

2.3.5 Vecino más cercano

La técnica de vecino más cercano es un tipo de algoritmos de clasificación supervisados, es decir toma un montón de datos etiquetados y los usa para aprender a etiquetar otros datos.

Para etiquetar un nuevo dato, busca los que se etiquetan más cercanos a ese nuevo datos (esos son sus vecinos más cercanos) y los vecinos votan, por lo que la etiqueta que tenga la mayoría de los vecinos es la etiqueta del nuevo dato (el “k” es el número de vecinos que verifica).

2.3.6 Redes Neuronales

Una Red Neuronal que consta de neuronas(unidades), dispuestas en capas, que convierten un vector de entrada en una salida, en la que cada neurona toma una entrada,

se aplica la función (por lo general no lineal) y pasa la salida a la siguiente capa.
En general, las redes se definen como feed-forward: la salida de una neurona alimenta a todas las neuronas en la siguiente capa, pero no hay retroalimentación a la capa anterior.

Las ponderaciones se aplican a las señales de una neurona a otra, y son estas ponderaciones las que están sintonizadas en la fase de entrenamiento para adaptar la Red Neuronal al problema particular en cuestión.

Las Redes Neuronales como se describe en este Capítulo, son métodos y técnicas propias de la evolución de Deep Learning y como se demuestra en los diferentes tipos de algoritmos de clasificación en el que consta como uno, es de gran utilidad al momento de no requerir una función lineal e ir aprendiendo de cada capa en la fase de entrenamiento donde existe la posibilidad de adaptar la Red Neuronal al problema.
Las Redes Neuronales se usan para modelar todo tipo de problemas, en un contexto de clasificación, la instancia u objeto que debe clasificarse se usa para establecer los valores de la primera capa de la red (capa de entrada). Los valores se propagan a lo largo de la red a través de una o más capas ocultas hasta que la capa final (capa de salida) contiene la clase predicha (Nielsen, 2015).

Una Red Neuronal profunda o arquitectura profunda (DNN) es una red con muchas capas ocultas, cada una de las cuales extrae, en términos generales, un cierto nivel de abstracción del patrón de entrada.
Por lo tanto, un mayor número de capas permite al DNN aprender patrones más complejos y genéricos. Hay diferentes tipos de capas de neuronas para las DNNs (Lecun et al., 1998b):

- **Capas completamente conectadas**

Cada neurona está conectada con pesos a todas las neuronas de la capa anterior.

- **Capas convolucionales**

Cada neurona está conectada a un parche de neuronas en la capa anterior. Los pesos se comparten entre todas las neuronas de la misma capa, reduciendo el espacio de búsqueda del proceso de aprendizaje.

- **Agrupación de capas**

Por lo general, se ubican después de una capa convolucional. Como en estos, cada neurona está conectada a un parche de la capa anterior y calcula el máximo o el promedio de esos valores.

En la práctica, las redes que combinan los tres tipos de capas se llaman Redes Neuronales Convolucionales. Están bien adaptados al procesamiento de imágenes y estructuras con alguna relación espacial, como lo demuestran los buenos resultados obtenidos en diferentes competiciones (Nair and Hinton, 2010).

Cuando se usa una red como un clasificador para un problema con clases *c1, ..., cm,* la capa de salida contiene una neurona por cada clase, formando un vector **a** = *(a1, ..., am)*. La función de activación utilizada para modelar la no linealidad suele ser la Unidad Lineal Rectificada (ReLU), que puede calcularse más rápido que las funciones tangentes sigmoideas o hiperbólicas utilizadas tradicionalmente y también ofrece interesantes propiedades de convergencia (Goodfellow et al., 2016).

El entrenamiento de una red consiste en optimizar los pesos de cada neurona para obtener la salida deseada para cada entrada. Por lo tanto, la dimensionalidad del espacio de búsqueda es tan alta como el número total de ponderaciones como se demuestra en el algoritmo de referencia para el entrenamiento de BP con Pendiente de Gradiente (GD) (Williams and Hinton, 1986).

Sin embargo, GD se vuelve computacionalmente costoso cuando se aplica a un DNN, debido a la alta dimensionalidad del espacio de búsqueda. El descenso de gradiente estocástico (SGD) puede reducir esta limitación utilizando solo un subconjunto (o lote) de las instancias de entrenamiento en cada iteración, de modo que el cálculo del error

esté sesgado con respecto al óptimo pero puede realizarse mucho más rápido. Cada iteración en todo el conjunto de entrenamiento (época) requiere múltiples iteraciones sobre los lotes pequeños.
Este algoritmo, junto con los avances recientes en unidades de procesamiento gráfico (GPU) y la disponibilidad de grandes conjuntos de datos, ha permitido la implementación y capacitación de DNN con muy buenos resultados (Nair and Hinton, 2010) al momento de ejecutar algoritmos de clasificación.

Recientemente, los métodos de DL han comenzado a aplicarse a la clasificación de textos como el método de CNN basado en palabras. Entre la gran cantidad de trabajos recientes sobre CNN basados en palabras, para la clasificación de texto, una de las diferencias es la elección del uso de representaciones de palabras preentrenadas con el uso de la inclusión preestablecida de word2vec[1] (McAuley and Leskovec, 2013) y el uso de tablas de búsqueda (Collobert et al., 2011). El tamaño de inserción es de 300 palabras en ambos casos.

Los modelos para cada caso deben ser iguales, en términos tanto del número de capas como del tamaño de salida de cada capa, aunque también se realizan experimentos que utilizan un diccionario de sinónimos para el aumento de datos. Word2vec y Natural Language Processing (NLP) son modelos específicos para su uso en palabras o datos textuales que trabajan con la librería TensorFlow explicada en la siguiente sección, siendo la librería a utilizar en el proyecto a desarrollar

2.4 Librerías para Deep Learning

Actualmente existen varias bibliotecas de software que pueden ser utilizadas para simplificar la implementación de algoritmos DL, que proporcionan herramientas capaces de ejecutar automáticamente algunas de las tareas necesarias para configurar una Red Neuronal funcional.

[1] `https://www.TensorFlow.org/tutorials/word2vec`

Estas bibliotecas difieren en su implementación. En esta sección se muestran varias de las librerías más comunes utilizadas para DL, la mayoría de estas bibliotecas del lenguaje Python[1].

2.4.1 TensorFlow

Originalmente desarrollado por Google como parte del proyecto Google Brain, TensorFlow[2] se convirtió en código abierto a finales de 2015. Es una biblioteca multiplataforma y su funcionamiento es sencillo.

la Figura 2.4 ilustra su arquitectura general en la cuál por una C API separa el código de nivel de usuario en diferentes idiomas del tiempo de ejecución del núcleo, las librerías de entrenamiento y clientes (Python en el proyecto a desarrollar) con el núcleo donde define el cálculo como un gráfico de flujo de datos.

Primero inicia la ejecución del gráfico usando una sesión como Maestro Distribuido, posteriormente inicia la ejecución de la pieza del gráfico por los servicios del trabajador (uno para cada tarea) y programa la ejecución de operaciones gráficas utilizando implementaciones del kernel apropiadas para el hardware disponible (CPU, GPU, etc.). Envía y recibe resultados de operación hacia y desde otros servicios del trabajador, finalmente realiza el cálculo para operaciones gráficas individuales.

la Figura 2.5 ilustra la interacción de estos componentes. "/ job: worker / task: 0" y "/ job: ps / task: 0" son tareas con servicios de trabajador. "PS" significa "servidor de parámetros": una tarea responsable de almacenar y actualizar los parámetros del modelo.

Otras tareas envían actualizaciones a estos parámetros mientras trabajan en la optimización de los parámetros. Esta división de trabajo particular entre tareas no es

[1] `https://www.python.org`
[2] `https://www.TensorFlow.org`

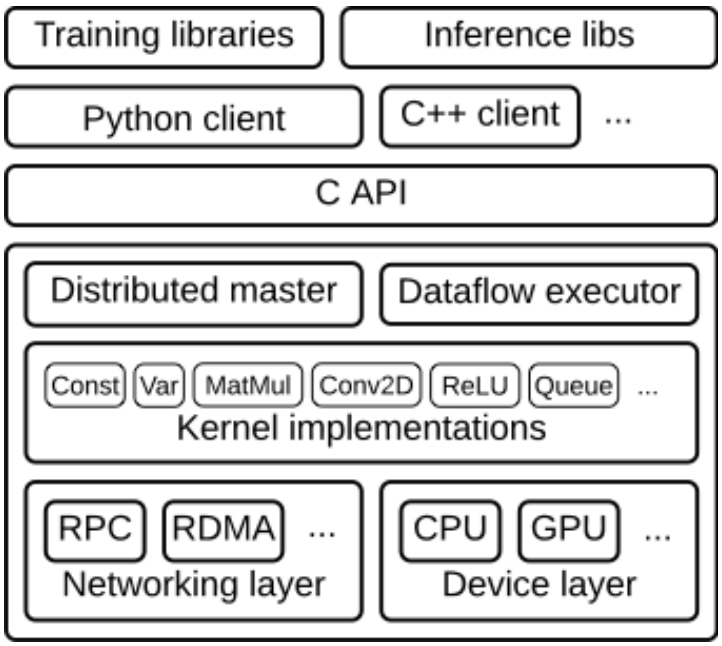


Figure 2.4: Arquitectura de la Librería TensorFlow
Fuente: TensorFlow (TF)
Elaborado por: TensorFlow (TF)

necesaria, pero es común para la capacitación distribuida.

TensorFlow se distingue por ser la única de las principales bibliotecas desarrolladas desde cero por una gran corporación, mientras que otros tienen su origen en la comunidad de investigación.

TensorFlow tiene algunas herramientas integradas de calidad de vida como TensorBoard[1], que le permite al usuario producir fácilmente gráficos que visualizan cosas tales como la velocidad de aprendizaje, el peso del modelo, las funciones de pérdida y más.
Es también la única biblioteca que puede distribuir la carga de trabajo no solo a través de GPU en el mismo dispositivo sino también en varios dispositivos conectados, lo que puede ser una gran ventaja de cómputo.

[1] `https://github.com/TensorFlow/tensorboard`

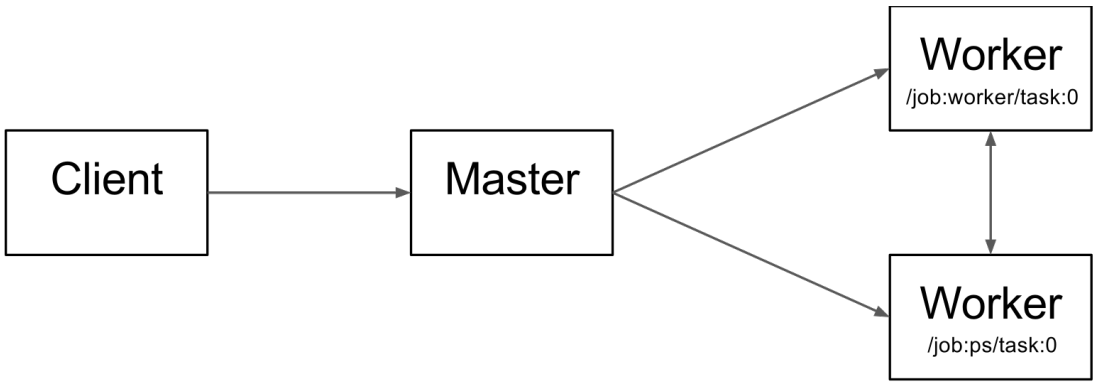


Figure 2.5: Diagrama de Arquitectura de la Librería TensorFlow
Fuente: TensorFlow (TF)
Elaborado por: TensorFlow (TF)

Para el presente Trabajo de Titulación se ha definido trabajar con la librería TensorFlow. En el Anexo 1 se detalla la instalación del entorno.

2.4.2 Keras

Al igual que Lasagne, Keras[1] es una envoltura de alto nivel que se ejecuta en la parte superior de Theano. Además, también se puede ejecutar en la parte superior de TensorFlow.

A diferencia de Lasagne, que siempre usa algún código de Theano, Keras no mostrará el trabajo subyacente. Está diseñado para minimizar los gastos generales, para permitir el prototipado rápido y fácil de los algoritmos de aprendizaje automático.

2.4.3 Theano

Theano[2] es una biblioteca de Python para expresiones matemáticas en Python, desarrollada con el objetivo de facilitar la investigación en aprendizaje profundo (Bergstra et al., 2011).

[1] `https://keras.io`
[2] `http://www.deeplearning.net/software/theano/`

Con Theano puede definir, optimizar y evaluar expresiones matemáticas de manera eficiente, utilizando matrices multidimensionales. Su sintaxis es similar a NumPy[1] y esto combinado con un código de máquina nativo optimizado hace de Theano una herramienta poderosa, especialmente al implementar algoritmos de aprendizaje automático.

Theano optimiza la elección de expresiones antes del cálculo y luego puede traducirlo a C ++ o CUDA[2], dependiendo de si el programa se ejecutará en la CPU o GPU. Bergstra et al. demostró en 2010 que la implementación de algoritmos comunes de aprendizaje automático con Theano es entre 1,6 y 7,5 veces más rápido que las alternativas competitivas cuando se compila para funcionar en una CPU y desde 6,5 hasta 44 veces más rápido cuando se compila para la GPU (Bergstra et al., 2010).

2.4.4 Lasagne

Lasagne[3] es una biblioteca de Python destinada a simplificar el proceso de creación y entrenamiento de algoritmos de aprendizaje automático con Theano. Sin embargo, lasaña se importa junto con Theano (y NumPy) y no se pretende como un sustituto, pero algún código simple de Theano se usará generalmente tan bien como lasagne, que es principalmente una herramienta útil.

2.4.5 Caffe

Caffe[4] fue creado por Yangqing Jia durante su doctorado en la Universidad de California, Berkeley y, aunque es una biblioteca C ++, utiliza Python como su API.

Caffe es excelente para implementar redes de feed-forward y en realidad es posible hacerlo sin escribir ningún código. Esto permite una prueba y ajuste rápidos cuando se usan redes existentes (Li et al., 2017).

[1] `http://www.numpy.org`
[2] `https://opencv.org/platforms/cuda.html`
[3] `http://lasagne.readthedocs.io`
[4] `http://caffe.berkeleyvision.org`

2.4.6 Torch

Torch[1] se originó en la Universidad de Nueva York y está escrita en C y Lua. Es utilizado, entre otros, por Facebook, Twitter y el proyecto DeepMind para implementar Redes Neuronales. Como Lua es una interfaz de alto nivel para C, optimiza el código y permite que cosas como for-loops se ejecuten mucho más rápido en comparación con Python (Li et al., 2017).

2.5 Trabajos relacionados

En los algoritmos de clasificación en el campo de DL existe una tendencia común en el uso de las arquitecturas profundas más utilizadas como la Red Neuronal Convolucional (CNN) para diferentes campos como computer visión (visión por computador), es decir, clasificación de datos como rostros faciales, huellas digitales, videos, entre otros.

En la clasificación de rostros (Yuan et al., 2017) lo hace con base en la Máquina de Vectores de Soporte (SVM), donde la capacidad de generalización y la alta capacidad de clasificación se obtienen al encontrar el hiperplano de clasificación óptimo. Sin embargo, sobre la base de la detección de rostros, se propone una estructura o arquitectura profunda sobre una Red Neuronal Convolucional (CNN) basada en TensorFlow, un marco de aprendizaje abierto de aprendizaje profundo para el reconocimiento facial logrando en los resultados experimentales muestran que el método propuesto tiene una mayor precisión de reconocimiento y una mayor robustez en entornos complejos.

A diferencia de este autor, (Esteva et al., 2017) entrena una CNN usando un conjunto de datos de 129,450 imágenes clínicas, en comparación a los 778 rostros utilizados en anterior trabajo, aquí se utiliza el marco de aprendizaje profundo de la arquitectura CNN y la librería de Google TensorFlow para el entrenamiento, previo a la imple-

[1] `http://torch.ch`

mentación de algoritmos de clasificación para las imágenes clínicas sobre las que se pretende clasificar el tipo de cáncer de piel.

Adicionalmente de la misma manera lo hace (Yao et al., 2017) al lograr clasificar modelos urbanos utilizando un modelo de aprendizaje profundo basado en Redes Neuronales Convolucionales (CNN) preentrenado sobre un dataset de ImageNet, utilizando la librería TensorFlow.

Los algoritmos de clasificación junto a DL han colaborado para solucionar problemas diferentes a los de la visión por computador, como explica (Kolosnjaji et al., 2016), en este trabajo se realiza la clasificación de secuencias de llamadas al sistema de malware (programas maliciosos), donde se usa las librerías TensorFlow y Theano para construir y entrenar las Redes Neuronales, ejecutando los algoritmos de entrenamiento y finalmente los de clasificación.

Según (Yoshua Bengio, 2013) se prueban dos tipos de Redes Neuronales para la tarea de detectar el énfasis en la voz, una Red Neuronal con capas totalmente conectadas (FCNN) y una Red Neuronal Convolucional (CNN), entrenando estas redes con la librería TensorFlow que se utilizó para realizar los experimentos en un GPU.
Aquí podemos ver como las Redes Neuronales pueden trabajar juntas con la librería a utilizar en el proyecto a desarrollar.

Aunque los métodos de aprendizaje profundo y neuronal están ahora bien establecidos en el aprendizaje automático y han sido especialmente exitosos para tareas de procesamiento de imágenes y voz, se han propuesto algunos métodos para la clasificación de texto.

Recientemente, las incrustaciones de palabras se han explotado para la clasificación de oraciones utilizando arquitecturas CNN. (Kalchbrenner et al., 2014) propuso una arquitectura CNN con múltiples capas de convolución, postulando vectores de palabras latentes, densas y de baja dimensión inicializados en valores aleatorios como

entradas.

En (Kim, 2014) se definió una arquitectura de CNN de una capa que funcionaba de manera similar. Este modelo utiliza vectores de palabras preentreados como entradas, que pueden tratarse como estáticos o no estáticos. En el trabajo previo, los vectores de palabras se tratan como entradas fijas, mientras que en este último se "ajustan" para una tarea específica.

Por otra parte, (Johnson and Zhang, 2014) propusieron un modelo similar, pero intercambiaron representaciones de palabras de alta resolución "one-shot" como entradas de CNN. Su atención se centró en la clasificación de textos más largos, en lugar de oraciones aunque el modelo se puede utilizar para la clasificación de oraciones.

Actualmente, hay muchos datos empíricos y no etiquetados disponibles sin ser tratados, abordar este problema es el objetivo del proyecto a desarrollar.

2.6 Discusión

Como se observa en la Tabla 2.1, para trabajos de clasificación para los diferentes tipos de datos en el ámbito de Deep Learning, la arquitectura profunda CNN es la más utilizada en muchos casos junto a la librería TensorFlow.
Por lo tanto, para el desarrollo del presente Trabajo de Titulación se decide realizar la aplicación de CNN con la librería TensorFlow para clasificación de conjunto de datos de texto, siendo una nueva contribución en el campo de Deep Learning.

En el siguiente Capítulo se define la metodología para implementar los diferentes algoritmos a realizar pruebas junto a librería TensorFlow y a las arquitecturas profundas de DL, en el Capítulo 4 se realiza en análisis de resultados de las clasificaciones resultantes.

Table 2.1: Resumen de trabajos relacionados

Trabajo	**Red Neuronal o Arquitectura Profunda**	**Datos trabajados**	**Librerías Implementadas**	**Área de aplicación**	**Organización y/o grupo de investigación**
A convolutional neural network based on TensorFlow forface recognition.	SVM y CNN	Imágenes	TensorFlow	DL	(Yuan et al., 2017)
Dermatologist-levelclassification of skin cancer with deep neural networks.	CNN	Imágenes	TensorFlow	Medicina	(Esteva et al., 2017)
Deep Learning for Classification of Malware System Call Sequences.	CNN	Imágenes	TensorFlow y Theano	Inteligencia Artificial	(Kolosnjaji et al., 2016)
Statistical Language and Speech Processing.	FCNN y CNN	Audio	TensorFlow	Deep Learning	(Yoshua Bengio, 2013)
A convolutional neural network for modelling sentences.	CNN	Texto	-	Deep Learning	(Kalchbrenner et al., 2014)
Convolutional neural networks for sentence classification.	CNN	Texto	-	Deep Learning	(Kim, 2014)
Effective use of word order for text categorization with convolutional neural networks.	CNN	Texto	-	Deep Learning	(Johnson and Zhang, 2014)

Fuente: El Autor

Elaborado por: El Autor

CAPÍTULO 3

PROCESO DE CLASIFICACIÓN

En este capítulo se presenta el proceso de clasificación que se da en este Trabajo de Titulación. En la sección 3.1 se presenta la metodología que se utiliza, una modificación del proceso KDD acoplado a Deep Learning. En la sección 3.2 se define el problema a resolver. La sección 3.4 se enfoca al preprocesado y depuración de los datos. En la sección 3.5 se elije las variables para la clasificación en base al problema y en la sección 3.6 se realiza la selección del algoritmo de clasificación. La sección 3.7 demuestra la implementación de la CNN y finalmente en la sección 3.8 se presenta la clasificación y validación de los datos.

3.1 Metodología

La metodología propuesta está basada en la metodología KDD y se usa para el descubrimiento de conocimiento en grandes colecciones de datos.

El proceso KDD es iterativo por naturaleza, y depende de la interacción para la toma de decisiones, de manera dinámica. La metodología que se ilustra en la Figura 3.1 es un proceso metodológico y además secuencial que se sigue para trabajar sobre un conjunto de datos en bruto a través de la clasificación con técnicas de Deep Learning.

3.2 Definición del Problema

Para la implementación de los algoritmos de clasificación, objetivo del presente Trabajo de Titulación, se realiza la prueba y análisis en base al tipo de algoritmo de Red Neuronal junto a la librería TensorFlow y la arquitecturas profundas como la CNN y arquitecturas básicas como la RNN.

Con la implementación del algoritmo de clasificación, se logra resolver varias incógnitas como la clasificación de interacciones del usuario a la plataforma LMS y a través de este parámetro se da solución al problema:

- Qué tipo de evento o problemas revisa el o los usuarios (estudiantes) dentro de

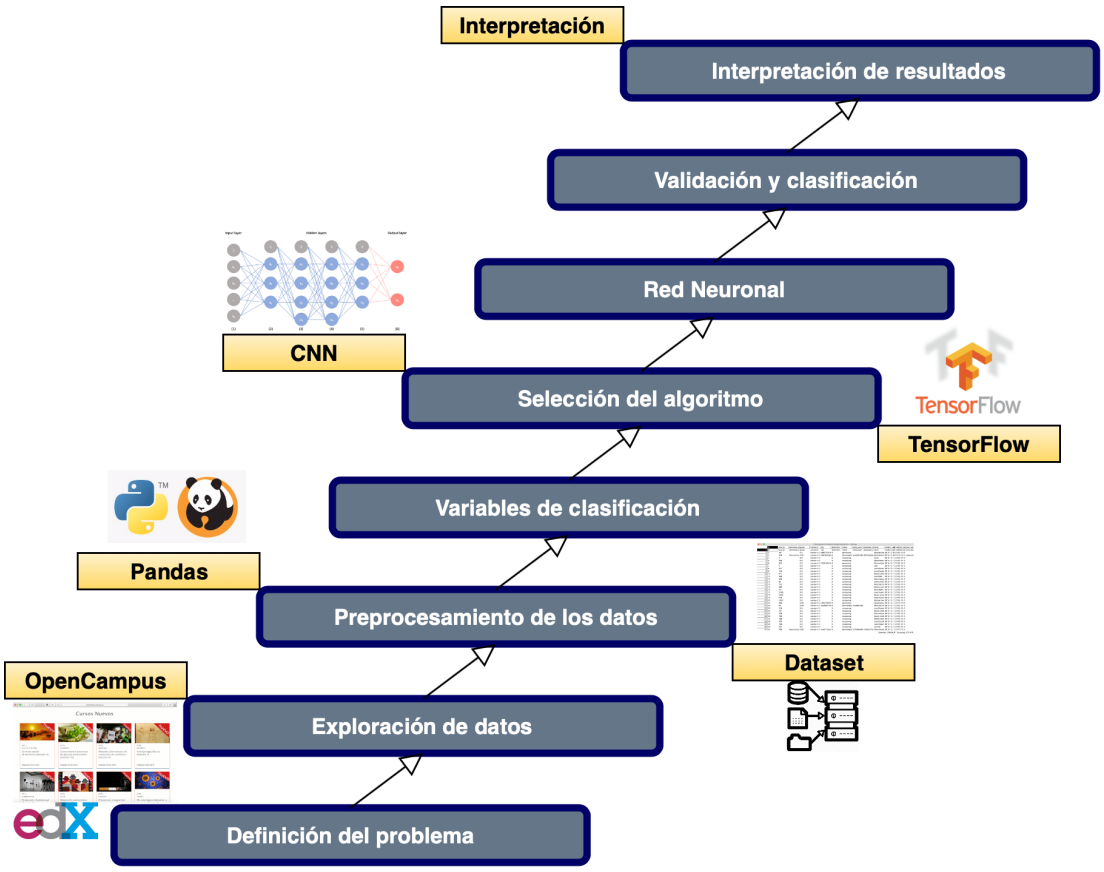


Figure 3.1: Metodología propuesta
Fuente: (Fernández)
Elaborado por: El Autor

la plataforma LMS constantemente?

- Cuál es la eficiencia de poder clasificar de manera inteligente los cursos o la acciones que se ejecuta en estos?
- En qué estado se encuentra el usuario (aprobado / reprobado)?

3.3 Exploración de datos

Para poder explorar los datos o logs junto a los atributos que son contemplados en la clasificación, y son analizados por el algoritmo de clasificación en esta sección se

describe los campos JSON que son comunes a las definiciones de esquema de todos los eventos. Estos campos están en el nivel raíz de los documentos JSON del evento.

Un evento a diferencia de otro puede incluir estos campos en diferentes secuencias, a continuación se presenta un evento tomado del dataset de la plataforma OpenEdx con los campos comunes en orden alfabético.

```
{
  "username": "josguarnizo",
  "event_type": "/courses/course-v1:UTPL+EFHE+2016/xblock/
   block-v1:UTPL+EFHE+2016+type@video+block@84e98c49a6184798
   b41a07302ca7cae8/handler/transcript/translation/en",
  "ip": "172.18.4.169",
  "agent": "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit
   /537.36 (KHTML, like Gecko) Chrome/54.0.2840.71 Safari/53
   7.36",
  "host": "opencampus.utpl.edu.ec",
  "referer": "http://opencampus.utpl.edu.ec/courses/course-v
   1:UTPL+EFHE+2016/courseware/98f91ff6ec8d42768b1eb404c27
   eae5a/e995bae5cd114ad7942c8cf39556a5e4/",
  "accept_language": "es-ES,es;q=0.8",
  "event": "{\"POST\": {}, \"GET\": {\"videoId\": [\"W3
   QqknVtHGw\"]}}",
  "event_source": "server",
  "context": {
    "course_user_tags": {},
    "user_id": 3139,
    "org_id": "UTPL",
```

```
    "course_id": "course-v1:UTPL+EFHE+2016",
    "path": "/courses/course-v1:UTPL+EFHE+2016/xblock/block-
    v1:UTPL+EFHE+2016+type@video+block@84e98c49a6184798b41a07
    302ca7cae8/handler/transcript/translation/en"
  },
  "time": "2016-11-14T15:32:45.406497+00:00",
  "page": null
}
```


Los campos con los que cuenta el dataset se muestran a continuación, con su descripción y tipo (Edx, 2015):

- **accept_language**

Tipo: cadena
Descripción: el valor del campo HTTP-Accept-Language request-header.

- **agent**

Tipo: cadena
Descripción: secuencia del agente del navegador del usuario que activó el evento.

- **context**

Tipo: objeto
Descripción:El campo de contexto incluye campos de miembros que proporcionan información contextual.

Este campo contiene un conjunto básico de campos de miembros que son comunes a todos los eventos. Para ciertos eventos con requisitos contextuales adicionales, este campo contiene un conjunto de campos de miembros adicionales que son comunes a esos eventos solamente. Para cualquier evento, este campo también puede incluir uno o más campos de miembros adicionales.

- **context (campo común en todos los eventos)**

Los siguientes campos están presentes en el campo de contexto para todos los eventos.

course_id

Tipo: cadena

Descripción: Identifica el curso que generó el evento.

org_id

Tipo: cadena

Descripción: La organización que enumera el curso.

route

Tipo: cadena

Descripción: La URL que generó el evento.

user_id

Tipo: numérico

Descripción: Identifica a la persona que realiza la acción.

- **context (campo para eventos aplicables)**

Cuando corresponda para un evento, el campo de contexto también incluye estos campos para proporcionar información adicional.

course_user_tags

Tipo: objeto

Descripción: Contiene la (s) clave (s) y los valores de la tabla user_api_usercoursetag para el usuario.

module

Tipo: objeto

Descripción: Proporciona información de identificación para los componentes involucrados en un evento de servidor.

Los campos de miembros de contexto están en blanco si no se pueden determinar los valores.

- **event**

Tipo: objeto
Descripción: este campo incluye campos de miembros que identifican detalles de cada evento desencadenado. Se suministran diferentes campos de miembros para diferentes eventos.

- **event_source**

Tipo: cadena
Descripción: especifica la fuente de la interacción que activó el evento. Los valores en este campo son:
'navegador'
'móvil'
'servidor'
'tarea'

- **event_type**

Tipo: cadena
Descripción: el tipo de evento desencadenado. Los valores dependen de event_source.

- **host**

Tipo: cadena
Descripción: el sitio visitado por el usuario, por ejemplo, courses.edx.org.

- **ip**

Tipo: cadena
Descripción: dirección IP del usuario que activó el evento. Vacío para eventos que se originan en dispositivos móviles.

- **name**

Tipo: cadena
Descripción: identifica el tipo de evento desencadenado.

- **page**

Tipo: cadena
Descripción: el 'URL' de la página que el usuario estaba visitando cuando se emitió el evento.
Para los eventos de video que se originan en dispositivos móviles, identifica la URL para el componente de video.

- **referer**

Tipo: cadena
Descripción: el URI del campo del encabezado de solicitud de HTTP Referer.

- **session**

Tipo: cadena
Descripción: este valor de 32 caracteres es una clave que identifica la sesión del usuario. Todos los eventos del navegador y los eventos de inscripción del servidor incluyen un valor para la sesión. Otros eventos de servidor y eventos móviles no incluyen un valor de sesión.

- **time**

Tipo: cadena
Descripción: da la hora UTC en la que se emitió el evento en el formato 'AAAA-MM-DDThh: mm: ss.xxxxxx'.

- **username**

Tipo: cadena
Descripción: el nombre de usuario del usuario que provocó la emisión del evento.

3.4 Preprocesamiento de los datos

El entrenamiento de los algoritmos de clasificación de texto necesitan que las oraciones o lineas de los datasets tengan una estructura similar, se inicia la preparación de los logs eliminando algunas inconsistencias en los datos y se preparan los logs de la plataforma OpenEdx para su primer entrenamiento.

3.4.1 Depuración del dataset de Open Campus

El dataset utilizado para el proyecto a desarrollar cuenta con seis millones de lineas de texto a clasificar obtenidos de la plataforma OpenEdx de la iniciativa OpenCampus como se demuestra en la Figura 3.2.

Figure 3.2: Dataset de Logs
Fuente: El Autor
Elaborado por: El Autor

Primero se depura las cadenas de texto que están demás como la dirección de tracking que se observa en la Figura 3.2 y como resultado las lineas de logs se encuentran en elementos JSON embebidos, cada uno con los atributos explicados en la sección anterior con los datos del docente o estudiante que ha ejecutado un evento ya sea en el cliente o servidor de la plataforma OpenEdx.

Como segundo paso para la depuración se procede a realizar la limpieza manual del dataset original, eliminando campos inconsistentes, obteniendo ya un formato adecuado y similar para cada elemento como se muestra en la Figura 3.3.

{"username": "BelenCobo", "event_type": "/courses/course-v1:UTPL+EFHE-Ed8+2018_1/about", "ip": "186.101.172.228", "agent": "Mozilla/5.0 (Macinto
{"username": "BelenCobo", "event_type": "/i18n.js", "ip": "186.101.172.228", "agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3) AppleWebK
{"username": "jdatiaja", "event_type": "/courses/course-v1:UTPL+EFHE-Ed8+2018_1/about", "ip": "170.244.211.17", "agent": "Mozilla/5.0 (Windows N
{"username": "jdatiaja", "event_type": "/i18n.js", "ip": "170.244.211.17", "agent": "Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, lik
{"username": "jdatiaja", "event_type": "/change_enrollment", "ip": "170.244.211.17", "agent": "Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (
{"username": "jdatiaja", "event_source": "server", "name": "edx.course.enrollment.activated", "accept_language": "es-ES,es;q=0.9", "time": "2018
{"username": "jdatiaja", "event_type": "/dashboard", "ip": "170.244.211.17", "agent": "Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, l
{"username": "Fabry", "event_type": "/courses/course-v1:UTPL+EFHE-Ed8+2018_1/about", "ip": "181.199.39.71", "agent": "Mozilla/5.0 (Linux; Androi
{"username": "Fabry", "event_type": "/i18n.js", "ip": "181.199.39.71", "agent": "Mozilla/5.0 (Linux; Android 5.1.1; SM-T285M Build/LMY47V) Apple
{"username": "", "event_type": "/courses/course-v1:UTPL+EFHE-Ed8+2018_1/about", "ip": "181.196.54.38", "agent": "Mozilla/5.0 (Windows NT 10.0; W
{"username": "", "event_type": "/i18n.js", "ip": "181.196.54.38", "agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
{"username": "GabyR", "event_type": "/courses/course-v1:UTPL+EFHE-Ed2+2017_ENE/about", "ip": "186.46.73.243", "agent": "Mozilla/5.0 (Windows NT
{"username": "GabyR", "event_type": "/i18n.js", "ip": "186.46.73.243", "agent": "Mozilla/5.0 (Windows NT 6.1; rv:51.0) Gecko/20100101 Firefox/51
{"username": "MarenaFranco", "event_type": "/courses/course-v1:UTPL+EFHE-Ed8+2018_1/about", "ip": "181.198.17.35", "agent": "Mozilla/5.0 (Window
{"username": "MarenaFranco", "event_type": "/i18n.js", "ip": "181.198.17.35", "agent": "Mozilla/5.0 (Windows NT 5.1) AppleWebKit/537.36 (KHTML,
{"username": "MarenaFranco", "event_type": "/change_enrollment", "ip": "181.198.17.35", "agent": "Mozilla/5.0 (Windows NT 5.1) AppleWebKit/537.3
{"username": "MarenaFranco", "event_source": "server", "name": "edx.course.enrollment.activated", "accept_language": "es-ES,es;q=0.8,en;q=0.6",
{"username": "MarenaFranco", "event_type": "/dashboard", "ip": "181.198.17.35", "agent": "Mozilla/5.0 (Windows NT 5.1) AppleWebKit/537.36 (KHTML
{"username": "ElizabethCadme", "event_source": "server", "name": "edx.course.enrollment.activated", "accept_language": "es-ES,es;q=0.8", "time":
{"username": "ElizabethCadme", "event_type": "/course/course-v1:UTPL+EFHE+2016", "ip": "172.17.26.153", "agent": "Mozilla/5.0 (Macintosh; Intel
{"username": "ElizabethCadme", "event_type": "/i18n.js", "ip": "172.17.26.153", "agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleW
{"username": "ElizabethCadme", "event_type": "/", "ip": "172.17.26.153", "agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/5
{"username": "ElizabethCadme", "event_type": "/home/", "ip": "172.17.26.153", "agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWeb
{"username": "ElizabethCadme", "event_type": "/course/course-v1:UTPL+EFHE+2016", "ip": "172.17.26.153", "agent": "Mozilla/5.0 (Macintosh; Intel
{"username": "ElizabethCadme", "event_type": "/i18n.js", "ip": "172.17.26.153", "agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleW
{"username": "ElizabethCadme", "event_type": "/course_team/course-v1:UTPL+EFHE+2016", "ip": "172.17.26.153", "agent": "Mozilla/5.0 (Macintosh; I
{"username": "ElizabethCadme", "event_type": "/i18n.js", "ip": "172.17.26.153", "agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleW
{"username": "ElizabethCadme", "event_type": "/course_team/course-v1:UTPL+EFHE+2016/viejose@hotmail.es", "ip": "172.17.26.153", "agent": "Mozill
{"username": "ElizabethCadme", "event_source": "server", "name": "edx.course.enrollment.activated", "accept_language": "es-ES,es;q=0.8", "time":

Figure 3.3: Dataset Open Campus Depuración Inicial
Fuente: El Autor
Elaborado por: El Autor

A través de la librería Pandas del lenguaje de programación Python con el método DataFrame, se organiza de una manera visual más legible los datos para así poder determinar las variables o parametros a tomar en cuenta para el entrenamiento de los algoritmos.

La representación de los datos en una tabla organizada gracias al método DataFrame de la librería Pandas se muestra en la Figura 3.4.

3.5 Variables para la clasificación

Tomando en cuenta la complejidad del dataset de OpenEdx y para cumplir con el objetivo propuesto del Trabajo de Titulación, se puede realizar varias clasificaciones con campos de los logs descritos en este Capítulo.

In [1]:
```
import pandas as pd
df = pd.read_json('outputa.json', orient='index')
df.head(14)
```

Out[1]:

	0	1	2
accept_language	en-ca	en-ca	es-ES,es;q=0.9
agent	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_3...	Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.3...
context	{u'course_id': u'course-v1:UTPL+EFHE-Ed8+2018_...	{u'course_id': u'', u'org_id': u'', u'user_id'...	{u'course_id': u'course-v1:UTPL+EFHE-Ed8+2018_...
event	{"POST": {}, "GET": {}}	{"POST": {}, "GET": {}}	{"POST": {}, "GET": {}}
event_source	server	server	server
event_type	/courses/course-v1:UTPL+EFHE-Ed8+2018_1/about	/i18n.js	/courses/course-v1:UTPL+EFHE-Ed8+2018_1/about
host	opencampus.utpl.edu.ec	opencampus.utpl.edu.ec	opencampus.utpl.edu.ec
ip	186.101.172.228	186.101.172.228	170.244.211.17
name	NaN	NaN	NaN
page	None	None	None
referer	http://opencampus.utpl.edu.ec/courses	http://opencampus.utpl.edu.ec/courses/course-v...	http://opencampus.utpl.edu.ec/courses
session	NaN	NaN	NaN
time	2018-03-28T21:22:51.317132+00:00	2018-03-28T21:22:52.159125+00:00	2018-03-28T21:30:05.122656+00:00
username	BelenCobo	BelenCobo	jdatiaja

Figure 3.4: Demostración en tabla del Dataset
Fuente: El Autor
Elaborado por: El Autor

Sin embargo para realizar la implementación de los algoritmos de clasificación y estos a su vez como requerimiento al ser semi-supervisados necesitan de una descripción que es donde los campos de los logs entrarían, una etiqueta o categoría que le explica al algoritmo y la librería TensorFlow qué significa cada campo para poder ser entrenado por el método de Deep Learning de Redes Neuronales.

Además se propone la prueba de dos algoritmos de clasificación para ver cuál obtiene mejor exactitud y menos pérdida en la menor cantidad de tiempo en el siguiente Capítulo.

Se debe tener un solo grupo de datos etiquetados para el entrenamiento de cada uno de los algoritmos y para el test se debe tener el grupo entero, por lo que resulta

más conveniente clasificar los datos por su tipo de evento y ver cuantas interacciones realiza el usuario dentro de la plataforma (User id, Username, Name, Event, Event Source y Event type) de tal manera se puede clasificar e identificar las interacciones en todo el archivo de logs.

Se utiliza estas variables para la clasificación permitiendo que en cualquier momento con diferentes lineas de logs con el mismo algoritmo de clasificación en cualquier fase del curso dentro la plataforma OpenEdx determinar si los usuarios se encuentran en estado de aprobados o reprobados en los cursos que siguen. Estas variables también permiten facilitar la clasificación y simplificar el tiempo tomado por la ejecución del algoritmo.

3.6 Selección del algoritmo de clasificación

Para la selección del algoritmo más adecuado se toma en cuenta el resultado de las pruebas de los algoritmos de clasificación para texto basados en DL con el uso de la librería TF. Las arquitecturas profundas de las Redes Neuronales utilizadas en el Trabajo de Titulación, la Red Neuronal Recurrente y la Red Neuronal Convolucional, con un total de seis millones de lineas de logs para el entrenamiento de esta manera los algoritmos pueden realizar la clasificación sobre el millón de lineas de logs en los que se realiza la evaluación del algoritmo. El proceso de entrenamiento y los parámetros de la Red Neuronal implica la ejecución del algoritmo explicado a detalle en la sección 3.7 sobre los seis millones de logs.

En las pruebas de entrenamiento a realizar se mide tanto la exactitud como la pérdida que son las dos métricas más importantes durante el entrenamiento de los datos en la ejecución de los algoritmos.

El accurancy en español la exactitud del entrenamiento, se observa en la Figura 3.5 donde existe demasiado esparcimiento desde las 100 primeras iteraciones hasta la iteración 450 que es donde pasa el valor de 0.997 lo que significa que la clasificación

podría ser estable en este punto sin embargo el algoritmo necesita una óptima exactitud

Desde la iteración 700 deja los datos de esparcimiento y pasa de un valor de 0.998 a 1 constante sin embargo continua hasta 1200 iteraciones por las pérdidas en la curva de aprendizaje del algoritmo ocurridas durante el entrenamiento.

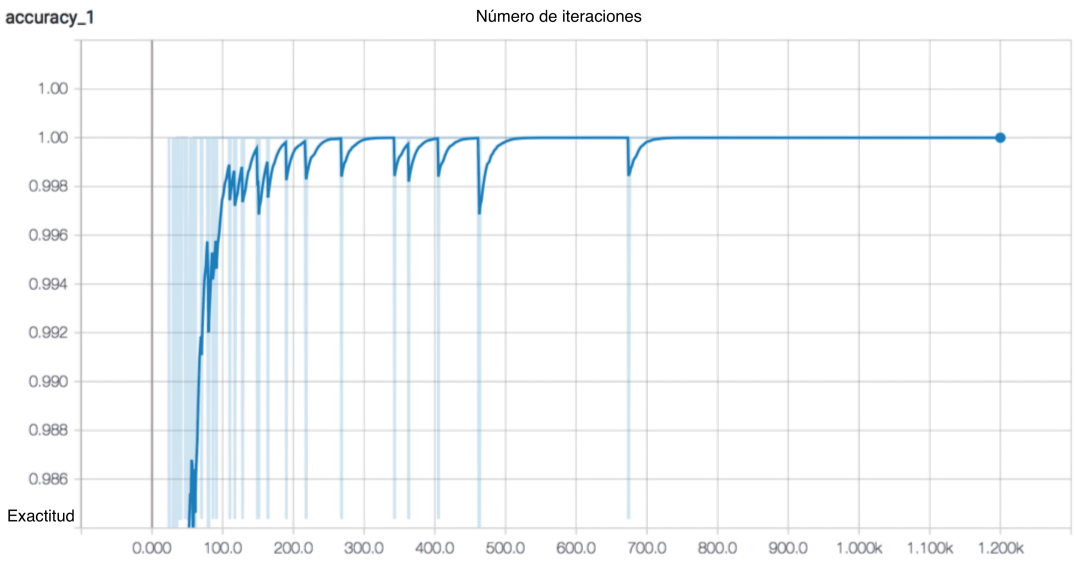


Figure 3.5: Exactitud Red Neuronal Recurrente
Fuente: El Autor
Elaborado por: El Autor

Lo contrario a la exactitud, loss en español la pérdida del entrenamiento de los datos tiene a recibir un esparcimiento y un ruido demasiado alto lo que significa que la técnica de abandono no funciona de una manera correcta en el algoritmo o con los datos trabajados, esto sucede hasta la iteración 500 como se observa en la Figura 3.6.

La pérdida sigue existiendo hasta la iteración 1200 en pequeñas cantidades y es por eso que existe esta gran cantidad de iteraciones para el procedimiento de estabilización en el algoritmo RNN dentro del entrenamiento o aprendizaje de datos o logs

para la clasificación.

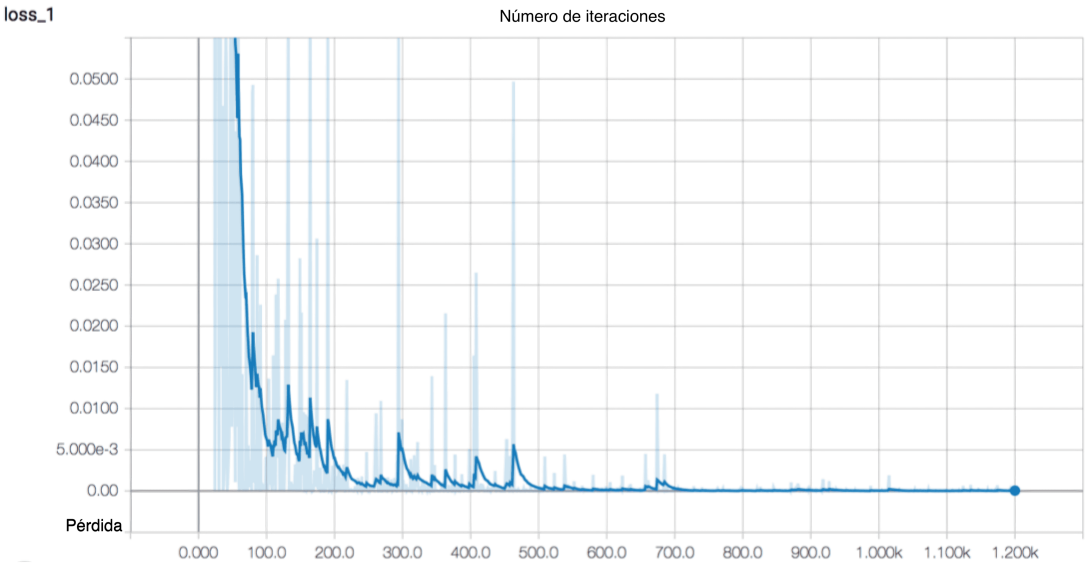


Figure 3.6: Pérdida Red Neuronal Recurrente
Fuente: El Autor
Elaborado por: El Autor

A diferencia de la Red Neuronal Recurrente las pruebas del algoritmo que se realiza mediante la Red Neuronal Convolucional obtiene mejores resultados en la exactitud de los datos como se ilustra en la Figura 3.7 que con los mismos datos de entrenamiento que se usan en ambos algoritmos tomados de la depuración tiende a llegar al 100% de exactitud con menos ruido y esparcimiento de datos en pocas iteraciones, comenzando con una exactitud de menos de 0.86 en la iteración 10 hasta un grado de exactitud completo en la iteración 80 por lo que se estabiliza a partir de la iteración 100, sin embargo continúa la evaluación hasta la 200 del entrenamiento por la pérdida.

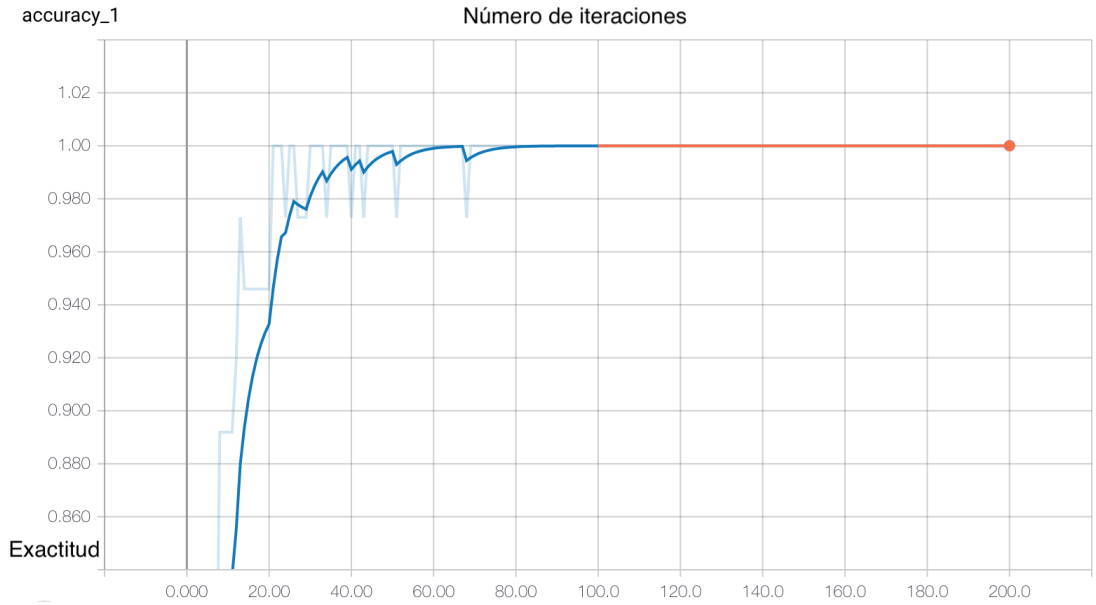


Figure 3.7: Exactitud Red Neuronal Convolucional
Fuente: El Autor
Elaborado por: El Autor

La Figura 3.8 ilustra la pérdida en el algoritmo CNN la cual a pesar de ser la que obtiene el menor grado en todas las pruebas de ruido o esparcimiento se puede notar que estos pequeños datos se logran estabilizar en la iteración 200 llegando al 0% de pérdida.

Para la selección del algoritmo, es el algoritmo CNN que más funcionalidad muestra en un menor tiempo, con los datos a utilizar de tipo texto para la clasificación que junto a la Red Neuronal Convolucional que proporciona y con la implementación de librería TensorFlow es la mejor opción para realizar la ejecución de algoritmo de clasificación con algunos ajustes con la intención de determinar los parámetros que permite lograr resultados más precisos.

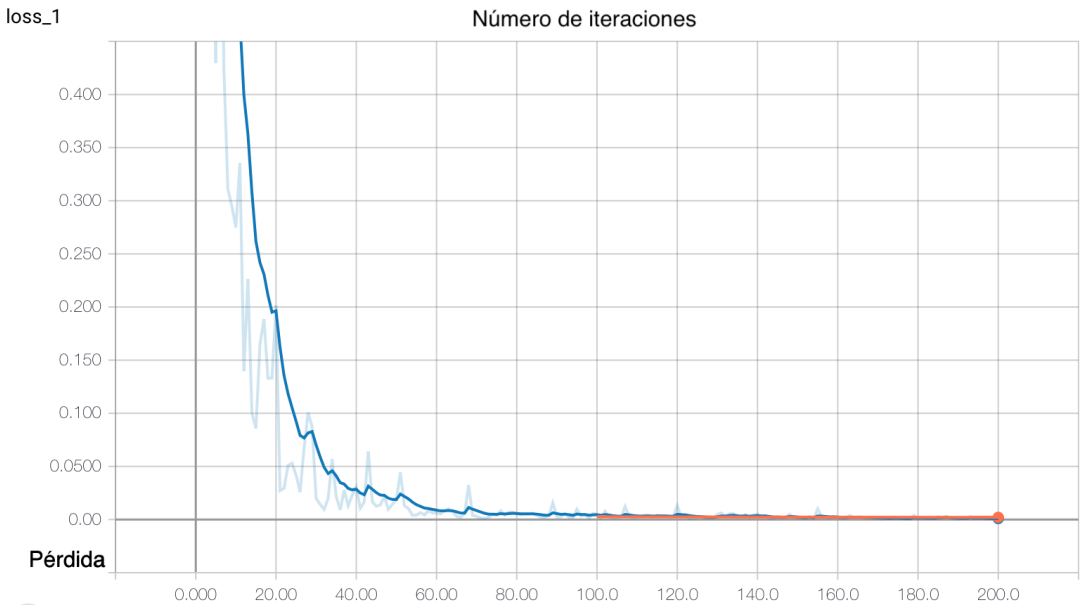


Figure 3.8: Pérdida Red Neuronal Convolucional
Fuente: El Autor
Elaborado por: El Autor

La pérdida en los datos de entrenamiento disminuye a medida que avanza la ejecución del algoritmo, a excepción de algunas fluctuaciones introducidas por el descenso de gradiente de técnicas de regularización como en el proyecto a desarrollar de abandono que introduce ruido aleatorio.

Si la pérdida disminuye, el proceso de entrenamiento va bien es por eso que la selección de algoritmo a trabajar favorece más trabajar con la Red Neuronal Convolucional porque el ruido introducido en ambos casos RNN y CNN tiene a desaparecer de manera rápida y contundente en el algoritmo con CNN el código del algoritmo RNN para la prueba y selección se encuentra en el Anexo 2.

La precisión en cambio, es la medida de cuán buenas son las clasificaciones del modelo. Si el modelo está aprendiendo, la precisión aumenta en algunos casos de clasificación la precisión podría mantenerse en 1 o 100% lo que convendría un gran

logro debido a que el algoritmo funcionaría de una manera óptima independiente de los datos de prueba.

Si el modelo se adapta en exceso, la precisión deja de aumentar e incluso puede comenzar a disminuir, si la pérdida disminuye y la precisión disminuye, el modelo se adapta en exceso lo cuál también estaría incorrecto que no sucede en ninguna prueba de ambos algoritmos pero se logra una mejor precisión con el algoritmo CNN por los métodos de convolución que se explican a detalle en la Sección 4.3.

Para la tarea de clasificación se realiza el aprendizaje con 1 millón de logs para cada algoritmo ambos conjunto de datos idénticos, en la Tabla 3.1 se observa la diferencia de cada uno en base al tiempo que toma cada uno y la puntuación sobre 0.5 donde también se considera en base a los resultados del número de iteraciones que realiza cada algoritmo al conjunto de datos para que exactitud y pérdida sean optimos para la clasificación ya vistos en esta sección.

Table 3.1: Comparación de rendimiento para la selección de algoritmos

Algoritmo de clasificación	**CNN**	**RNN**
Número de iteraciones	200	1200
Tiempo promedio de aprendizaje	20 minutos	1 hora
Puntuación	0.46	0.32

Fuente: El Autor
Elaborado por: El Autor

Finalmente para la selección de algoritmos se toma en cuenta que si la pérdida disminuye y la precisión aumenta se debe a que sus técnicas de regularización están funcionando bien y está luchando contra el problema de sobre ajuste.

Esto es cierto solo si la pérdida comienza a disminuir mientras la precisión continúa aumentando. De lo contrario, si la pérdida sigue creciendo, el modelo es divergente y debe existir alguna causa por lo general, se está usando un valor de tasa de apren-

dizaje demasiado alto.

3.7 Implementación de Red Neuronal Convolucional

Para poder realizar la clasificación de texto dentro del algoritmo y la Red Neuronal se debe permitir varias configuraciones de hiperparámetros, en el código se coloca una clase TextCNN, generando el modelo de la función init.

```
1  import TensorFlow as tf
2  import numpy as np
3
4  class TextCNN(object):
5      """
6      A CNN for text classification.
7      Uses an embedding layer, followed by a convolutional,
       max-pooling and softmax layer.
8      """
9      def __init__(
10       self, sequence_length, num_classes, vocab_size,
11       embedding_size, filter_sizes, num_filters):
12
13 # Implementación ...
```

Para instanciar la clase, se pasa los siguientes argumentos:

- `sequence_length`: Longitud de las oraciones. Se han rellenado todas las oraciones en este caso los logs para tener la misma longitud (59 para el conjunto de datos).
- `num_classes`: Número de clases en la capa de salida, dos en este proyecto

(videos y cursos).

- `vocab_size`: Tamaño del vocabulario. Esto es necesario para definir el tamaño de la capa de inserción, que tendrá la forma `[vocabulary_size, embedding_size]`.
- `embedding_size`: Dimensionalidad de las incrustaciones.
- `filter_sizes`: Cantidad de palabras que van a cubrir los filtros convolucionales. Existe `num_filters` para cada tamaño especificado aquí. En el proyecto se tiene filtros que se deslizan sobre 3, 4 y 5 palabras respectivamente, para un total de 3 * filtros `num_filters`.
- `num_filters`: Cantidad de filtros por tamaño de filtro.

Para los marcadores de posición de entrada, se definen los datos de entrada que se pasa en la red:

```
1 # Marcadores de posición para entrada, salida y abandono
2 self.input_x = tf.placeholder(tf.int32, [None,
     sequence_length], name="input_x")
3 self.input_y = tf.placeholder(tf.float32, [None, num_classes
     ], name="input_y")
4 self.dropout_keep_prob = tf.placeholder(tf.float32, name="
     dropout_keep_prob")
```

`tf.placeholder` crea una variable de marcador de posición que alimenta la red cuando se la ejecuta en el entrenamiento o en el tiempo de prueba. El segundo argumento es la forma del tensor de entrada. None significa que la longitud de esa dimensión podría ser cualquier cosa. En este caso, la primera dimensión es el tamaño del lote, y usando None permite que la red maneje lotes de tamaño arbitrario.

La probabilidad de mantener una neurona en la capa de abandono también es una entrada a la red porque se habilita el abandono solo durante el entrenamiento. Se desactivará al evaluar el modelo.

La primera capa que se define es la capa de incrustación, que mapea índices de palabras de vocabulario en representaciones de vectores de baja dimensionalidad. Básicamente es una tabla de búsqueda que aprende de los datos.

```
1 with tf.device('/cpu:0'), tf.name_scope("embedding"):
2     W = tf.Variable(
3         tf.random_uniform([vocab_size, embedding_size],
   -1.0, 1.0),
4         name="W")
5     self.embedded_chars = tf.nn.embedding_lookup(W, self.
   input_x)
6     self.embedded_chars_expanded = tf.expand_dims(self.
   embedded_chars, -1)
```

Se usa nuevas funciones aquí:

- `tf.device('/ cpu: 0')` fuerza una operación para ser ejecutada en la CPU. De forma predeterminada, TensorFlow intentará poner la operación en la GPU si hay una disponible, pero la implementación de incrustación actualmente no admite GPU y arroja un error si se coloca en la GPU.
- `tf.name_scope` crea un nuevo nombre de ámbito con el nombre “embedding” en español incrustación. El alcance agrega todas las operaciones a un nodo de nivel superior llamado “embedding” para que pueda obtener una buena jerarquía al visualizar la red en TensorBoard.

- `W` es la matriz de integración que aprende durante el entrenamiento. Se inicia usando una distribución uniforme al azar `tf.nn.embedding_lookup` crea la operación de incrustación actual.

  El resultado de la operación de incrustación es un tensor tridimensional de forma `[None, sequence_length, embedding_size]`.

La operación conv2d convolucional de TensorFlow espera un tensor de 4 dimensiones con dimensiones correspondientes a batch, width, height and channel, en español lote, ancho, alto y canal.

El resultado de esta incrustación no contiene la dimensión del canal, por lo que se agregará manualmente, dejando una capa de forma
`[None, sequence_length, embedding_size, 1]`.

Ahora se construye las capas de Convolución seguidas de las capas Max-Pooling en español Máximo de Agrupamiento. Se usa filtros de diferentes tamaños.
Debido a que cada convolución produce tensores de diferentes formas, se debe iterar a través de ellos, crear una capa para cada uno de ellos y luego fusionar los resultados en un gran vector de características.

```
1 pooled_outputs = []
2 for i, filter_size in enumerate(filter_sizes):
3     with tf.name_scope("conv-maxpool-%s" % filter_size):
4         # Capa de convolución
5         filter_shape = [filter_size, embedding_size, 1,
    num_filters]
6         W = tf.Variable(tf.truncated_normal(filter_shape,
    stddev=0.1), name="W")
7         b = tf.Variable(tf.constant(0.1, shape=[num_filters
    ]), name="b")
```

```
8           conv = tf.nn.conv2d(
9               self.embedded_chars_expanded,
10              W,
11              strides=[1, 1, 1, 1],
12              padding="VALID",
13              name="conv")
14          # Se aplica no linealidad
15          h = tf.nn.relu(tf.nn.bias_add(conv, b), name="relu")
16          # Max-pooling over the outputs
17          pooled = tf.nn.max_pool(
18              h,
19              ksize=[1, sequence_length - filter_size + 1, 1,
    1],
20              strides=[1, 1, 1, 1],
21              padding='VALID',
22              name="pool")
23          pooled_outputs.append(pooled)
24
25  # Combina todas las funciones agrupadas
26  num_filters_total = num_filters * len(filter_sizes)
27  self.h_pool = tf.concat(3, pooled_outputs)
28  self.h_pool_flat = tf.reshape(self.h_pool, [-1,
    num_filters_total])
```

Aquí, `W` es la matriz de filtro y h es el resultado de aplicar no linealidad a la salida de convolución. Cada filtro se desliza sobre toda la incrustación, pero varía en la cantidad de palabras que cubre.

`padding=''VALID''` significa que se desliza el filtro sobre la oración sin rellenar los bor-

des, realizando una convolución estrecha que da una salida de la forma `[1, sequence_length - filter_size  1, 1, 1]+`. La realización de la agrupación máxima en la salida de un tamaño de filtro específico que deja con un tensor de forma `[batch_size, 1, 1, num_filters]`.

Este es esencialmente un vector de características, donde la última dimensión corresponde a las características definidas para el proyecto. Una vez que se tiene todos los tensores de salida combinados de cada tamaño de filtro, se combinan en un vector de forma larga `[batch_size, num_filters_total]`. El uso de `-1` en `tf.reshape` le dice a TensorFlow que aplana la dimensión siempre que sea posible.

la Figura 3.9 ilustra la Red Neuronal utilizada (para tamaños de filtro específicos 3, 4 y 5), mientras que la Figura 3.10 ilustra una de las capas convolucionales de la Red Neuronal y las operaciones en TensorBoard aplicadas en la misma.

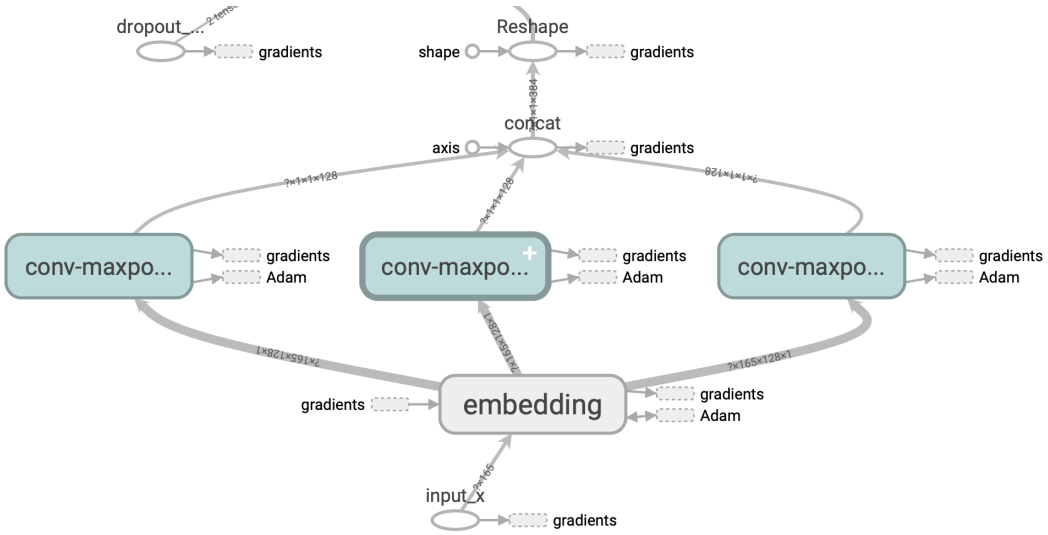


Figure 3.9: Red Neuronal Convolucional en la Librería TensorFlow
Fuente: El Autor
Elaborado por: El Autor

El abandono es quizás el método más popular para regularizar las Redes Neuronales

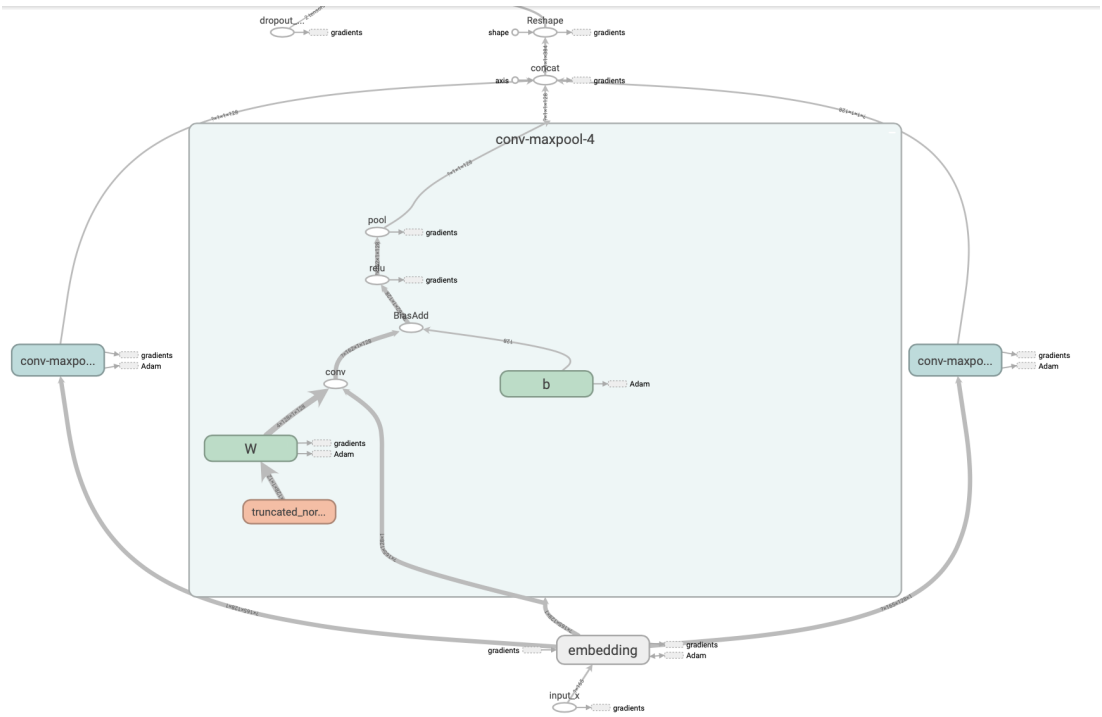


Figure 3.10: Capa Convolucional de la Red Neuronal
Fuente: El Autor
Elaborado por: El Autor

convolucionales, la idea detrás del abandono es simple, donde una capa de abandono en español dropout estocástica desactiva una fracción de sus neuronas, esto evita que las neuronas se adapten juntas y las obliga a aprender funciones útiles individualmente.

La fracción de neuronas que se mantiene habilitada se define mediante la entrada `dropout_keep_prob` en la red del proyecto a desarrollar. Se configura esto como 0.5 durante el entrenamiento, y a 1 (deshabilitar el abandono) durante la evaluación.

```
1 # Add dropout
2 with tf.name_scope("dropout"):
```

```
3     self.h_drop = tf.nn.dropout(self.h_pool_flat, self.
    dropout_keep_prob)
```

Usando el vector de características de la combinación máxima (con abandono aplicado) se generan las clasificaciones haciendo una multiplicación de la matriz y seleccionando la clase con la puntuación más alta.

También se puede aplicar una función softmax para convertir puntajes brutos en probabilidades normalizadas, pero eso no cambiaría las clasificaciones finales.

```
1 with tf.name_scope("output"):
2     W = tf.Variable(tf.truncated_normal([num_filters_total,
    num_classes], stddev=0.1), name="W")
3     b = tf.Variable(tf.constant(0.1, shape=[num_classes]),
    name="b")
4     self.scores = tf.nn.xw_plus_b(self.h_drop, W, b, name="
    scores")
5     self.predictions = tf.argmax(self.scores, 1, name="
    predictions")
```

Aquí, `tf.nn.xw_plus_b` es un contenedor de conveniencia para multiplicar la matriz `Wx b+`.

Usando los puntajes se define la función de pérdida. La pérdida es una medida del error que hace la Red Neuronal del proyecto, y el objetivo es minimizarla. La función de pérdida estándar para problemas de categorización es cross entropy en español la pérdida de entropía cruzada.

```
1 # Calcular la precisión
```

```
2 with tf.name_scope("loss"):
3     losses = tf.nn.softmax_cross_entropy_with_logits(self.
    scores, self.input_y)
4     self.loss = tf.reduce_mean(losses)
```

De esta manera se ha terminado con la definición de la Red Neuronal final a ser utilizada en el proyecto a desarrollar. Para obtener una visión general, en la Figura 3.11 se puede visualizar la red y en el Anexo 3 esta el código del algoritmo para el entrenamiento de la Red Neuronal y el código del algoritmo para la evaluación o clasificación.

3.8 Validación y clasificación

En la presente sección se analiza la clasificación de logs de la obtención y generación del algoritmo de clasificación junto al resultado obtenido con el número de iteraciones a los recursos del curso por usuario en base a las preguntas propuestas en la definición del problema.

El proceso de clasificación de logs generado por el algoritmo de clasificación en el archivo resultante clasification.csv muestra el número de iteraciones a los recursos que el estudiante ha accedido por la generación de eventos en los logs.

A continuación se presenta la clasificación de los logs con su respectivo resultado. Dichas clasificaciones se definen en el Capítulo 4.

3.8.1 Clasificación por tipo de evento

La clasificación resultante de la ejecución del algoritmo mostrada en la Figura 3.12 en la que el valor en la columna de clasificación al ser 0 significa que las acciones que el estudiante ejecuta son acciones negativas o sin importancia como una revisión a la información del curso, perfil del estudiante y navegación en diferentes cursos que

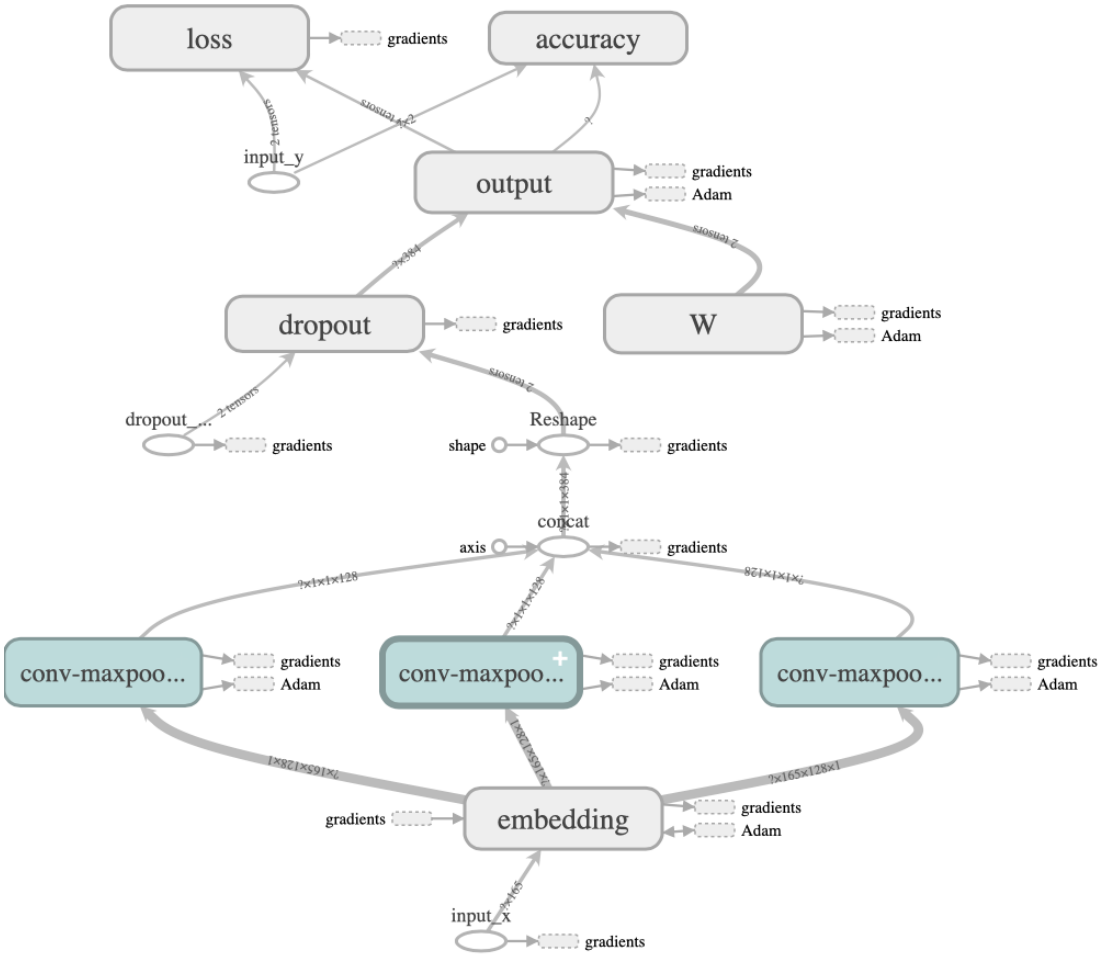


Figure 3.11: Red Neuronal Convolucional Final
Fuente: El Autor
Elaborado por: El Autor

aún no se ha registrado que aunque son acciones registradas en las lineas de logs, no aportan calificación alguna sobre los cursos y como resultado el algoritmo los clasifica como eventos negativos o que no constan dentro de la clasificación.

El resultado de la clasificación es 1 cuando el algoritmo encuentra en cada log eventos relacionados a la ejecución de cuestionarios, videos y tareas dentro de un curso, por lo que se deduce que el estudiante ha revisado o está constantemente ingresando a

los contenidos del curso dentro de la plataforma es decir este realiza los ejercicios propuestos.

```
In [73]: import pandas as pd
         cols = ['log', 'clasificacion']
         df = pd.read_csv('clasification.csv', index_col=False, sep=',', names=cols);
         df.sample(n=10)
```

Out[73]:

	log	clasificacion
149972	username gmalvarado5 , event type courses cour...	1.0
186332	username anagabrielav , event source browser ,...	1.0
629115	username lorenac , event source browser , name...	0.0
807730	username andreacarrion , event type i18n js , ...	1.0
277403	username anagabrielav , event source browser ,...	1.0
92582	username jarodriguez21 , event type i18n js , ...	1.0
170662	username jbgarcia3 , event type courses course...	1.0
591281	username celiaheras , event source browser , n...	1.0
87716	username mariaeugeniamendez , event type cours...	1.0
315993	username marfer2017 , event type courses cours...	0.0

Figure 3.12: Resultado de la clasificación de logs
Fuente: El Autor
Elaborado por: El Autor

3.8.2 Clasificación de tipo de evento por estudiante

La siguiente clasificación separa los usuarios de los logs a partir de los resultados anteriores, sin embargo como cada log es una acción ejecutada en el servidor de la plataforma o computador del estudiante existe una cantidad variable de iteraciones en los diferentes tipo de evento por cada estudiante como se observa en la Figura 3.13.

Es necesario agrupar por nombre de usuario del estudiante para evitar la redundancia de datos lo que permite hacer un conteo de cuantas iteraciones positivas ha realizado en los cursos que se encuentra siguiendo.

Se toma como clasificación positiva los logs que han resultado en 1 que se definen

en eventos que implican que el estudiante tiende a realizar tareas para crear una tendencia por aprobar los cursos y 0 cuando el estudiante ingresa solo a observar la plataforma.

```
In [95]: df.head(n=10)
```

Out[95]:

	clasificacion	nombre_usuario
0	0.0	asdfdsa
1	0.0	342543
2	1.0	asfd
3	1.0	sfdf342
4	1.0	lesliejim
5	1.0	fernandaalvarez
6	0.0	fernandaalvarez
7	1.0	fernandaalvarez
8	1.0	fernandaalvarez
9	1.0	fernandaalvarez

Figure 3.13: Resultado de la clasificación por estudiante
Fuente: El Autor
Elaborado por: El Autor

3.8.3 Clasificación de iteraciones por estudiante

La siguiente clasificación agrupa a los usuarios con el número de iteraciones positivas relacionados con la plataforma y los diferentes cursos en el dataset procesado, se puede observar en la Figura 3.14 como existe una diferencia muy alta entre algunos resultados esto es debido a que cada acción ejecutada se muestra en diferentes logs.

Es decir si un estudiante entra a un video/presentación o tarea y solo ve parte del video/presentación o realiza algunos ejercicios de la tarea y los finaliza otro día también cuenta como clasificaciones positivas.

```
In [102]: sinordenar = df2.sort_values(['nombre_usuario'], ascending=False)
          sinordenar.head(10)
```

Out[102]:

	nombre_usuario	#_iteraciones
2983	zrllanos	308.0
2982	zoebeatrizma	2.0
2981	zairy	269.0
2980	zadiel	70.0
2979	yyolmedo	156.0
2978	yvpezo	1.0
2977	yvojeda	164.0
2976	yuryko	31.0
2975	yuridia valladares	266.0
2974	yuriconforme	77.0

Figure 3.14: Resultado de número de iteraciones
Fuente: El Autor
Elaborado por: El Autor

El promedio para que un estudiante se encuentre dentro de la clasificación de estudiantes con tendencia a aprobar es de al menos 199 iteraciones en total.
En la Figura 3.15 se visualiza el resultado de las iteraciones que cada estudiante posee en la plataforma y de los 80 últimos estudiantes que se toma como muestra 35 están con tendencia ha ser aprobados y el 45 restante ha ser reprobados.

A partir del resultado se los agrupa a los usuarios por número de iteraciones con la finalidad de determinar con un grado de mayor eficiencia los estudiantes que están con tendencia aprobar o con mayor interés en el estudio de los cursos que se han registrado en la plataforma.

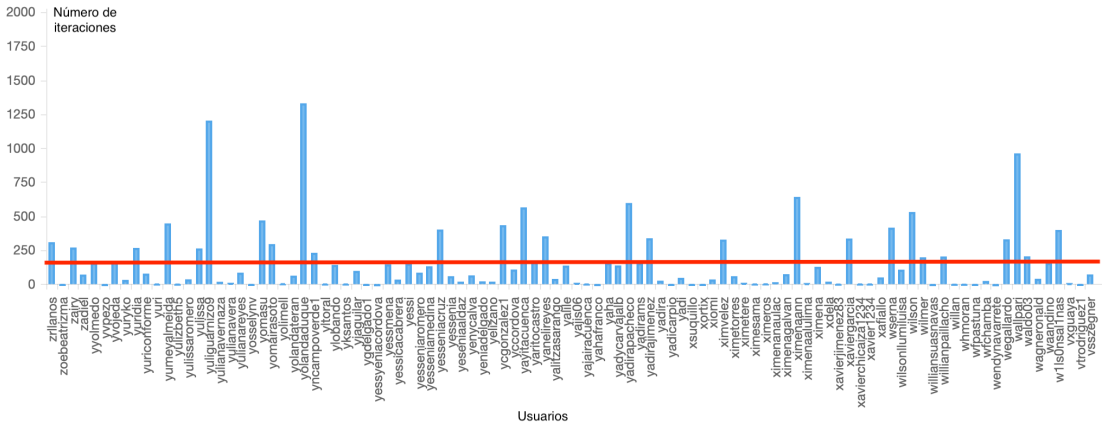


Figure 3.15: Gráfico del número de iteraciones en los cursos
Fuente: El Autor
Elaborado por: El Autor

La siguiente clasificación agrupa los estudiantes con el mayor número de iteraciones positivas en la plataforma, por lo que en este caso la Figura 3.16 muestra los estudiantes aprobados y que tienen un mayor nivel de participación en los cursos.

En la Figura 3.17 permite analizar que de los 57.058 estudiantes registrados en la plataforma el 35% de estudiantes tienen la probabilidad de aprobar al pasar las 199 iteraciones de las clasificaciones positivas sobre la plataforma.

3.9 Discusión

Como se observa en la Tabla 3.2, se logra determinar los estudiantes que no interactúan con los curso, o componentes dentro de la plataforma y están por reprobar antes de que finalice el periodo o en cualquier momento utilizando los logs existentes con el algoritmo de clasificación implementado.

En este caso de los 57.058 estudiantes los que tienen un número de eventos acciona-

```
In [105]: resultado = df2.sort_values('#_iteraciones', ascending=False)
          resultado.head(20)
```

Out[105]:

	nombre_usuario	#_iteraciones
159	anagabrielav	111411.0
1136	ingridlojant	37792.0
939	franciscobuenano	14950.0
784	egsantos1	9644.0
2448	pezambrano	8887.0
1932	mariadelcisne	4201.0
2778	thaliaangamarca	4118.0
2356	pablovillarreal123	4081.0
1718	lili2017	3751.0
810	elizita	3735.0
1923	maria piedad toro	3576.0
525	cristinatorres	3537.0
2156	mirn1978	3507.0
1745	llsantillan	3427.0
527	cruzcaya labanda	3200.0
1585	katherineguayllas	2590.0
2186	mlcordova	2538.0
1582	katherine vasquez 1995	2343.0
175	andrea pisco91	2311.0
2751	tannyaborbor	2219.0

Figure 3.16: Resultado de estudiantes aprobados

Fuente: El Autor

Elaborado por: El Autor

Table 3.2: Discusión

Estado	Promedio iteraciones positivas	Promedio de estudiantes
Por Aprobar	778.137232845894	10.000 - 20.000
Por Reprobar	40.5425023877746	40.000 - 50.000

Fuente: El Autor

Elaborado por: El Autor

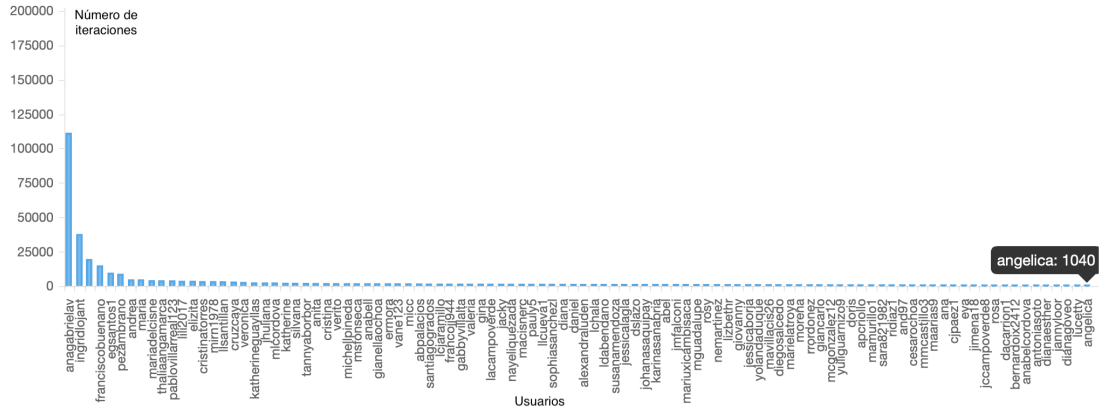


Figure 3.17: Gráfico de estudiantes aprobados
Fuente: El Autor
Elaborado por: El Autor

dos o iteraciones positivas mayor a 199 son entre 10.000 y 20.000 y entre 40.000 y 50.000 tienen tendencia a ser reprobados, es decir entre el 65% los estudiantes suelen reprobar los cursos.

La cantidad exacta de estudiantes aprobados y reprobados se analizan en el Capítulo 4 debido a que se entrena el algoritmo una segunda vez con la adición de datos pertinentes a la evaluación de estudiantes reprobados y aprobados en cursos previos dentro de la plataforma.

CAPÍTULO 4

ANÁLISIS DE RESULTADOS

En este capítulo se presentan los resultados del análisis de los datos clasificados y obtenidos en el Capítulo 3. Estos resultados muestran la mejora general que consigue en gran parte el algoritmo al clasificar el dataset de logs inicial y la evolución particular del algoritmo en cuanto a las características de exactitud y pérdida anteriormente comentadas. Se destaca especialmente las clasificaciones por tipo de evento, así como clasificaciones sobre los estudiantes aprobados y reprobados que influyen significativamente en la plataforma al ser directamente los usarios que intervienen principalmente, que dan lugar a dichos resultados.

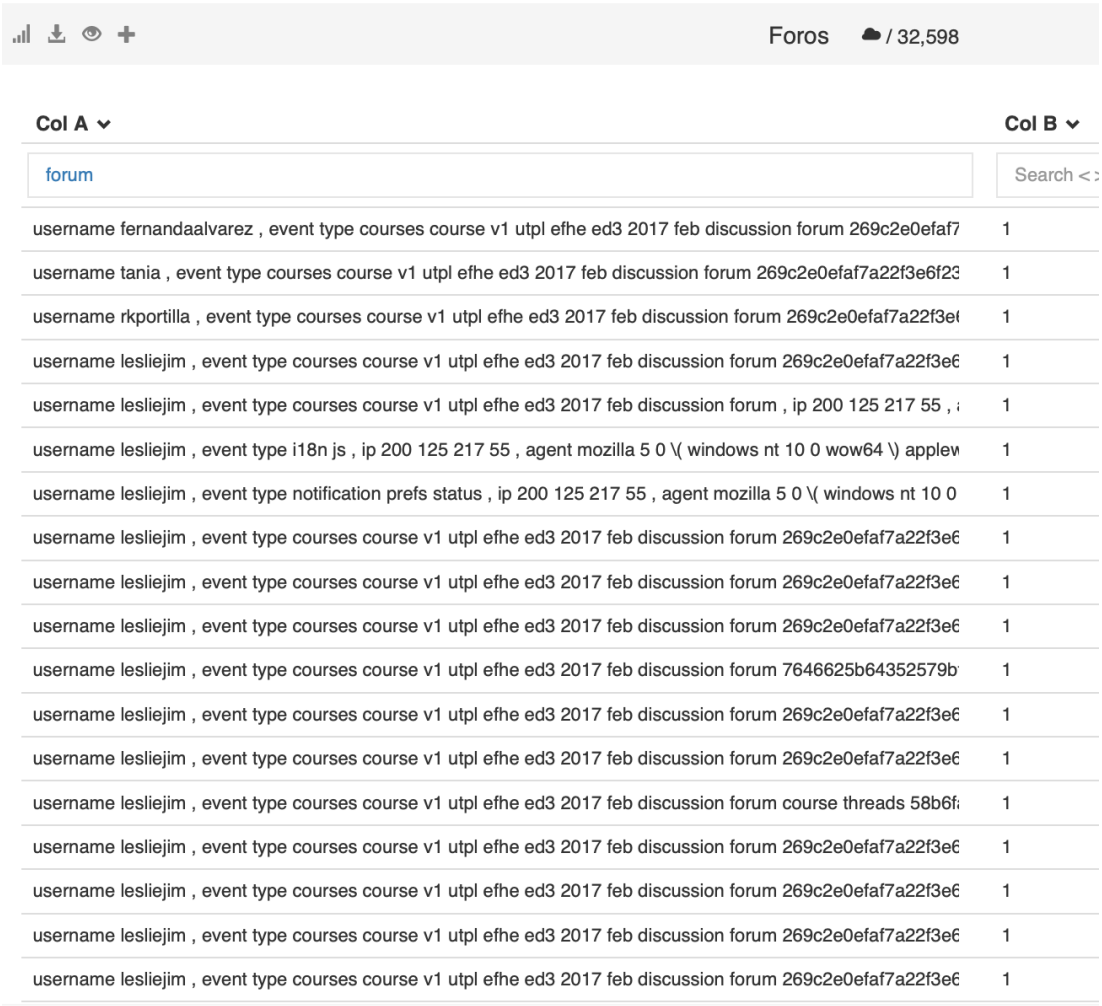
4.1 Resultados de clasificación de logs por tipo de evento

En la Figura 4.1 se observa la clasificación del dataset por tipo de evento en foros accionado por el servidor de la plataforma o el estudiante clasificados como iteraciones positivas por el algoritmo.

En algunos casos de la clasificación como se ve en la Figura 4.2 existen eventos como problemas calificados que son un ejemplo claro de un evento accionado por el servidor de la plataforma hacia el usuario que no lo cuenta como iteración positiva ya que el usuario no interactúa con la plataforma a diferencia de la Figura 4.3 que muestra las iteraciones hacia los videos o presentaciones de los cursos en que el estudiante observa.

4.2 Resultados de clasificación de estudiantes por número de iteración

La clasificación resultante permite determinar si un estudiante está por aprobar o reprobar mediante el número de iteraciones o uso de la plataforma, permite que en cualquier momento que se ejecute el algoritmo se puede determinar este resultado:

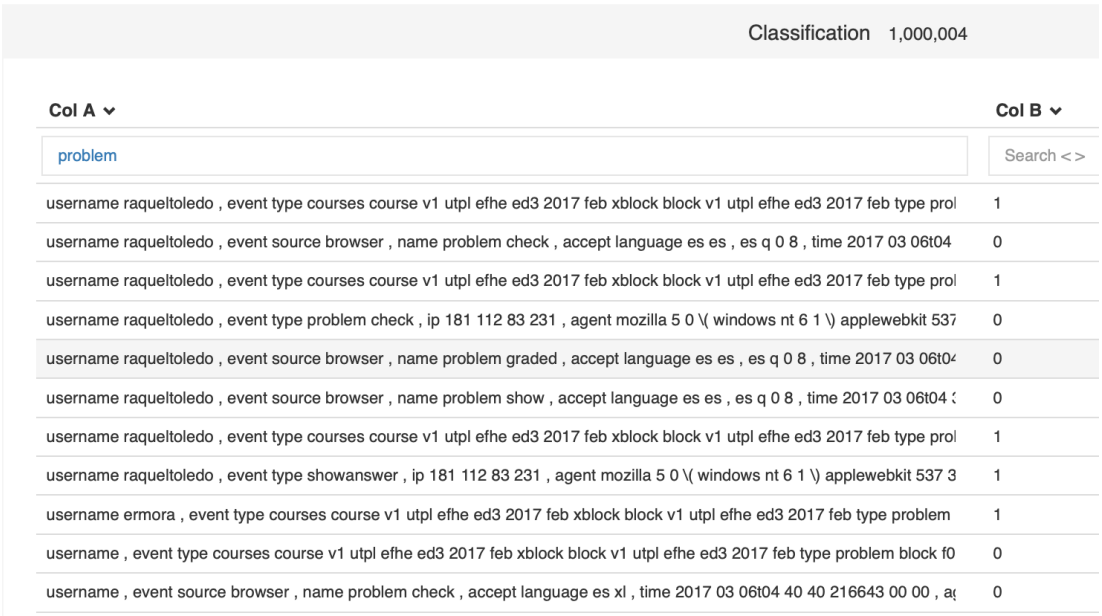


Foros / 32,598

Col A	Col B
forum	Search
username fernandaalvarez , event type courses course v1 utpl efhe ed3 2017 feb discussion forum 269c2e0efaf7	1
username tania , event type courses course v1 utpl efhe ed3 2017 feb discussion forum 269c2e0efaf7a22f3e6f23	1
username rkportilla , event type courses course v1 utpl efhe ed3 2017 feb discussion forum 269c2e0efaf7a22f3e	1
username lesliejim , event type courses course v1 utpl efhe ed3 2017 feb discussion forum 269c2e0efaf7a22f3e6	1
username lesliejim , event type courses course v1 utpl efhe ed3 2017 feb discussion forum , ip 200 125 217 55 ,	1
username lesliejim , event type i18n js , ip 200 125 217 55 , agent mozilla 5 0 \(windows nt 10 0 wow64 \) applev	1
username lesliejim , event type notification prefs status , ip 200 125 217 55 , agent mozilla 5 0 \(windows nt 10 0	1
username lesliejim , event type courses course v1 utpl efhe ed3 2017 feb discussion forum 269c2e0efaf7a22f3e6	1
username lesliejim , event type courses course v1 utpl efhe ed3 2017 feb discussion forum 269c2e0efaf7a22f3e6	1
username lesliejim , event type courses course v1 utpl efhe ed3 2017 feb discussion forum 269c2e0efaf7a22f3e6	1
username lesliejim , event type courses course v1 utpl efhe ed3 2017 feb discussion forum 7646625b64352579b	1
username lesliejim , event type courses course v1 utpl efhe ed3 2017 feb discussion forum 269c2e0efaf7a22f3e6	1
username lesliejim , event type courses course v1 utpl efhe ed3 2017 feb discussion forum 269c2e0efaf7a22f3e6	1
username lesliejim , event type courses course v1 utpl efhe ed3 2017 feb discussion forum course threads 58b6f	1
username lesliejim , event type courses course v1 utpl efhe ed3 2017 feb discussion forum 269c2e0efaf7a22f3e6	1
username lesliejim , event type courses course v1 utpl efhe ed3 2017 feb discussion forum 269c2e0efaf7a22f3e6	1
username lesliejim , event type courses course v1 utpl efhe ed3 2017 feb discussion forum 269c2e0efaf7a22f3e6	1
username lesliejim , event type courses course v1 utpl efhe ed3 2017 feb discussion forum 269c2e0efaf7a22f3e6	1

Figure 4.1: Resultado de la clasificación de logs por foros
Fuente: El Autor
Elaborado por: El Autor

En la Figura 4.4 se observa la clasificación de los estudiantes con tendencia a aprobar ordenados alfabéticamente por el nombre de usuario y en la Figura 4.5 los estudiantes a reprobar de igual manera organizados alfabéticamente.



Classification 1,000,004

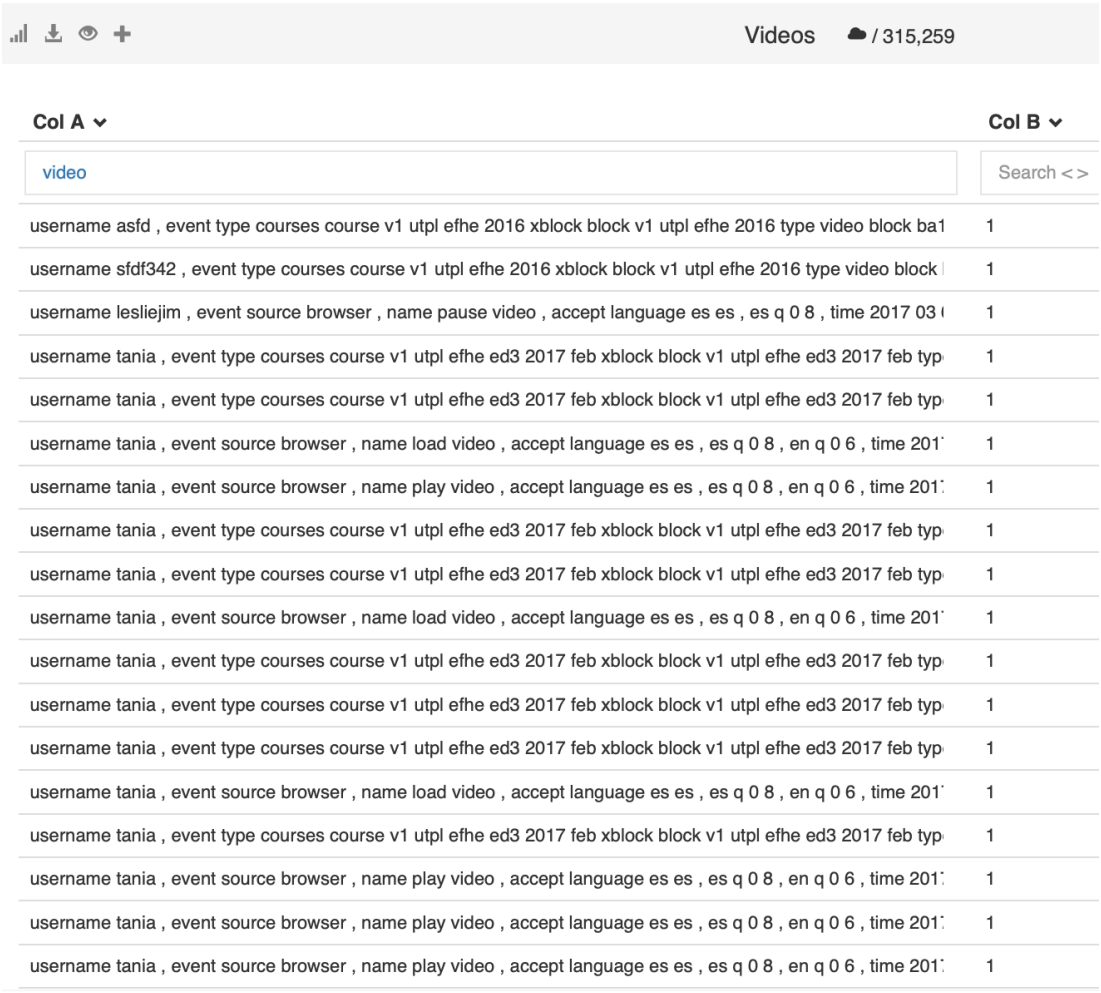
Col A	Col B
problem	Search < >
username raqueltoledo , event type courses course v1 utpl efhe ed3 2017 feb xblock block v1 utpl efhe ed3 2017 feb type prol	1
username raqueltoledo , event source browser , name problem check , accept language es es , es q 0 8 , time 2017 03 06t04	0
username raqueltoledo , event type courses course v1 utpl efhe ed3 2017 feb xblock block v1 utpl efhe ed3 2017 feb type prol	1
username raqueltoledo , event type problem check , ip 181 112 83 231 , agent mozilla 5 0 \(windows nt 6 1 \) applewebkit 537	0
username raqueltoledo , event source browser , name problem graded , accept language es es , es q 0 8 , time 2017 03 06t04	0
username raqueltoledo , event source browser , name problem show , accept language es es , es q 0 8 , time 2017 03 06t04 3	0
username raqueltoledo , event type courses course v1 utpl efhe ed3 2017 feb xblock block v1 utpl efhe ed3 2017 feb type prol	1
username raqueltoledo , event type showanswer , ip 181 112 83 231 , agent mozilla 5 0 \(windows nt 6 1 \) applewebkit 537 3	1
username ermora , event type courses course v1 utpl efhe ed3 2017 feb xblock block v1 utpl efhe ed3 2017 feb type problem	1
username , event type courses course v1 utpl efhe ed3 2017 feb xblock block v1 utpl efhe ed3 2017 feb type problem block f0	0
username , event source browser , name problem check , accept language es xl , time 2017 03 06t04 40 40 216643 00 00 , a	0

Figure 4.2: Resultado de la clasificación de logs por problemas
Fuente: El Autor
Elaborado por: El Autor

4.3 Resultados de clasificación de estudiantes aprobados/reprobados

Para lograr un resultado mejor en el aprendizaje del algoritmo se agregaron datos en la fase de entrenamiento, estos datos son referentes a cursos sobre la plataforma previos en base a la relación del estudiante y si el curso fue aprobado o reprobado.
En base a esa colección de datos que se agrega al entrenamiento los resultados concluyentes del algoritmo muestran en la Figura 4.6 una mejora notable de un 40% en estabilidad al no tener tanto ruido en las iteraciones. En base a esto sobre las mismas 200 iteraciones iniciales se observa que en la iteración 52 llega a la exactitud de 0.999 como resultado final en el entrenamiento de los mismos datos.

El resultado de la implementación de estos datos sobre el entrenamiento también añade una mejora de un 40% en estabilidad por la disminución de ruido y sobre las mismas 200 iteraciones que se presenta en el Capítulo 3 como resultado de la clasi-



Videos / 315,259

Col A	Col B
video	Search < >
username asfd , event type courses course v1 utpl efhe 2016 xblock block v1 utpl efhe 2016 type video block ba1	1
username sfdf342 , event type courses course v1 utpl efhe 2016 xblock block v1 utpl efhe 2016 type video block	1
username lesliejim , event source browser , name pause video , accept language es es , es q 0 8 , time 2017 03	1
username tania , event type courses course v1 utpl efhe ed3 2017 feb xblock block v1 utpl efhe ed3 2017 feb typ	1
username tania , event type courses course v1 utpl efhe ed3 2017 feb xblock block v1 utpl efhe ed3 2017 feb typ	1
username tania , event source browser , name load video , accept language es es , es q 0 8 , en q 0 6 , time 201	1
username tania , event source browser , name play video , accept language es es , es q 0 8 , en q 0 6 , time 201	1
username tania , event type courses course v1 utpl efhe ed3 2017 feb xblock block v1 utpl efhe ed3 2017 feb typ	1
username tania , event type courses course v1 utpl efhe ed3 2017 feb xblock block v1 utpl efhe ed3 2017 feb typ	1
username tania , event source browser , name load video , accept language es es , es q 0 8 , en q 0 6 , time 201	1
username tania , event type courses course v1 utpl efhe ed3 2017 feb xblock block v1 utpl efhe ed3 2017 feb typ	1
username tania , event type courses course v1 utpl efhe ed3 2017 feb xblock block v1 utpl efhe ed3 2017 feb typ	1
username tania , event type courses course v1 utpl efhe ed3 2017 feb xblock block v1 utpl efhe ed3 2017 feb typ	1
username tania , event source browser , name load video , accept language es es , es q 0 8 , en q 0 6 , time 201	1
username tania , event type courses course v1 utpl efhe ed3 2017 feb xblock block v1 utpl efhe ed3 2017 feb typ	1
username tania , event source browser , name play video , accept language es es , es q 0 8 , en q 0 6 , time 201	1
username tania , event source browser , name play video , accept language es es , es q 0 8 , en q 0 6 , time 201	1
username tania , event source browser , name play video , accept language es es , es q 0 8 , en q 0 6 , time 201	1

Figure 4.3: Resultado de la clasificación de logs por videos
Fuente: El Autor
Elaborado por: El Autor

ficación se obtiene una pérdida de 0.010 en la iteración 76 como se observa en la Figura 4.7

La clasificación que se realiza sobre los datos de evaluación tienen un porcentaje promedio de más del 70% de exactitud por lo que llega a ser muy preciso al utilizar una Red Neural Convolucional.

Clasificacion #_iteraciones >= 199
/ (889)

nombre_usuario	#_iteraciones
1997	325
aaespinel1	421
aaherrera3	539
abel	1,454
abied	366
abigail lara2	812
abpalacios	1,809
abrahambebe13junio	679
abrojas2	280
abroman	276
acaicedo2	483
achicasadobay	244
acoba	829
admalla	438
adolfoarguello	978
adrianaangamarca	385

Figure 4.4: Resultado de la clasificación de estudiantes por aprobar
Fuente: El Autor
Elaborado por: El Autor

El resultado de los estudiantes que no han aprobado los cursos propuestos en la plataforma en los últimos 3 años son un total de 44.194 estudiantes en los que el algoritmo de clasificación como se ve en la Figura 4.8 los ha clasificado correctamente ya que tienen de nota 0 y no poseen certificado.

Clasificacion #_iteraciones < 199
/ (2,094)

Nombre Usuario	Csv_ Iteraciones
0803029297	3
342543	0
4fsotomayor	45
62000	4
a7xleonel	1
aabenitez2	153
aaroncabrera	29
aasanchez8	114
abgranja	29
abigail	11
abigail23	55
ablojan	8
abproano	0
acpozo1	151
acvicente	1
acvillavicencio	10

Figure 4.5: Resultado de la clasificación de estudiantes por reprobar
Fuente: El Autor
Elaborado por: El Autor

Los estudiantes clasificados como aprobados que denotan en la Figura 4.9 deben tener certificado y una nota mayor a 0.6 y como resultado del total de 57.058 estudiantes registrados en la plataforma explicados en el Capítulo 3.

Como resultado el algoritmo de clasificación realiza el proceso correctamente y da

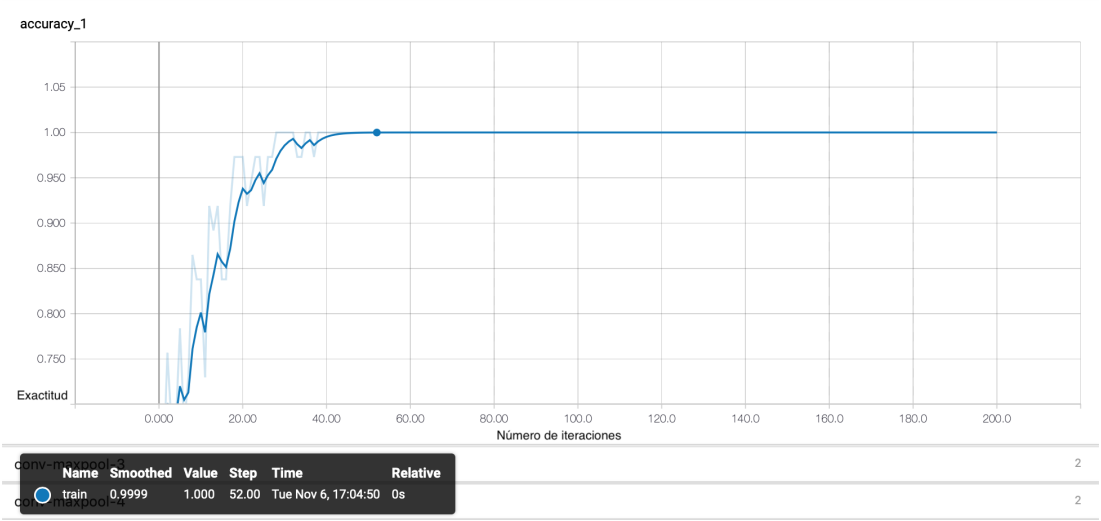


Figure 4.6: Resultado de la exactitud mejorada
Fuente: El Autor
Elaborado por: El Autor

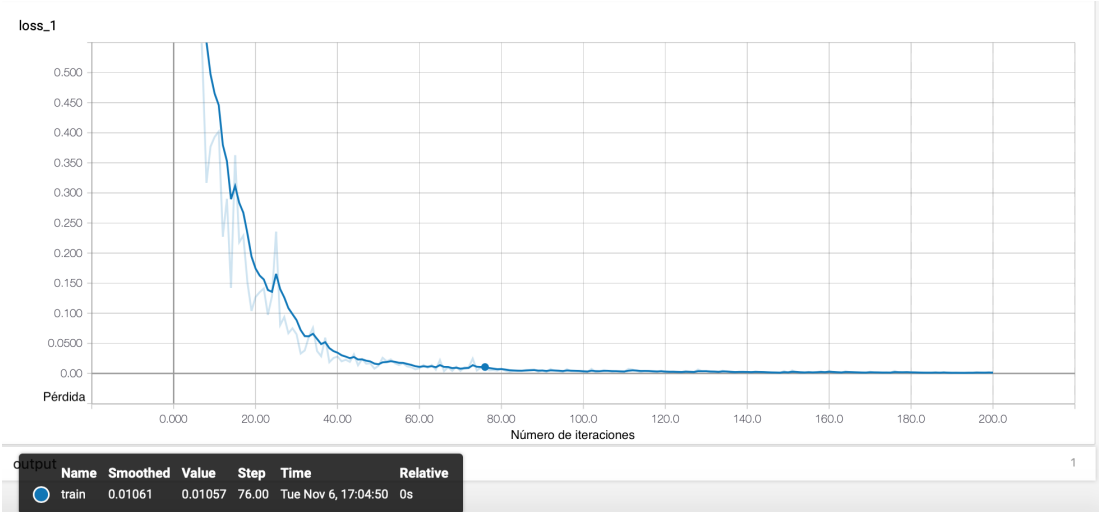


Figure 4.7: Resultado de la pérdida mejorada
Fuente: El Autor
Elaborado por: El Autor

como salida los 12.864 estudiantes restantes que han logrado aprobar los cursos de la plataforma en base a los últimos 3 años dispuestos en el dataset.

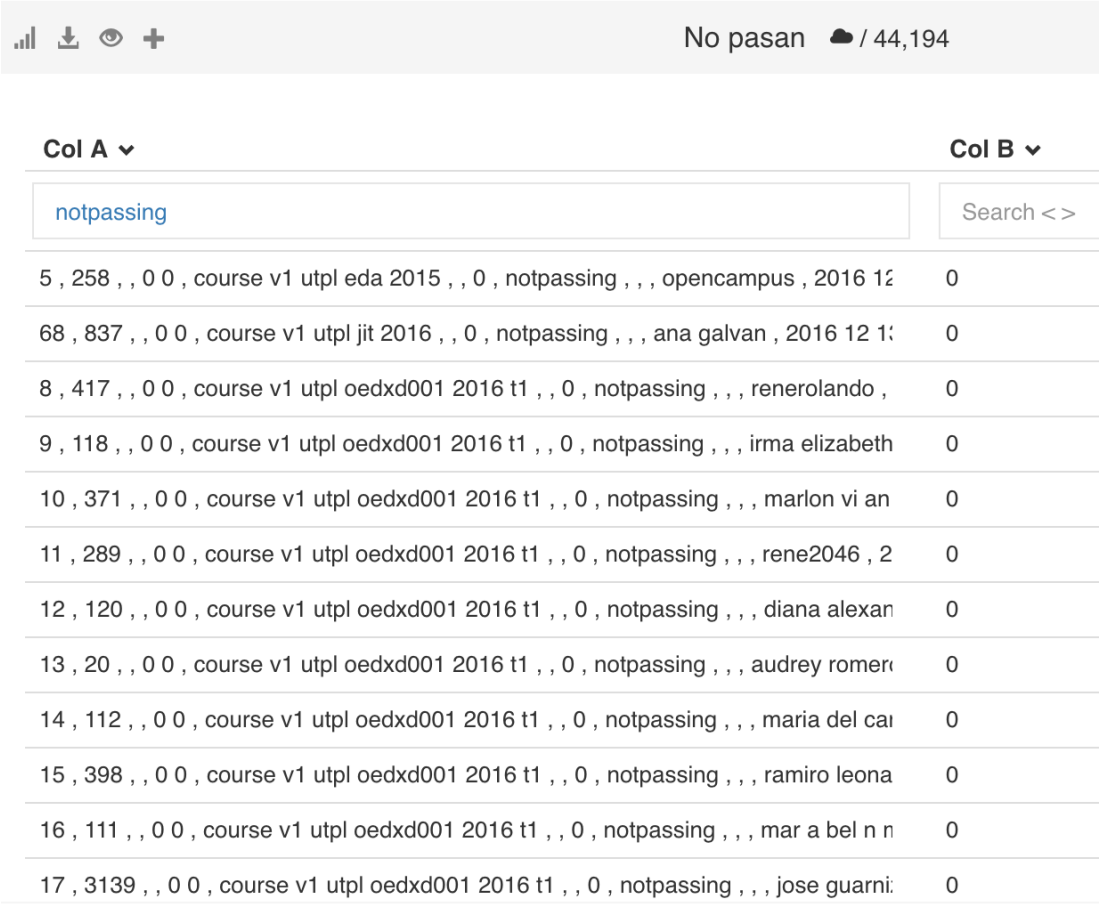


Figure 4.8: Resultado de la clasificación de estudiantes reprobados
Fuente: El Autor
Elaborado por: El Autor

Del total de 1.054.457 logs utilizados para la evaluación mediante el algoritmo de clasificación como resultado de este aprendizaje es capaz de clasificar tanto las iteraciones positivas sobre el tipo de evento accionado expuesto en el log como la clasificación en base a un curso y estudiante sobre su aprobación y reprobación en el mismo.

En la Figura 4.10 se demuestra la clasificación del algoritmo que da un resultado de 368.293 logs positivos y 686.163 logs restantes negativos. De esta manera como se ve en el Capítulo 3 que existe una tendencia en el que la mayoría de estudiantes son

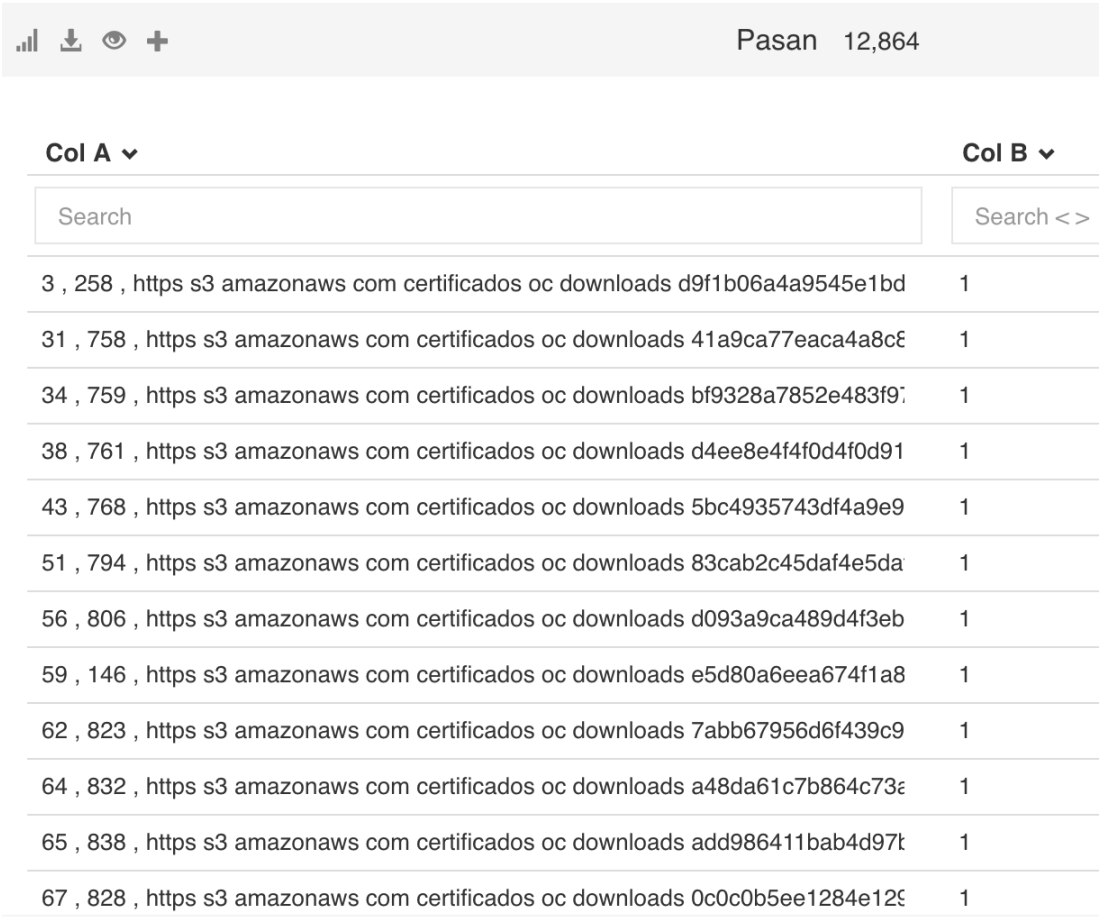


Figure 4.9: Resultado de la clasificación de estudiantes aprobados
Fuente: El Autor
Elaborado por: El Autor

reprobados, concuerda con esta clasificación al demostrarse que existe una menor cantidad de logs positivos y una mayor cantidad de logs negativos.

Al finalizar la ejecución del algoritmo, este muestra algunos valores finales que se obtiene de la evaluación realizada. Estos datos se muestran en la Tabla 4.1 tales como la cantidad de líneas de logs utilizados para su entrenamiento y evaluación, el promedio de las métricas del algoritmo de TensorFlow (pérdida y exactitud) mostrando un 99% de eficiencia en clasificación, la mejora de estas métricas en comparación a

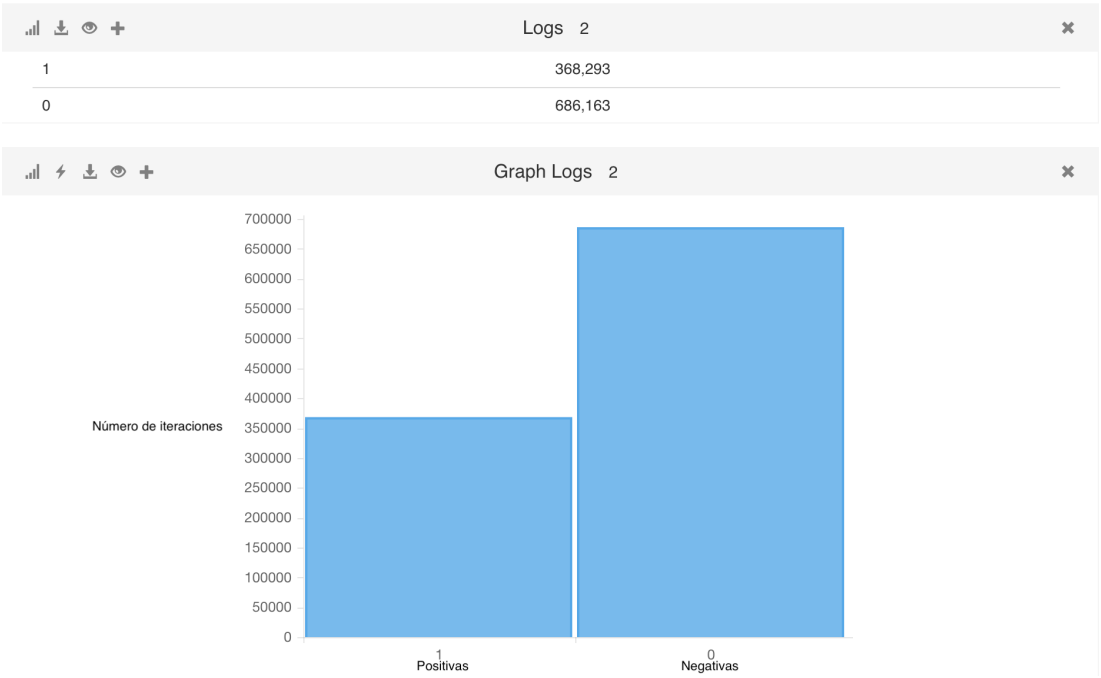


Figure 4.10: Logs clasificados como aprobados/reprobados
Fuente: El Autor
Elaborado por: El Autor

las pruebas del primer algoritmo demostrado en el Capítulo 3, también se obtiene el tiempo de ejecución total en minutos de la clasificación y la plataforma sobre la que el algoritmo trabajo (CPU).

Table 4.1: Datos de la evaluación del algoritmos

Parametros	**Valor**
Logs para evaluación	1.054.457
Logs para entrenamiento	6.000.000
Promedio pérdida	0.00791444
Promedio exactitud	1
Mejora final	40%
Tiempo de evaluación	16:31:22
Plataforma de evaluación	CPU

Fuente: El Autor
Elaborado por: El Autor

CONCLUSIONES

Tras la culminación del presente trabajo de titulación se concluye que:

La implementación del algoritmo presentó porcentajes de 100% de exactitud, 1% de perdida para la clasificación. El algoritmo con Red Neuronal Convolucional en el entrenamiento de datos realiza 10% menos iteraciones sobre los datos que el algoritmo con Red Neuronal Recurrente y tiene un 50% menos de ruido siendo el primero mejor para la clasificación de datos de texto.

El algoritmo fué capaz de clasificar los eventos de la plataforma OpenEdx off-task como iteraciones negativos y los eventos más influyentes a la hora de determinar el estado de aprobación de los estudiantes como iteraciones positivas.

El análisis de las pruebas obtenidas de las clasificaciones dieron como resultado del total de 1.054.457 logs, 368.293 logs como iteraciones positivas, 686.163 logs como iteraciones negativas. Dentro de estos logs del total de 57.058 estudiantes, 44.194 estudiantes fueron clasificados como aprobados y 12.864 estudiantes como reprobados.

Se logró determinar la cantidad de iteraciones dentro de la tendencia del algoritmo de clasificación para que un estudiante apruebe el curso, deben existir al menos 199 iteraciones positivas en cada curso para que el estudiante pueda aprobarlos.

El uso del conjunto de datos de logs obtenido de la plataforma OpenEdx de la iniciativa OpenCampus siendo este de tipo texto proporciona una contribución al campo de

Deep Learning, ya que existen escasos trabajos sobre la implementación de algoritmos de clasificación sobre este tipo de dato.

TRABAJOS FUTUROS

Como continuación de este Trabajo de Titulación y al ser este un proyecto práctico, existen diversas líneas en campo de Deep Learning y algoritmos de clasificación que quedan abiertas y en las que es posible continuar trabajando. Durante el desarrollo de esta tesis han surgido algunas líneas futuras que quedan abiertas y que se esperan resolver en un futuro.

A continuación se presentan algunos trabajos futuros que pueden desarrollarse como resultado de este trabajo o que, por exceder el alcance de este Trabajo de Titulación, no han podido ser tratados con la suficiente profundidad. Entre los posibles trabajos futuros se destacan:

- Realizar un análisis y comparación de diferentes tipos de algoritmos de clasificación en tecnologías como el Machine Learning y la Inteligencia Artificial.
- Mejorar y adecuar la orientación de los datasets de la plataforma OpenEdx con datos etiquetados para realizar la clasificación de datos textuales con algoritmos de TensorFlow.
- Desarrollar e implementar la metodología propuesta tanto en diferentes organizaciones educativas o servidores de iniciativas con la plataforma OpenEdx, lo que permitirá comprobar su eficacia.
- Realizar pruebas en algoritmos de clasificación con varios conjuntos de datos de texto y verificar su rendimiento en la ejecución con el GPU y CPU.

BIBLIOGRAFÍA

Tensorflow architecture | tensorflow. 29, 30

Arel, I., Rose, D., and Karnowski, T. (2010). Deep machine learning - a new frontier in artificial intelligence research [research frontier]. 5:13–18. 14, 17, 18, 19

Bengio, Y. (2009). Learning deep architectures for ai. *Foundations and Trends® in Machine Learning*, 2(1):1–127. 9, 13

Bengio, Y. (2011). Learning Deep Architectures for AI. 9(1):1–4. 10

Bengio, Y. (2012). Deep learning of representations for unsupervised and transfer learning. In Guyon, I., Dror, G., Lemaire, V., Taylor, G., and Silver, D., editors, *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, volume 27 of *Proceedings of Machine Learning Research*, pages 17–36, Bellevue, Washington, USA. PMLR. 21

Bengio, Y. and LeCun, Y. (2007). Scaling Learning Algorithms towards AI. 16, 19

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: A cpu and gpu math compiler in python. In *Proc. 9th Python in Science Conf*, pages 1–7. 31

Bergstra, J., Breuleux, O., Lamblin, P., Pascanu, R., Delalleau, O., Desjardins, G., Goodfellow, I., Bergeron, A., Bengio, Y., and Kaelbling, P. (2011). Theano: Deep learning on gpus with python. 30

Bhatia, N. and Rana, C. (2015). Deep Learning Techniques and its Various Algorithms

and Techniques. *International Journal of Engineering Innovation and Research*, 4(5):707–710. 14

Chen, Z., Wang, J., He, H., and Huang, X. (2014). A fast deep learning system using GPU. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1552–1555. IEEE. 10

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537. 27

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297. 10

Dauphin, G. M. Y., Glorot, X., Rifai, S., Bengio, Y., Goodfellow, I., Lavoie, E., Muller, X., Desjardins, G., Warde-Farley, D., Vincent, P., Courville, A., and Bergstra, J. (2012). Unsupervised and transfer learning challenge: a deep learning approach. In Guyon, I., Dror, G., Lemaire, V., Taylor, G., and Silver, D., editors, *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, volume 27 of *Proceedings of Machine Learning Research*, pages 97–110, Bellevue, Washington, USA. PMLR. 20

Deng, L. (2014). Deep Learning: Methods and Applications. *Foundations and Trends® in Signal Processing*, 7(1):199–200. 11

Deng, L., Abdel-Hamid, O., and Yu, D. (2013). A deep convolutional neural network using heterogeneous pooling for trading acoustic invariance with phonetic confusion. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. 10

Devlin, J., Zbib, R., Huang, Z., Lamar, T., Schwartz, R., and Makhoul, J. (2014). Fast and robust neural network joint models for statistical machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1370–1380. 20

Edx (2015). Events in the tracking logs. 40

Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., and Thrun, S. (2017). Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542:115 EP –. 32, 35

Fernández, R. Introducción a r. 38

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. and Titterington, M., editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy. PMLR. 10

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`. 26

Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*. 20

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554. PMID: 16764513. 10

Jarrett, K., Kavukcuoglu, K., Ranzato, M. A., and Lecun, Y. What is the Best Multi-Stage Architecture for Object Recognition? 10

Johnson, R. and Zhang, T. (2014). Effective use of word order for text categorization with convolutional neural networks. *CoRR*, abs/1412.1058. 34, 35

Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). A convolutional neural network for modelling sentences. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*. 33, 35

Kim, Y. (2014). Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882. 34, 35

Kolosnjaji, B., Zarras, A., Webster, G., and Eckert, C. (2016). *Deep Learning for Classification of Malware System Call Sequences*, pages 137–149. Springer International Publishing, Cham. 33, 35

Le, Q. V. (2015). A Tutorial on Deep Learning Part 1: Nonlinear Classifiers and The Backpropagation Algorithm. 15

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551. 9

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998a). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324. 10

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998b). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324. 25

Lee, H., Grosse, R., Ranganath, R., and Ng, A. Y. (2009). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 609–616, New York, NY, USA. ACM. 14, 16, 18

Li, F.-F., Johnson, J., and Yeung, S. (2017). Lecture 1. 31, 32

Liou, C.-Y., Cheng, W.-C., Liou, J.-W., and Liou, D.-R. (2014). Autoencoder for words. *Neurocomputing*, 139(Supplement C):84 – 96. 10

McAuley, J. and Leskovec, J. (2013). Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 165–172. ACM. 27

Mo, D. (2012). A survey on deep learning: one small step toward ai. *Dept. Computer Science, Univ. of New Mexico, USA*. 15, 16, 18, 19

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814. 26, 27

Najafabadi, M. M., Villanustre, F., Khoshgoftaar, T. M., Seliya, N., Wald, R., and Muharemagic, E. (2015). Deep learning applications and challenges in big data analytics. *Journal of Big Data*, 2(1):1. 22

Ngiam, J., Chen, Z., Chia, D., Koh, P. W., Le, Q. V., and Ng, A. Y. (2010). Tiled convolutional neural networks. In Lafferty, J. D., Williams, C. K. I., Shawe-Taylor, J., Zemel, R. S., and Culotta, A., editors, *Advances in Neural Information Processing Systems 23*, pages 1279–1287. Curran Associates, Inc. 19

Nielsen, M. A. (2015). Neural Networks and Deep Learning. 25

Peralta, D., Triguero, I., García, S., Saeys, Y., Benitez, J. M., and Herrera, F. (2018). On the use of convolutional neural networks for robust classification of multiple fingerprint captures. *International Journal of Intelligent Systems*, 33(1):213–230. 22

Poonia, P., Jain, V. K., and Kumar, A. (2016). Deep Learning: Review. *Anil Kumar International Journal of Computer & Mathematical Sciences IJCMS ISSN*, 5(12):2347–8527. 13

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386. 9

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536. 9

Salakhutdinov, R. and Hinton, G. Deep Boltzmann Machines. 17

Shalev-Shwartz, S. and Singer, Y. (2005). A New Perspective on an Old Perceptron Algorithm. pages 264–278. 15

Sifium, M. S. (2017). Types of classification algorithms in machine learning. 22

Stamford, C. (2017). Gartner Identifies Three Megatrends That Will Drive Digital Business Into the Next Decade. 12, 13

Williams, D. and Hinton, G. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–538. 26

Yao, Y., Liang, H., Li, X., Zhang, J., and He, J. (2017). Sensing urban land-use patterns by integrating google tensorflow and scene-classification models. volume 42, pages 981–988. cited By 0. 33

Yoshua Bengio, Adrian-Horia Dediu, C. M.-V. R. M. B. T. e. (2013). *Statistical Language and Speech Processing: First International Conference, SLSP 2013, Tarragona, Spain, July 29-31, 2013. Proceedings*. Lecture Notes in Computer Science 7978. Springer-Verlag Berlin Heidelberg, 1 edition. 21, 33, 35

Yuan, L., Qu, Z., Zhao, Y., Zhang, H., and Nian, Q. (2017). A convolutional neural network based on tensorflow for face recognition. pages 525–529. cited By 0. 32, 35

Zhang, C. (2015). Deep Learning : Review & Discussion. 5(12):43–47. 11

ANEXOS

Instalación entorno Tensorflow

Algunos de los siguientes comandos requieren sudo (lo que significa súper usuario), para instalar pip.

```
sudo easy_install pip
```

El siguiente paso es crear un entorno virtual. Esto asegura que el código de TensorFlow no afectará a ninguno de los otros proyectos de Python que esté realizando.

```
sudo pip install --upgrade virtualenv
```

Para crear la carpeta en la que se instalará todo el proyecto, tan pronto como cree su entorno virtual.

```
virtualenv --system-site-packages ~/tensorflow
```

Tenga en cuenta que instalamos un entorno virtual en la carpeta, pero aún no lo hemos activado, para activarlo.

```
source tensorflow/bin/activate
```

Para desactivar el entorno virtual.

```
deactivate
```

Para utilizar Tensorflow se activará de nuevo.

```
source tensorflow/bin/activate
```

Posteriormente a la activación del entorno virtual, dentro de este vamos a instalar la librería de Tensorflow de deep learning herramienta principal en el proyecto desarrollado con todos sus requerimientos.

```
pip install --upgrade tensorflow
```

Para poder trabajar con el dataset se utilizó la librería Pandas de Python.

```
pip install pandas
```

Para la visualización del dataset, limpieza y depuración de datos, así como la prueba de algoritmos a través del lenguaje Python se usó el proyecto open source Jupyter.

```
pip install jupyter
```

Finalmente para empezar con el uso de la herramienta jupyter.

```
jupyter notebook
```

Código RNN utilizado para la selección de algoritmo más eficiente

Para la fase de implementación de algoritmo en la selección del mejor algoritmo se utilizan dos algoritmos con Redes Neuronales diferentes:

1. algoritmoRNN

La estructura del proyecto se observa en la Figura 5.1.

- train.py

```
import os
import sys
import json
import time
```

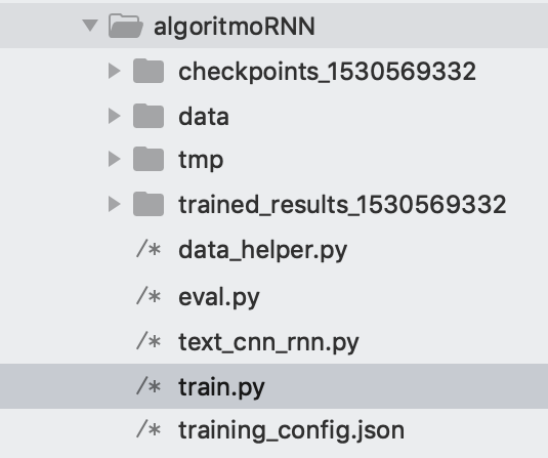


Figure 5.1: Estructura del algoritmo RNN

Fuente: El Autor

Elaborado por: El Autor

```
import shutil
import pickle
import logging
import data_helper
import numpy as np
import pandas as pd
import tensorflow as tf
from text_cnn_rnn import TextCNNRNN
from sklearn.model_selection import train_test_split

logging.getLogger().setLevel(logging.INFO)

def train_cnn_rnn():
  input_file = sys.argv[1]
  x_, y_, vocabulary, vocabulary_inv, df, labels =
   data_helper.load_data(input_file)
```

```
training_config = sys.argv[2]
params = json.loads(open(training_config).read())

# Assign a 300 dimension vector to each word
word_embeddings = data_helper.load_embeddings(vocabulary)
embedding_mat = [word_embeddings[word] for index, word in
 enumerate(vocabulary_inv)]
embedding_mat = np.array(embedding_mat, dtype = np.float32
 )

# Split the original dataset into train set and test set
x, x_test, y, y_test = train_test_split(x_, y_, test_size
 =0.1)

# Split the train set into train set and dev set
x_train, x_dev, y_train, y_dev = train_test_split(x, y,
 test_size=0.1)

logging.info('x_train: {}, x_dev: {}, x_test: {}'.format(
 len(x_train), len(x_dev), len(x_test)))
logging.info('y_train: {}, y_dev: {}, y_test: {}'.format(
 len(y_train), len(y_dev), len(y_test)))

# Create a directory, everything related to the training
 will be saved in this directory
timestamp = str(int(time.time()))
trained_dir = './trained_results_' + timestamp + '/'
if os.path.exists(trained_dir):
  shutil.rmtree(trained_dir)
```

```
os.makedirs(trained_dir)

graph = tf.Graph()
with graph.as_default():
  session_conf = tf.ConfigProto(allow_soft_placement=True,
  log_device_placement=False)
  sess = tf.Session(config=session_conf)
  with sess.as_default():
    cnn_rnn = TextCNNRNN(
      embedding_mat=embedding_mat,
      sequence_length=x_train.shape[1],
      num_classes = y_train.shape[1],
      non_static=params['non_static'],
      hidden_unit=params['hidden_unit'],
      max_pool_size=params['max_pool_size'],
      filter_sizes=map(int, params['filter_sizes'].split("
,")),
      num_filters = params['num_filters'],
      embedding_size = params['embedding_dim'],
      l2_reg_lambda = params['l2_reg_lambda'])

    global_step = tf.Variable(0, name='global_step',
trainable=False)
    optimizer = tf.train.RMSPropOptimizer(1e-3, decay=0.9)
    grads_and_vars = optimizer.compute_gradients(cnn_rnn.
loss)
    train_op = optimizer.apply_gradients(grads_and_vars,
global_step=global_step)
```

```
    grad_summaries = []
    for g, v in grads_and_vars:
      if g is not None:
        grad_hist_summary = tf.summary.histogram("{}/grad/
hist".format(v.name), g)
        sparsity_summary = tf.summary.scalar("{}/grad/
sparsity".format(v.name), tf.nn.zero_fraction(g))
        grad_summaries.append(grad_hist_summary)
        grad_summaries.append(sparsity_summary)
    grad_summaries_merged = tf.summary.merge(
grad_summaries)

    acc_summary = tf.summary.scalar("accuracy", cnn_rnn.
accuracy)
    loss_summary = tf.summary.scalar("loss", cnn_rnn.loss)

    train_summary_op = tf.summary.merge([loss_summary,
acc_summary, grad_summaries_merged])
    dev_summary_op = tf.summary.merge([loss_summary,
acc_summary])

    train_summary_writer = tf.summary.FileWriter("tmp/
train", sess.graph)
    dev_summary_writer = tf.summary.FileWriter("tmp/dev",
sess.graph)

    # Checkpoint files will be saved in this directory
during training
    checkpoint_dir = './checkpoints_' + timestamp + '/'
```

```
    if os.path.exists(checkpoint_dir):
      shutil.rmtree(checkpoint_dir)
    os.makedirs(checkpoint_dir)
    checkpoint_prefix = os.path.join(checkpoint_dir, '
model')

    def real_len(batches):
      return [np.ceil(np.argmin(batch + [0]) * 1.0 /
params['max_pool_size']) for batch in batches]

    def train_step(x_batch, y_batch):
      feed_dict = {
        cnn_rnn.input_x: x_batch,
        cnn_rnn.input_y: y_batch,
        cnn_rnn.dropout_keep_prob: params['
dropout_keep_prob'],
        cnn_rnn.batch_size: len(x_batch),
        cnn_rnn.pad: np.zeros([len(x_batch), 1, params['
embedding_dim'], 1]),
        cnn_rnn.real_len: real_len(x_batch),
      }
      _, step, summaries, loss, accuracy = sess.run([
train_op, global_step, train_summary_op, cnn_rnn.loss,
cnn_rnn.accuracy], feed_dict)
      train_summary_writer.add_summary(summaries, step)

    def dev_step(x_batch, y_batch, writer=None):
      feed_dict = {
        cnn_rnn.input_x: x_batch,
```

```
        cnn_rnn.input_y: y_batch,
        cnn_rnn.dropout_keep_prob: 1.0,
        cnn_rnn.batch_size: len(x_batch),
        cnn_rnn.pad: np.zeros([len(x_batch), 1, params['
embedding_dim'], 1]),
        cnn_rnn.real_len: real_len(x_batch),
      }
      step, summaries, loss, accuracy, num_correct,
predictions = sess.run(
        [global_step, dev_summary_op, cnn_rnn.loss,
cnn_rnn.accuracy, cnn_rnn.num_correct, cnn_rnn.
predictions], feed_dict)
      if writer:
        writer.add_summary(summaries, step)
      return accuracy, loss, num_correct, predictions

    saver = tf.train.Saver()
    sess.run(tf.global_variables_initializer())

    # Training starts here
    train_batches = data_helper.batch_iter(list(zip(
x_train, y_train)), params['batch_size'], params['
num_epochs'])
    best_accuracy, best_at_step = 0, 0

    # Train the model with x_train and y_train
    for train_batch in train_batches:
      x_train_batch, y_train_batch = zip(*train_batch)
      train_step(x_train_batch, y_train_batch)
```

```
      current_step = tf.train.global_step(sess,
global_step)

      # Evaluate the model with x_dev and y_dev
      if current_step % params['evaluate_every'] == 0:
        dev_batches = data_helper.batch_iter(list(zip(
x_dev, y_dev)), params['batch_size'], 1)
        dev_step(x_train_batch, y_train_batch, writer=
dev_summary_writer)
        total_dev_correct = 0
        for dev_batch in dev_batches:
          x_dev_batch, y_dev_batch = zip(*dev_batch)
          acc, loss, num_dev_correct, predictions =
dev_step(x_dev_batch, y_dev_batch)
          total_dev_correct += num_dev_correct
        accuracy = float(total_dev_correct) / len(y_dev)
        logging.info('Accuracy on dev set: {}'.format(
accuracy))

        if accuracy >= best_accuracy:
          best_accuracy, best_at_step = accuracy,
current_step
          path = saver.save(sess, checkpoint_prefix,
global_step=current_step)
          logging.critical('Saved model {} at step {}'.
format(path, best_at_step))
          logging.critical('Best accuracy {} at step {}'.
format(best_accuracy, best_at_step))
    logging.critical('Training is complete, testing the
```

```
 best model on x_test and y_test')

    # Save the model files to trained_dir. predict.py
 needs trained model files.
    saver.save(sess, trained_dir + "best_model.ckpt")

    # Evaluate x_test and y_test
    saver.restore(sess, checkpoint_prefix + '-' + str(
 best_at_step))
    test_batches = data_helper.batch_iter(list(zip(x_test,
  y_test)), params['batch_size'], 1, shuffle=False)
    total_test_correct = 0
    for test_batch in test_batches:
      x_test_batch, y_test_batch = zip(*test_batch)
      acc, loss, num_test_correct, predictions = dev_step(
 x_test_batch, y_test_batch)
      total_test_correct += int(num_test_correct)
    logging.critical('Accuracy on test set: {}'.format(
 float(total_test_correct) / len(y_test)))

# Save trained parameters and files since predict.py needs
  them
with open(trained_dir + 'words_index.json', 'w') as
 outfile:
  json.dump(vocabulary, outfile, indent=4, ensure_ascii=
 False)
with open(trained_dir + 'embeddings.pickle', 'wb') as
 outfile:
  pickle.dump(embedding_mat, outfile, pickle.
```

```
   HIGHEST_PROTOCOL)
  with open(trained_dir + 'labels.json', 'w') as outfile:
    json.dump(labels, outfile, indent=4, ensure_ascii=False)

  params['sequence_length'] = x_train.shape[1]
  with open(trained_dir + 'trained_parameters.json', 'w') as
    outfile:
    json.dump(params, outfile, indent=4, sort_keys=True,
   ensure_ascii=False)

if __name__ == '__main__':
  train_cnn_rnn()
```

Código CNN utilizado para la implemetación del algoritmo

En la fase de implementación, pruebas y análisis de resultados el algoritmo con el que se realiza la clasificación utiliza la Red Neuronal Convolucional:

1. algoritmoCNN

La estructura del proyecto se observa en la Figura 5.2.

- train.py

```
import tensorflow as tf
import numpy as np
import os
import time
```

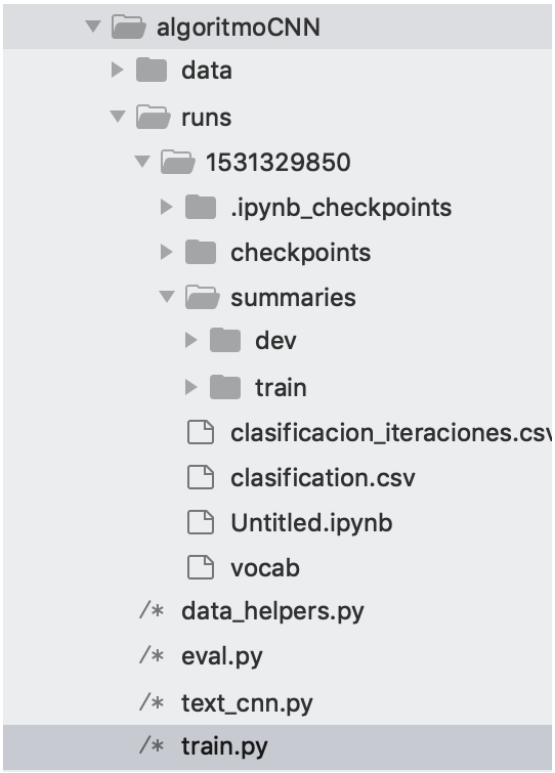


Figure 5.2: Estructura del algoritmo CNN
Fuente: El Autor
Elaborado por: El Autor

```
import datetime
import data_helpers
from text_cnn import TextCNN
from tensorflow.contrib import learn

# Data loading params
tf.flags.DEFINE_float("dev_sample_percentage", .1, "
   Percentage of the training data to use for validation")
tf.flags.DEFINE_string("positive_data_file", "./data/rt-
   polaritydata/videos.vid", "Data source for the positive
   data.")
```

```
tf.flags.DEFINE_string("negative_data_file", "./data/rt-
   polaritydata/cursos.cur", "Data source for the negative
   data.")

# Model Hyperparameters
tf.flags.DEFINE_integer("embedding_dim", 128, "
   Dimensionality of character embedding (default: 128)")
tf.flags.DEFINE_string("filter_sizes", "3,4,5", "Comma-
   separated filter sizes (default: '3,4,5')")
tf.flags.DEFINE_integer("num_filters", 128, "Number of
   filters per filter size (default: 128)")
tf.flags.DEFINE_float("dropout_keep_prob", 0.5, "Dropout
   keep probability (default: 0.5)")
tf.flags.DEFINE_float("l2_reg_lambda", 0.0, "L2
   regularization lambda (default: 0.0)")

# Training parameters
tf.flags.DEFINE_integer("batch_size", 64, "Batch Size (
   default: 64)")
tf.flags.DEFINE_integer("num_epochs", 200, "Number of
   training epochs (default: 200)")
tf.flags.DEFINE_integer("evaluate_every", 100, "Evaluate
   model on dev set after this many steps (default: 100)")
tf.flags.DEFINE_integer("checkpoint_every", 100, "Save model
    after this many steps (default: 100)")
tf.flags.DEFINE_integer("num_checkpoints", 5, "Number of
   checkpoints to store (default: 5)")
# Misc Parameters
tf.flags.DEFINE_boolean("allow_soft_placement", True, "Allow
```

```
    device soft device placement")
tf.flags.DEFINE_boolean("log_device_placement", False, "Log
   placement of ops on devices")

FLAGS = tf.flags.FLAGS

def preprocess():
    # Load data
    print("Loading data...")
    x_text, y = data_helpers.load_data_and_labels(FLAGS.
   positive_data_file, FLAGS.negative_data_file)

    # Build vocabulary
    max_document_length = max([len(x.split(" ")) for x in
   x_text])
    vocab_processor = learn.preprocessing.
   VocabularyProcessor(max_document_length)
    x = np.array(list(vocab_processor.fit_transform(x_text))
   )

    # Randomly shuffle data
    np.random.seed(10)
    shuffle_indices = np.random.permutation(np.arange(len(y)
   ))
    x_shuffled = x[shuffle_indices]
    y_shuffled = y[shuffle_indices]

    # Split train/test set
    # TODO: This is very crude, should use cross-validation
```

```
    dev_sample_index = -1 * int(FLAGS.dev_sample_percentage
* float(len(y)))
    x_train, x_dev = x_shuffled[:dev_sample_index],
x_shuffled[dev_sample_index:]
    y_train, y_dev = y_shuffled[:dev_sample_index],
y_shuffled[dev_sample_index:]

    del x, y, x_shuffled, y_shuffled

    print("Vocabulary Size: {:d}".format(len(vocab_processor
.vocabulary_)))
    print("Train/Dev split: {:d}/{:d}".format(len(y_train),
len(y_dev)))
    return x_train, y_train, vocab_processor, x_dev, y_dev

def train(x_train, y_train, vocab_processor, x_dev, y_dev):
    # Training

    with tf.Graph().as_default():
        session_conf = tf.ConfigProto(
          allow_soft_placement=FLAGS.allow_soft_placement,
          log_device_placement=FLAGS.log_device_placement)
        sess = tf.Session(config=session_conf)
        with sess.as_default():
            cnn = TextCNN(
                sequence_length=x_train.shape[1],
                num_classes=y_train.shape[1],
                vocab_size=len(vocab_processor.vocabulary_),
                embedding_size=FLAGS.embedding_dim,
```

```
                filter_sizes=list(map(int, FLAGS.
filter_sizes.split(","))),
                num_filters=FLAGS.num_filters,
                l2_reg_lambda=FLAGS.l2_reg_lambda)

            # Define Training procedure
            global_step = tf.Variable(0, name="global_step",
 trainable=False)
            optimizer = tf.train.AdamOptimizer(1e-3)
            grads_and_vars = optimizer.compute_gradients(cnn
.loss)
            train_op = optimizer.apply_gradients(
grads_and_vars, global_step=global_step)

            # Keep track of gradient values and sparsity
            grad_summaries = []
            for g, v in grads_and_vars:
                if g is not None:
                    grad_hist_summary = tf.summary.histogram
("{}/grad/hist".format(v.name), g)
                    sparsity_summary = tf.summary.scalar("
{}/grad/sparsity".format(v.name), tf.nn.zero_fraction(g))
                    grad_summaries.append(grad_hist_summary)
                    grad_summaries.append(sparsity_summary)
            grad_summaries_merged = tf.summary.merge(
grad_summaries)

            # Output directory for models and summaries
            timestamp = str(int(time.time()))
```

```
            out_dir = os.path.abspath(os.path.join(os.path.
curdir, "runs", timestamp))
            print("Writing to {}\n".format(out_dir))

            # Summaries for loss and accuracy
            loss_summary = tf.summary.scalar("loss", cnn.
loss)
            acc_summary = tf.summary.scalar("accuracy", cnn.
accuracy)

            # Train Summaries
            train_summary_op = tf.summary.merge([
loss_summary, acc_summary, grad_summaries_merged])
            train_summary_dir = os.path.join(out_dir, "
summaries", "train")
            train_summary_writer = tf.summary.FileWriter(
train_summary_dir, sess.graph)

            # Dev summaries
            dev_summary_op = tf.summary.merge([loss_summary,
 acc_summary])
            dev_summary_dir = os.path.join(out_dir, "
summaries", "dev")
            dev_summary_writer = tf.summary.FileWriter(
dev_summary_dir, sess.graph)

            # Checkpoint directory. Tensorflow assumes this
directory already exists so we need to create it
            checkpoint_dir = os.path.abspath(os.path.join(
```

```
out_dir, "checkpoints"))
          checkpoint_prefix = os.path.join(checkpoint_dir,
 "model")
          if not os.path.exists(checkpoint_dir):
              os.makedirs(checkpoint_dir)
          saver = tf.train.Saver(tf.global_variables(),
max_to_keep=FLAGS.num_checkpoints)

          # Write vocabulary
          vocab_processor.save(os.path.join(out_dir, "
vocab"))

          # Initialize all variables
          sess.run(tf.global_variables_initializer())

          def train_step(x_batch, y_batch):

              # A single training step
              feed_dict = {
                cnn.input_x: x_batch,
                cnn.input_y: y_batch,
                cnn.dropout_keep_prob: FLAGS.
dropout_keep_prob
              }
              _, step, summaries, loss, accuracy = sess.
run(
                  [train_op, global_step, train_summary_op
, cnn.loss, cnn.accuracy],
                  feed_dict)
```

```
            time_str = datetime.datetime.now().isoformat
()
            print("{}: step {}, loss {:g}, acc {:g}".
format(time_str, step, loss, accuracy))
            train_summary_writer.add_summary(summaries,
step)

        def dev_step(x_batch, y_batch, writer=None):

            # Evaluates model on a dev set
            feed_dict = {
              cnn.input_x: x_batch,
              cnn.input_y: y_batch,
              cnn.dropout_keep_prob: 1.0
            }
            step, summaries, loss, accuracy = sess.run(
                [global_step, dev_summary_op, cnn.loss,
cnn.accuracy],
                feed_dict)
            time_str = datetime.datetime.now().isoformat
()
            print("{}: step {}, loss {:g}, acc {:g}".
format(time_str, step, loss, accuracy))
            if writer:
                writer.add_summary(summaries, step)

        # Generate batches
        batches = data_helpers.batch_iter(
            list(zip(x_train, y_train)), FLAGS.
```

```
    batch_size, FLAGS.num_epochs)
            # Training loop. For each batch...
            for batch in batches:
                x_batch, y_batch = zip(*batch)
                train_step(x_batch, y_batch)
                current_step = tf.train.global_step(sess,
    global_step)
                if current_step % FLAGS.evaluate_every == 0:
                    print("\nEvaluation:")
                    dev_step(x_dev, y_dev, writer=
    dev_summary_writer)
                    print("")
                if current_step % FLAGS.checkpoint_every ==
    0:
                    path = saver.save(sess,
    checkpoint_prefix, global_step=current_step)
                    print("Saved model checkpoint to {}\n".
    format(path))

def main(argv=None):
    x_train, y_train, vocab_processor, x_dev, y_dev =
    preprocess()
    train(x_train, y_train, vocab_processor, x_dev, y_dev)

if __name__ == '__main__':
    tf.app.run()
```

- eval.py

```
import tensorflow as tf
import numpy as np
import os
import time
import datetime
import data_helpers
from text_cnn import TextCNN
from tensorflow.contrib import learn
import csv

# Parameters

# Data Parameters
tf.flags.DEFINE_string("positive_data_file", "./data/rt-
    polaritydata/videos.vid", "Data source for the positive
    data.")
tf.flags.DEFINE_string("negative_data_file", "./data/rt-
    polaritydata/cursos.cur", "Data source for the negative
    data.")

# Eval Parameters
tf.flags.DEFINE_integer("batch_size", 64, "Batch Size (
    default: 64)")
tf.flags.DEFINE_string("checkpoint_dir", "", "Checkpoint
    directory from training run")
tf.flags.DEFINE_boolean("eval_train", False, "Evaluate on
    all training data")
```

```
# Misc Parameters
tf.flags.DEFINE_boolean("allow_soft_placement", True, "Allow
    device soft device placement")
tf.flags.DEFINE_boolean("log_device_placement", False, "Log
   placement of ops on devices")

FLAGS = tf.flags.FLAGS
FLAGS._parse_flags()
print("\nParameters:")
for attr, value in sorted(FLAGS.__flags.items()):
    print("{}={}".format(attr.upper(), value))
print("")

# CHANGE THIS: Load data. Load your own data here
if FLAGS.eval_train:
    x_raw, y_test = data_helpers.load_data_and_labels("./
   data/rt-polaritydata/final_small.cur", "./data/rt-
   polaritydata/final.cur")
    y_test = np.argmax(y_test, axis=1)
else:
    x_raw = ["a masterpiece four years in the making", "
   everything is off."]
    y_test = [1, 0]

# Map data into vocabulary
vocab_path = os.path.join(FLAGS.checkpoint_dir, "..", "vocab
   ")
vocab_processor = learn.preprocessing.VocabularyProcessor.
   restore(vocab_path)
```

```
x_test = np.array(list(vocab_processor.transform(x_raw)))

print("\nEvaluating...\n")

# Evaluation
checkpoint_file = tf.train.latest_checkpoint(FLAGS.
   checkpoint_dir)
graph = tf.Graph()
with graph.as_default():
    session_conf = tf.ConfigProto(
      allow_soft_placement=FLAGS.allow_soft_placement,
      log_device_placement=FLAGS.log_device_placement)
    sess = tf.Session(config=session_conf)
    with sess.as_default():
        # Load the saved meta graph and restore variables
        saver = tf.train.import_meta_graph("{}.meta".format(
   checkpoint_file))
        saver.restore(sess, checkpoint_file)

        # Get the placeholders from the graph by name
        input_x = graph.get_operation_by_name("input_x").
   outputs[0]
        # input_y = graph.get_operation_by_name("input_y").
   outputs[0]
        dropout_keep_prob = graph.get_operation_by_name("
   dropout_keep_prob").outputs[0]

        # Tensors we want to evaluate
        predictions = graph.get_operation_by_name("output/
```

```
    predictions").outputs[0]

        # Generate batches for one epoch
        batches = data_helpers.batch_iter(list(x_test),
    FLAGS.batch_size, 1, shuffle=False)

        # Collect the predictions here
        all_predictions = []

        for x_test_batch in batches:
            batch_predictions = sess.run(predictions, {
    input_x: x_test_batch, dropout_keep_prob: 1.0})
            all_predictions = np.concatenate([
    all_predictions, batch_predictions])

# Print accuracy if y_test is defined
if y_test is not None:
    correct_predictions = float(sum(all_predictions ==
    y_test))
    print("Total number of test examples: {}".format(len(
    y_test)))
    print("Accuracy: {:g}".format(correct_predictions/float(
    len(y_test))))

# Save the evaluation to a csv
predictions_human_readable = np.column_stack((np.array(x_raw
    ), all_predictions))
out_path = os.path.join(FLAGS.checkpoint_dir, "..", "
    clasification.csv")
```

```
print("Saving evaluation to {0}".format(out_path))
with open(out_path, 'w') as f:
    csv.writer(f).writerows(predictions_human_readable)
```

Librerías utilizadas en el algoritmo de clasificación CNN

Las librerias utilizadas se utilizan en el algoritmo para que la Red Neuronal Convolucional se adapte a trabajar con tipo de datos textuales en este caso los logs utilizados en el proyecto y se transforman en arreglos de números para que la Red Neuronal junto a Tensorflow logren la clasificación:

- text_cnn.py

```
import tensorflow as tf
import numpy as np

class TextCNN(object):
    def __init__(
      self, sequence_length, num_classes, vocab_size,
      embedding_size, filter_sizes, num_filters,
   l2_reg_lambda=0.0):

        # Placeholders for input, output and dropout
        self.input_x = tf.placeholder(tf.int32, [None,
   sequence_length], name="input_x")
        self.input_y = tf.placeholder(tf.float32, [None,
   num_classes], name="input_y")
        self.dropout_keep_prob = tf.placeholder(tf.float32,
   name="dropout_keep_prob")
```

```
        # Keeping track of l2 regularization loss (optional)
        l2_loss = tf.constant(0.0)

        # Embedding layer
        with tf.device('/cpu:0'), tf.name_scope("embedding")
:
            self.W = tf.Variable(
                tf.random_uniform([vocab_size,
embedding_size], -1.0, 1.0),
                name="W")
            self.embedded_chars = tf.nn.embedding_lookup(
self.W, self.input_x)
            self.embedded_chars_expanded = tf.expand_dims(
self.embedded_chars, -1)

        # Create a convolution + maxpool layer for each
filter size
        pooled_outputs = []
        for i, filter_size in enumerate(filter_sizes):
            with tf.name_scope("conv-maxpool-%s" %
filter_size):
                # Convolution Layer
                filter_shape = [filter_size, embedding_size,
 1, num_filters]
                W = tf.Variable(tf.truncated_normal(
filter_shape, stddev=0.1), name="W")
                b = tf.Variable(tf.constant(0.1, shape=[
num_filters]), name="b")
```

```
                conv = tf.nn.conv2d(
                    self.embedded_chars_expanded,
                    W,
                    strides=[1, 1, 1, 1],
                    padding="VALID",
                    name="conv")
                # Apply nonlinearity
                h = tf.nn.relu(tf.nn.bias_add(conv, b), name
="relu")
                # Maxpooling over the outputs
                pooled = tf.nn.max_pool(
                    h,
                    ksize=[1, sequence_length - filter_size
+ 1, 1, 1],
                    strides=[1, 1, 1, 1],
                    padding='VALID',
                    name="pool")
                pooled_outputs.append(pooled)

        # Combine all the pooled features
        num_filters_total = num_filters * len(filter_sizes)
        self.h_pool = tf.concat(pooled_outputs, 3)
        self.h_pool_flat = tf.reshape(self.h_pool, [-1,
num_filters_total])

        # Add dropout
        with tf.name_scope("dropout"):
            self.h_drop = tf.nn.dropout(self.h_pool_flat,
self.dropout_keep_prob)
```

```
        # Final (unnormalized) scores and predictions
        with tf.name_scope("output"):
            W = tf.get_variable(
                "W",
                shape=[num_filters_total, num_classes],
                initializer=tf.contrib.layers.
xavier_initializer())
            b = tf.Variable(tf.constant(0.1, shape=[
num_classes]), name="b")
            l2_loss += tf.nn.l2_loss(W)
            l2_loss += tf.nn.l2_loss(b)
            self.scores = tf.nn.xw_plus_b(self.h_drop, W, b,
 name="scores")
            self.predictions = tf.argmax(self.scores, 1,
name="predictions")

        # Calculate mean cross-entropy loss
        with tf.name_scope("loss"):
            losses = tf.nn.softmax_cross_entropy_with_logits
(logits=self.scores, labels=self.input_y)
            self.loss = tf.reduce_mean(losses) +
l2_reg_lambda * l2_loss

        # Accuracy
        with tf.name_scope("accuracy"):
            correct_predictions = tf.equal(self.predictions,
 tf.argmax(self.input_y, 1))
            self.accuracy = tf.reduce_mean(tf.cast(
```

```
   correct_predictions, "float"), name="accuracy")
```

- data_helpers.py

```
import numpy as np
import re
import itertools
from collections import Counter


def clean_str(string):

    # Tokenization/string cleaning for all datasets except
   for SST.
    string = re.sub(r"[^A-Za-z0-9(),!?\'\`]", " ", string)
    string = re.sub(r"\'s", " \'s", string)
    string = re.sub(r"\'ve", " \'ve", string)
    string = re.sub(r"n\'t", " n\'t", string)
    string = re.sub(r"\'re", " \'re", string)
    string = re.sub(r"\'d", " \'d", string)
    string = re.sub(r"\'ll", " \'ll", string)
    string = re.sub(r",", " , ", string)
    string = re.sub(r"!", " ! ", string)
    string = re.sub(r"\(", " \( ", string)
    string = re.sub(r"\)", " \) ", string)
    string = re.sub(r"\?", " \? ", string)
    string = re.sub(r"\s{2,}", " ", string)
    return string.strip().lower()
```

```
def load_data_and_labels(positive_data_file,
   negative_data_file):

    # Load data from files
    positive_examples = list(open(positive_data_file, "r").
   readlines())
    positive_examples = [s.strip() for s in
   positive_examples]
    negative_examples = list(open(negative_data_file, "r").
   readlines())
    negative_examples = [s.strip() for s in
   negative_examples]
    # Split by words
    x_text = positive_examples + negative_examples
    x_text = [clean_str(sent) for sent in x_text]
    # Generate labels
    positive_labels = [[0, 1] for _ in positive_examples]
    negative_labels = [[1, 0] for _ in negative_examples]
    y = np.concatenate([positive_labels, negative_labels],
   0)
    return [x_text, y]


def batch_iter(data, batch_size, num_epochs, shuffle=True):

    #Generates a batch iterator for a dataset.
    data = np.array(data)
```

```
 data_size = len(data)
 num_batches_per_epoch = int((len(data)-1)/batch_size) +
1
 for epoch in range(num_epochs):
     # Shuffle the data at each epoch
     if shuffle:
         shuffle_indices = np.random.permutation(np.
arange(data_size))
         shuffled_data = data[shuffle_indices]
     else:
         shuffled_data = data
     for batch_num in range(num_batches_per_epoch):
         start_index = batch_num * batch_size
         end_index = min((batch_num + 1) * batch_size,
data_size)
         yield shuffled_data[start_index:end_index]
```

Parametros mostrados en consola sobre la evaluación del algoritmo de clasificación

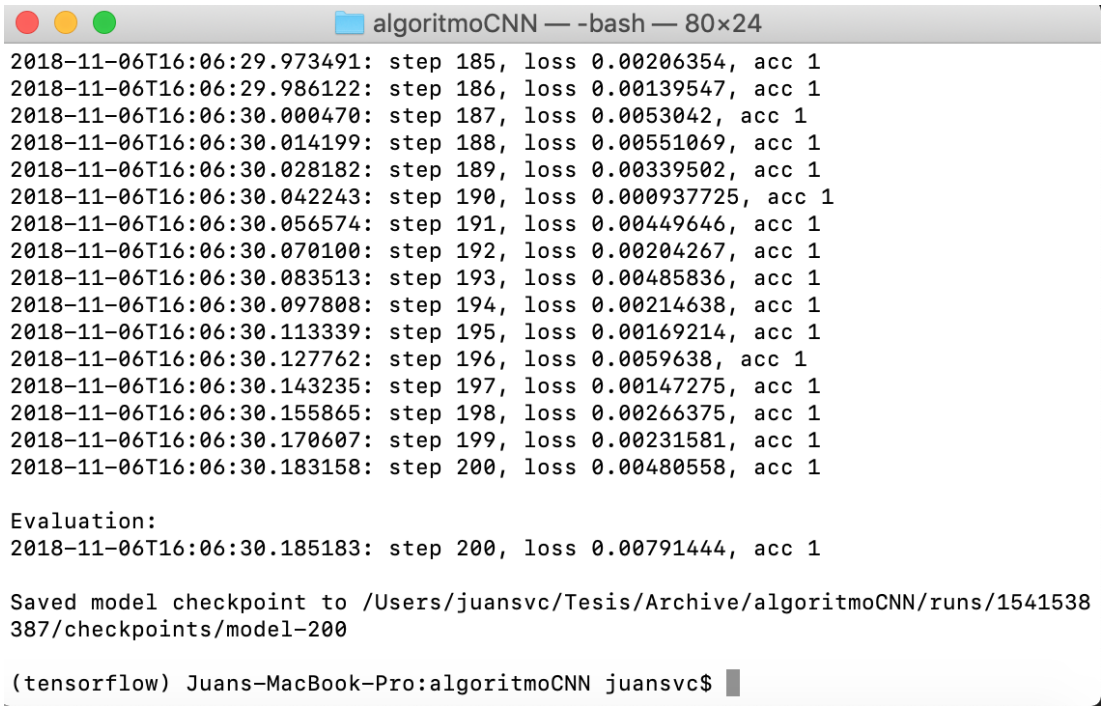


Figure 5.3: Iteraciones del algoritmos de clasificación
Fuente: El Autor
Elaborado por: El Autor

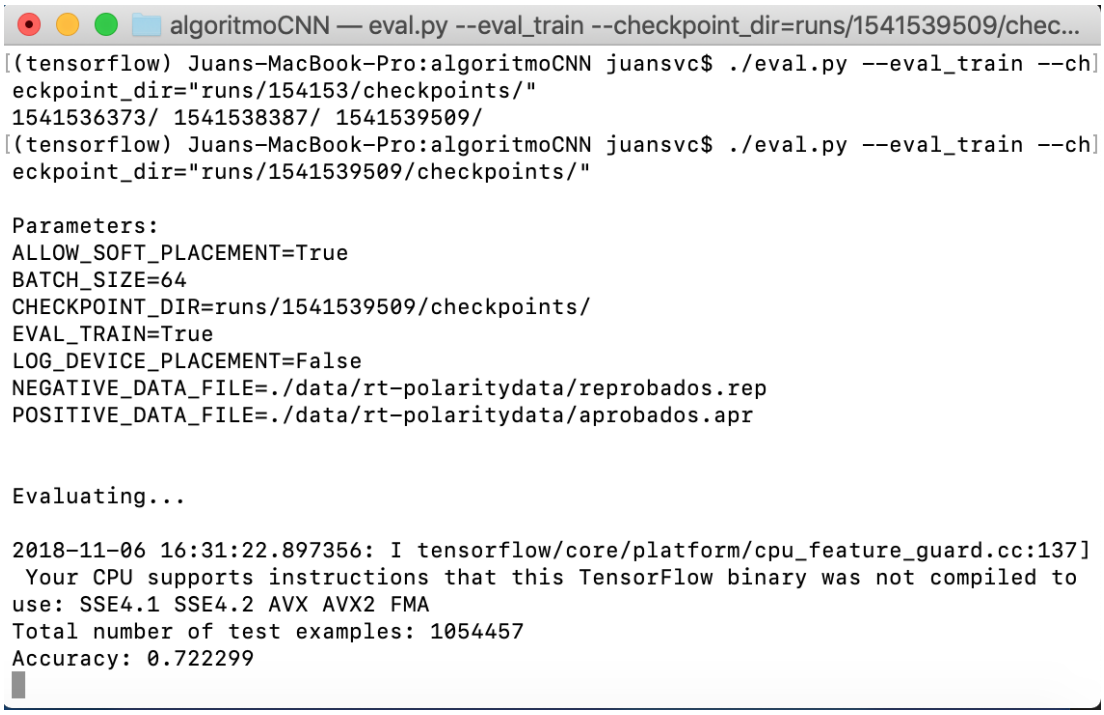


Figure 5.4: Parametros y resultado del algoritmos de clasificación
Fuente: El Autor
Elaborado por: El Autor