Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Detección de opiniones fraudulentas empleando Autoencoders

Deceitful opinion detection using Autoencoders

Vlad-Alexandru Comănescu

La Laguna, 12 de marzo de 2021

D. **Dagoberto Castellanos Nieves**, con N.I.F. 79.234.766-L profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

C E R T I F I C A (N)

Que la presente memoria titulada:

"*Detección de opiniones fraudulentas empleando Autoencoders*"

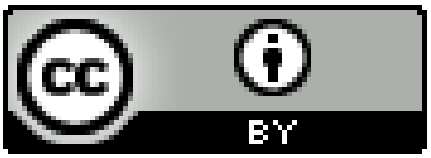
ha sido realizada bajo su dirección por D. **Vlad-Alexandru Comănescu**, con N.I.F. Y-20.539.58-A.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 12 de marzo de 2021

Agradecimientos

Le agradezco, en primer lugar al profesor D. **Dagoberto Castellanos Nieves**, por todo su esfuerzo y tiempo empleado en ayudarme a la realización de este trabajo.

También me gustaría agradecer a mi maravillosa familia y a toda la gente que me ha apoyado a lo largo de mi vida.



Resumen

Mediante este trabajo se propone un modelo de aprendizaje profundo para la detección de una problemática, la detección de valores atípicos de un conjunto de datos, basándose en la técnica no supervisada llamada Autoencoder.

Con el continuo crecimiento del comercio electrónico(e-commerce), el uso de la web para compartir experiencias sobre distintos servicios o productos se ha convertido en un factor determinante para las personas. Cada vez hay más usuarios que recurren a las opiniones de los demás para la compra de un bien o un servicio.

Sin embargo, esto también ha resultado en que haya personas que de forma consciente publiquen opiniones engañosas, tanto positivas para impulsar la venta de un bien o negativas para perjudicar a cierto servicio. Esto ha llevado a que el producto afectado tenga una gran dificultad en recuperar la confianza de los demás y seguir siendo rentable.

En la actualidad, el fraude online, incluido el uso de opiniones engañosas, es un fenómeno común impulsado por los beneficios.

Palabras clave: deep learning, redes neuronales, autoencoder, opiniones, detección de anomalías

Abstract

Through this project, a deep learning model is proposed for the detection of a well known problem, the detection of atypical values of a dataset, based on the unsupervised technique called Autoencoder.

With the continuous growth of electronic commerce (e-commerce), the use of the web to share experiences on different services or products has become a decisive factor for people. More and more users are turning to the opinions of others to buy a good or a service.

Nevertheless this has also resulted in people who consciously post misleading opinions, both positive to boost the sales of a good or negative to harm a certain service. This has made it difficult for the affected product to regain the trust of others and remain profitable.

Today, online fraud, including the use of misleading reviews, is a common phenomenon driven by benefits.

Keywords: deep learning, neural networks, autoencoder, anomaly detection, opinions

Índice general

1. Introducción **1**

1.1. Introducción 1

1.2. Antecedentes y estado actual del tema 1

2. Tecnologías empleadas y metodología propuesta **3**

2.1. Deep Learning 3

2.2. Redes neuronales recurrentes 4

2.3. Detección de anomalías 8

2.4. Técnicas para la detección de anomalías 9

2.5. Arquitectura del Autoencoder 11

2.6. Aplicaciones del Autoencoder 12

2.7. Metodología 13

3. Selección del conjunto de datos **15**

3.1. Búsqueda de diferentes conjuntos de datos 15

3.2. Criterios para la elección del conjunto de datos 15

3.3. Análisis de los datasets 17

3.4. Análisis del *Deceptive Opinion Spam Corpus* 18

4. Desarrollo del modelo de detección de opiniones fraudulentas **21**

4.1. Exploración de los datos 22

4.2. Preprocesamiento del *Deceptive Opinion Spam Corpus* 26

4.2.1. Limpieza de los datos 27

4.2.2. Reducción de los datos 28

4.2.3. Transformación de los datos 28

4.2.4. Limpieza del texto 29

4.2.5. Transformación del texto 30

4.3. Selección, modelización y entreno de modelos de detección de opiniones fraudulentas 31

4.3.1. Primer modelo 31

4.3.2. Segundo y tercer modelo 33

4.4. Métricas 34

4.5. Experimentación y resultados 37

5. Conclusiones y líneas futuras **43**

5.1. Conclusiones 43

5.2. Líneas futuras 44

6. Conclusions and future improvements **45**

6.1. Conclusions 45

6.2. Future improvements 46

7. Presupuesto **47**

7.1. Presupuesto de recursos humanos 47

7.2. Presupuesto material 48

7.3. Presupuesto para la implementación del proyecto 48

7.4. Presupuesto final 49

A. Métodos y tecnologías complementarias **50**

A.1. Uso de la librería WordCloud para la exploración de los datos 50

A.2. Lematización y Stemming 51

A.3. Doc2Vec y sus principales técnicas de conversión 51

A.4. Principales parámetros del método Doc2Vec de la librería Gensim . 53

A.5. Hiperparámetros del modelo utilizando Keras Tuner 54

Índice de Figuras

2.1. Estructura de una red neuronal recurrente 5
2.2. Ciclo de Keras 7
2.3. a) Análisis de imágenes [49] b) Medical [38] c) Redes [20] d) Detección de ánomalias en Big-data [26] 11
2.4. Autoencoder 12
2.5. Procedimiento general que se va a seguir para cumplir con el objetivo general 13

4.1. Proceso de aprendizaje automático 21
4.2. Distribución de las opiniones por fuente 22
4.3. Distribución de las opiniones por fuente 23
4.4. Distribución de la polaridad de las opiniones por fuente 24
4.5. Distribución de las opiniones sinceras/engañosas por fuente 24
4.6. Relación entre el tamaño de la opinión y la polaridad de la misma . 25
4.7. Relación entre el tamaño de la opinión y la clase engaño 26
4.8. Relación entre el tamaño de la opinión y la fuente 26
4.9. Valores duplicados del dataset 28
4.10 Ejemplo de un TaggedDocument 30
4.11 Diagrama general de la Matriz de Confusión 35
4.12 Diagrama de las curvas ROC para los diferentes modelos definidos . 38
4.13 Diagrama de las curvas Precision-Recall para los diferentes modelos definidos (Clase Negativa) 40
4.14 Error de reconstrucción para las diferentes clases del modelo 41
4.15 Matriz de confusión del segundo modelo LSTM definido 42

A.1. WordCloud con las palabras más frecuentes del dataset 50
A.2. WordCloud de opiniones sinceras 51
A.3. WordCloud de opiniones fraudulentas 51
A.4. Ejemplo del funcionamiento del algoritmo PV-DM 52
A.5. Ejemplo del funcionamiento del algoritmo PV-DBOW 53

Índice de Tablas

2.1. Comparación entre las ventajas y desventajas de las RNNs 6
2.2. Arquitectura propuesta para la detección de anomalías según el tipo de datos de entrada 10

3.1. Comparación de los diferentes datasets encontrados 16
3.2. Muestra del dataset Deceptive Opinion Spam Corpus 19

4.1. Métodos empleados para la evaluación del modelo 37
4.2. Características del ámbito empleado para el desarrollo del trabajo . . 38
4.3. Descripción de los diferentes modelos definidos 39
4.4. Métricas de los diferentes modelos definidos para ciertos umbrales . 40

7.1. Presupuesto Personal 47
7.2. Presupuesto de los materiales necesarios para el desarrollo del proyecto 48
7.3. Presupuesto de los materiales necesarios para el supuesto caso . . . 48
7.4. Presupuesto de los materiales necesarios para el supuesto caso haciendo uso del Google Cloud 48
7.5. Presupuesto Final 49

Capítulo 1

Introducción

1.1. Introducción

Mediante este trabajo se propone un modelo de aprendizaje profundo para la detección de una problemática basándose en la técnica no supervisada llamada Autoencoder.

Nuestro objetivo principal consiste, en la creación de un modelo basado en redes neuronales recursivas y sus variantes, capaz de detectar con cierta eficacia aquellos outliers o valores atípicos de un conjunto de datos.

Con el continuo crecimiento del comercio electrónico(e-commerce), el uso de la web para compartir experiencias sobre distintos servicios o productos se ha convertido en un factor determinante para las personas. Cada vez hay más usuarios que recurren a las opiniones de los demás para la compra de un bien o un servicio.

Sin embargo, esto también ha resultado en que haya personas que de forma consciente publiquen opiniones engañosas, tanto positivas para impulsar la venta de un bien o negativas para perjudicar a cierto servicio. Esto ha llevado a que el producto afectado tenga una gran dificultad en recuperar la confianza de los demás y seguir siendo rentable.

Este hecho es cada vez más frecuente dado el incremento exponencial que se ha dado en los últimos años en el uso de las plataformas online dentro del sector de la hostelería y las reservas online. Este suceso junto a la gran cantidad de datos, principalmente opiniones de usuarios, que se recogen diariamente hace que se trate de un problema que es cada vez más frecuente en múltiples ámbitos, principalmente el sector de la hostelería.

En la actualidad, el fraude online, incluido el uso de opiniones engañosas, es un fenómeno común impulsado por los beneficios [25].

1.2. Antecedentes y estado actual del tema

En este momento, se está viendo una revolución global de la inteligencia artificial en todos los campos. Uno de los factores impulsores de esta revolución es el aprendizaje profundo. De hecho, el aprendizaje profundo ha tenido una evolución gradual durante varias décadas. El desconocimiento de los avances del aprendizaje profundo se debe a que los enfoques utilizados al principio eran relativamente

impopulares debido a sus diversas deficiencias. Los grandes avances del último periodo de tiempo han llevado a la popularización de dicho campo.

La primera etapa de este ámbito se conoce como aprendizaje biológico e intentaba replicar el funcionamiento del cerebro humano en un modelo computacional más simple con la finalidad de construir sistemas que se comporten como cerebros reales. Al cabo de un periodo de tiempo se introdujeron las redes neuronales artificiales, la idea principal siendo una red de unidades individuales que se pueden programar para lograr un comportamiento inteligente. Durante este periodo se desarrollaron múltiples modelos y técnicas que se siguen utilizando hoy en día.

El aprendizaje profundo se ha ido popularizando tras la creación de las redes de creencias profundas(DBN). Son una composición de múltiples capas ocultas donde cada capa contiene varias variables latentes. Las conexiones que existen se dan solo entre las capas pero no entre las variables. Otro aspecto fundamental que ha ayudado en el crecimiento de este campo ha sido el aumento de la capacidad de cálculo y la disponibilidad de grandes conjuntos de datos.

Hoy en día, el aprendizaje profundo está en todas partes. Se usa en una multitud de ámbitos tales como: determinar qué anuncios mostrar a un usuario [52], identificar y etiquetar personas u objetos en fotos [14], traducir voz a texto [46] o la conducción de vehículos autónomos [7]. También se encuentra en lugares menos visibles como la detección de fraudes [24], detección de diagnósticos y tratamientos de enfermedades [32] y la gama de aplicaciones podría continuar así de manera ilimitada. El aprendizaje profundo se está utilizando ampliamente para automatizar procesos y para la detección de patrones y resolución de problemas.

Dentro del aprendizaje profundo nos centraremos sobre todo en la detección de opiniones fraudulentas [4] y en la detección de anomalías [33, 3] y en menor medida en el procesamiento de lenguaje natural [43]. Para ello nos fijamos en el uso de las redes neuronales recurrentes(RNNs), que se han empleado en una multitud de ámbitos y para la resolución de numerosos problemas tales como: reconocimiento de voz[37], traducción automática [11], análisis de sentimientos [45], etc. Nos centraremos en el uso de la técnica no supervisada denominada Autoencoder, que se ha utilizado en varios ámbitos tales como: generación de imágenes [51], extracción de características [10], detección de opiniones fraudulentas [12] y sistemas de recomendación [47].

Capítulo 2

Tecnologías empleadas y metodología propuesta

En el capítulo anterior se ha realizado una breve introducción a nuestro proyecto, se ha definido la motivación del proyecto y los objetivos que se pretenden obtener. Por último, se han precisado los antecedentes que existen en relación al proyecto y las circunstancias actuales vinculadas al mismo. Para ello, se han mencionado múltiples aplicaciones o estudios en distintos ámbitos que se basan en la tecnología que emplearemos para la realización de nuestro proyecto.

En este capítulo, hablaremos más en detalle sobre esta tecnología, puntualizando en aspectos como el aprendizaje profundo y las redes neuronales recurrentes. Proseguiremos a enumerar las ventajas y desventajas de este tipo de red neuronal, y sus aplicaciones en los diferentes sectores. Finalmente, se va a hablar de la detección de anomalías, el tipo de arquitectura a seguir, denominada Autoencoder, y la metodología que se va a proseguir para el desarrollo del proyecto.

2.1. Deep Learning

El aprendizaje profundo o el Deep Learning (DL) es un subconjunto del aprendizaje automático basados en redes neuronales artificiales con aprendizaje de características. Las redes neuronales artificiales están inspiradas en el funcionamiento del cerebro humano siendo formadas por un conjunto de nodos que están siempre conectados y transmitiendo señales entre sí. Estas señales se transmiten desde la entrada hasta generar una salida. De manera similar a como aprendemos de la experiencia, el algoritmo de aprendizaje profundo realiza una tarea de forma repetida, cada vez adaptándola para mejorar el resultado final.

Se suele denominar 'aprendizaje profundo' porque este tipo de redes neuronales presentan varias capas de neuronas que permiten el aprendizaje. Es decir, cualquier problema que queremos resolver y que requiere cierto tipo de pensamiento para resolverlo se puede caracterizar como un problema de aprendizaje profundo. Cada neurona típicamente transforma los valores que recibe usando una función de activación, que modifica los valores de una manera, que al final del ciclo de entreno, permita a la red calcular cómo de lejos se encuentra de hacer una pre-

dicción precisa. El aprendizaje profundo se diferencia de las técnicas tradicionales de aprendizaje automático por sus características intrínsecas de aprender ciertas relaciones a partir de las representaciones entre o de los datos sobre los cuales se trabajan, sin introducir algunas reglas de forma manual o mediante el uso del conocimiento humano. Las arquitecturas altamente flexibles de los mismos permiten un aprendizaje más directo a partir de los datos y un incremento en la precisión del mismo al proporcionar más datos.

En nuestro caso concreto, el de la detección de anomalías en las opiniones de los usuarios, estas técnicas de aprendizaje profundo permitirían, al proporcionarle diferentes opiniones, que están constituidas por secuencias de texto, relacionar las diferentes palabras hasta encontrar cierta relación que determine que las opiniones están en una categoría u otra. Usando una arquitectura concreta para nuestro caso, el Autoencoder, que se irá explicando a lo largo de este capítulo, procederemos a entrenar nuestro modelo mediante el uso de muchas opiniones distintas de carácter 'verídico'. Este tipo de red neuronal será capaz de aprender a relacionar las distintas palabras y agruparlas en grupos de palabras con determinadas características. Al final, comprobaremos nuestro modelo mediante el uso de opiniones fraudulentas. El modelo deberá ser capaz de distinguir las características de estas opiniones sobre las otras y concluirá que las opiniones son fraudulentas.

Hoy en día, la cantidad de datos que se genera de forma diaria es una cantidad conmovedora, este siendo el recurso más importante que hace posible el aprendizaje profundo. El continuo incremento de los datos ha sido una de las razones por las que los algoritmos de DL han evolucionado en una gran proporción en los últimos años. Esto ha hecho que los sistemas de entrenamiento también requieran acceso a grandes cantidades de potencia de cómputo distribuida de alguna manera. En las redes neuronales también puede resultar difícil el entrenamiento del mismo por el problema del gradiente de desaparición o 'vanishing gradient problem', que puede llevar a empeorar el entreno en función del número de capas que haya en la red. A medida que se añaden más capas a la red, este problema puede provocar que se tarde demasiado tiempo en entrenar una red neuronal con un buen nivel de precisión.

Hay una amplia variedad de redes neuronales profundas, cada una con sus propias características y estructuras, siendo usadas para un tipo concreto de dato o tarea. En este trabajo nos centraremos en el uso de las redes neuronales recurrentes para formalizar nuestro propósito inicial expuesto.

2.2. Redes neuronales recurrentes

Las redes neuronales recurrentes (RNNs) son aquellas redes en dónde las conexiones entre nodos conforman un grafo dirigido a lo largo de una serie temporal. Este tipo de redes neuronales se utilizan de forma continua en el campo del procesamiento del lenguaje natural. Se denominan recurrentes ya que realizan la misma tarea para cada elemento de una secuencia, y la salida depende de las operacio-

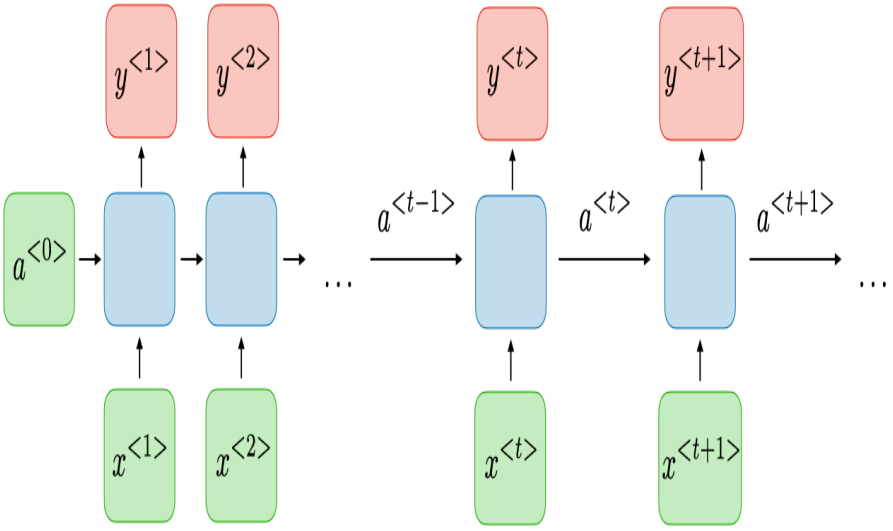


Figura 2.1: Estructura de una red neuronal recurrente

nes anteriores. Las RNNs también se pueden definir como unas redes neuronales con 'memoria', que recogen la información que ha sido calculada previamente. En teoría, aunque los RNNs podrían usar esta información previa en secuencias arbitrarias muy largas, en práctica, esto no es factible y se ve limitada a solo unos pasos de la secuencia. Normalmente estas redes tienen la siguiente estructura: Figura 2.1, dónde para cada *t*, el valor de activación $\mathbf{a}^{<\mathbf{t}>}$ y el output $\mathbf{y}^{<\mathbf{t}>}$ se definen de la siguiente manera:

$$\mathbf{a}^{<\mathbf{t}>} = \mathbf{g1}(\text{Waa} * \mathbf{a}^{<\mathbf{t-1}>} + \text{Wax} * \mathbf{x}^{<\mathbf{t}>} + \mathbf{ba}) \tag{2.1}$$

$$\mathbf{y}^{<\mathbf{t}>} = \mathbf{g2}(\text{Wya} * \mathbf{a}^{<\mathbf{t}>} + \mathbf{by}) \tag{2.2}$$

dónde: **Waa, Wax, Wya, ba, by** son coeficientes que se comparten a lo largo de toda la red neuronal y **g1, g2** son funciones de activación.

Por lo tanto, al tener los mismos pesos y sesgo para todas las capas ocultas de la red neuronal, estas capas se pueden combinar en una sola capa recurrente, cómo si haya una única neurona capaz de almacenar el estado de la entrada anterior y combinarlo con la entrada actual. De este modo se conservaría la relación entre las entradas actuales y pasadas.

En general, hay tres tipos de redes neuronales recurrentes:

- Una entrada para múltiples salidas: Usado en el reconocimiento de imágenes, basándose en una imágen que corresponde con múltiples 'palabras'.
- Varias entradas a una única salida: Usado en el análisis de sentimientos donde el texto se interpreta como negativo o como positivo.
- Varias entradas para múltiples salidas: Usado en la traducción automática, donde la palabra de una secuencia de texto se traduce de acuerdo con el contexto que representa en su conjunto.

Las claves de los algoritmos de las RNNs son los siguientes:

- Propagación hacia atrás a través del tiempo, para clasificar la entrada secuencial, vinculando un paso único con el siguiente.
- Gradientes de desaparicion o explosivos para preservar la precisión de los resultados.
- Unidades de memoria 'Long short-term' para poder reconocer las secuencias en los datos.

Las RNNs se basan en el hecho de que el resultado de una información depende del estado anterior o de varios resultados anteriores y esto podría dar lugar a ciertas inconsistencias para dependencias de largo alcance.Un ejemplo de esto dentro del procesamiento de lenguaje sería: 'El hombre que paseaba su perro tiene una chaqueta negra'. Esto haría, que el término 'una chaqueta negra' se refiera al 'hombre' y no al 'perro' y por esto mismo se considera una dependencia larga.

El problema anterior se conoce como el problema de gradiente que desaparece y surge cuando al aplicar la regla de la cadena, los nodos intermedios bajan de forma muy rápida, volviéndose innecesarios para la red. El otro problema que puede afectar a este tipo de redes, aunque no es tan preocupante como el primero, es el problema del gradiente explosivo, en los que se pueden dar errores sobre los pesos de una actualización y dar como resultado un gradiente muy grande resultando en unas actualizaciones 'falsas' en la red, haciendo que la red se vuelva inestable. Sin embargo, estos tipo de problema se pueden resolver empleando arquitecturas como LSTM o GRU [41]. En la Tabla 2.1 se pueden ver las ventajas y desventajas que presentan las redes neuronales recurrentes.

Ventajas	***Desventajas***
Posibilidad de procesar input de cualquier tamaño.	El cómputo de la red suele ser lento en muchos casos.
El tamaño del modelo no aumenta con el aumento del input.	Dificultad en el acceso a la información muy 'antigua'.
La computación en un momento dado tiene en cuenta información previa.	No puede considerar ninguna entrada futura para el estado actual.
Los pesos de la red se comparten a lo largo de todo el tiempo de la red.	Modelo poco interpretable en ocasiones.

Tabla 2.1: Comparación entre las ventajas y desventajas de las RNNs

En cuanto a las aplicaciones de las redes recurrentes distinguiremos su uso en los siguientes sectores: (1). Traducción automática, (2). Reconocimiento de voz, (3). Clasificación de texto en búsqueda semántica, (4). Detección de anomalías: Las

RNNs pueden ser muy útiles en la detección del fraude por los siguientes motivos: Los datos consisten en secuencias de patrones que se pueden explorar y evaluar. Esto permite al algoritmo asumir lo que puede suceder a continuación y determinar la probabilidad de que ocurra un cambio imprevisto. Este tipo de red ha sido empleada en la detección de fraudes [24], detección de opiniones fraudulentas [4, 12] y en la detección de anomalías [33, 3].

Existe una amplia gama de 'frameworks' de aprendizaje profundo, que permiten diseñar, entrenar y validar redes neuronales profundas, utilizando una variedad de lenguajes de programación diferentes. En nuestro caso concreto, nos basaremos en el uso de Keras [13]. Keras es una librería de código abierto que proporciona una interfaz intuitiva y amigable hecha en Python [34] para redes neuronales artificiales. Esta librería presenta múltiples ventajas tales como:

- API consistente y simple.
- Minimiza el número de acciones por parte del usuario.
- Ofrece unas acciones y logs de errores claros y comprensibles.
- Extensa documentación y guías para desarrolladores.

Keras hace uso de unos bloques comúnmente usados para las implementaciones de redes neuronales tales como: capas, objetivos, funciones de activación, optimizadores, etc. A parte de estos bloques estándar de las redes neuronales, Keras también ofrece soporte para redes neuronales convolucionales (CNNs) y RNNs. Para ello se disponen de ciertas herramientas como las capas de tipo: dropout, batch normalization, pooling. Keras es una librería flexible que te permite pasar de tu idea a plasmarla en código y crear una infraestructura denominada modelo que permita entrenar y luego evaluar dicho modelo (Figura 2.2).

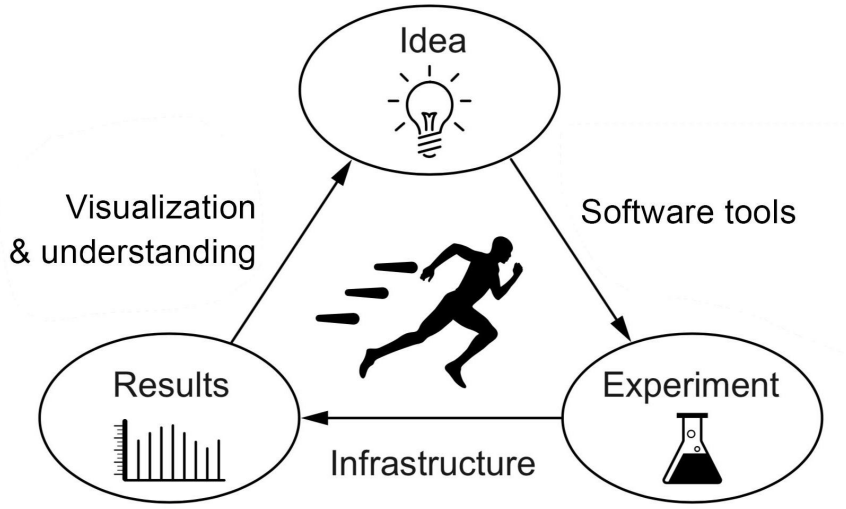


Figura 2.2: Ciclo de Keras

Keras al ser simplemente una API de alto nivel, actúa como interfaz para la librería TensorFlow [18], librería de código abierto para el aprendizaje automático. Se puede usar para una multitud de tareas distintas pero tiene un enfoque primario en el entreno y la inferencia de las redes neuronales profundas. Fue desarrollado por Google y permite su uso mediante múltiples lenguajes de programación tales como: Python, Java, C++. Su flexible arquitectura permite una sencilla implementación computacional sobre distintas 'plataformas' (CPUS, GPUs, TPUs) y con una variedad muy amplia de dispositivos desde escritorios hasta servidores y dispositivos móviles.

En cuanto al futuro del aprendizaje profundo, cabe destacar el hecho de que hoy en día hay múltiples arquitecturas de redes neuronales optimizadas para ciertos tipos de entradas y salidas. Hay redes, como las CNNs, que trabajan mejor en la clasificación de imágenes, mientras que otro tipo, las RNNs, permiten un mejor procesamiento y entreno de datos secuenciales.

Otro enfoque diferente al aprendizaje profundo sería el aprendizaje reforzado, en la que un 'agente' aprende por el método de prueba y error dentro de un entorno aislado solo a partir de unas recompensas y unos castigos. Estas extensiones dentro del aprendizaje profundo se conocen como aprendizaje profundo reforzado y han hecho un considerable progreso en los últimos años dentro de entornos como juegos tipo ajedrez o el juego de GO.

2.3. Detección de anomalías

Dentro del entorno del análisis de datos, una anomalía o un outlier se podría considerar una observación o un evento que tiene ciertas particularidades o características que lo diferencia de la mayoría de los datos. Normalmente, algunas posibles anomalías podrían resultar siendo problemas relacionados con el fraude bancario, errores de texto, o incertidumbres médicas [19].

La detección de anomalías es un problema crucial hoy en día, ya que normalmente suelen ayudar a indicar información valiosa y muchas veces crítica, que podría afectar un negocio u organización.

Generalmente, las anomalías se suelen clasificar en cuatro categorías [9]:

- Anomalías puntuales: Se consideran aquellos puntos de datos muy diferentes al resto de los puntos de datos. Habitualmente, son aquellos valores extremos de un dataset.
- Anomalías de grupo/colectivas: Son aquellos puntos que tomados de forma singular son normales, pero que consisten una anomalía al tratarse en un grupo.
- Anomalías contextuales: Se trata de los puntos de datos que vistos desde un cierto contexto son anómalos pero en otras circunstancias se consideran

normales. El ejemplo más común de este tipo de anomalía serían las series temporales de datos, que aunque los datos están dentro de un rango normal y concreto, no se adapta al patrón esperado de temperaturas.

- Anomalías de puntos cambiantes: Estas anomalías solo están relacionadas con las series temporales de datos y aluden a aquellos puntos cuyo patrón normal cambia.

Aunque tenemos y conocemos la existencia de estas categorías de anomalías, no siempre es tan fácil identificarlos. Estas anomalías se consideran una desviación del comportamiento de los datos 'normales' y por lo tanto, muchas veces es complicado conocer e identificar a priori cuáles y cómo son las anomalías de nuestro dataset. Al mismo tiempo debemos tener en cuenta que el tipo de los metadatos y meta características que conlleva la o las anomalías de un dataset depende del contexto en el que se encuentran, es decir varían en función del ámbito y su aplicación en el mismo.

Otro problema importante en cuanto a la detección y evaluación de las anomalías consiste en el hecho de que la mayoría de los datasets existentes en la actualidad no están etiquetados de ninguna forma. Para algunos datasets, en algunos ámbitos, es suficiente con fijar un nivel de tolerancia (threshold) y cualquier valor fuera de ese nivel de tolerancia implicaría una anomalía.

Sin embargo, este proceso de etiquetar las anomalías de un dataset es una labor compleja, que consume tanto tiempo de procesamiento, en caso de que se emplee un modelo artificial para la identificación y categorización de las anomalías, o tiempo y conocimiento humano, en caso de emplearse el procesamiento y categorización de forma 'manual' por parte de un individuo.

Este problema también se ha encontrado en la realización de este trabajo, encontrándose muchos datasets, que se analizarán en el siguiente capítulo, que no presentaban una categorización directa de la anomalía o los límites que necesitábamos saber sobre un campo concreto. En este caso, nos interesaba saber si la opinión de un usuario era engañosa, la cual se trataba como una anomalía o era sincera, la cual se trataba como un dato normal. Por lo tanto, se han tenido que descartar todos aquellos datasets que no presentaban dicha característica ya que al no presentar esta peculiaridad no se podría haber creado un modelo capaz de realizar el propósito manifestado en este trabajo, es decir, crear un modelo capaz de detectar opiniones fraudulentas. También se ha tenido en cuenta que esta categorización podría haberse hecho o bien por medio de un modelo artificial, lo cual hubiese tenido resultados posiblemente erróneos, o mediante el conocimiento humano, lo cual también se ha visto como una solución inviable.

2.4. Técnicas para la detección de anomalías

Para la detección y aprendizaje de las características jerárquicas de los datos sin necesidad de emplear métodos 'manuales', se pueden utilizar métodos supervisa-

dos, híbridos, redes neuronales de una clase, semi-supervisados o no supervisados mediante el uso de Autoencoder [8].

Estas técnicas de detección de anomalías se basan principalmente en los siguientes aspectos:

- El tipo de datos de entrada: Elegir un tipo concreto para la detección de las anomalías depende principalmente del tipo de datos de entrada. La arquitectura mas común según el tipo de datos se puede ver en la Tabla 2.2.
- Disponibilidad del etiquetado de los datos: Como se ha mencionado anteriormente, las anomalías son datos complicados de procesar y etiquetar. Nos centraremos en el uso de los Deep Autoencoders, en entrenar datos sin anomalías de forma semi supervisada. Con suficientes datos de entreno, los Autoencoders producirían un error de reconstrucción bajo lo que permitirá detectar aquellos datos que presentan características particulares que serían las anomalías [48].
- Objetivo del entreno: Técnicas basadas en el objetivo de entreno del modelo.
- Tipos de anomalías: Dependiendo de los diferentes tipos de anomalías, tipos que se han discutido anteriormente.
- Output de las técnicas de detección de anomalías: Un factor fundamental en la detección es saber cómo se han detectado dichas anomalías. Para ello, se suelen implementar métodos que se basan o bien en una puntuación o un etiquetado. La puntuación describe el nivel de cuánto 'anómalo' es cada punto. También se suele indicar un nivel límite fijado por un experto para la identificación de anomalías. Es una técnica que divulga más información que la técnica del etiquetado [36]. Por otra parte, la técnica etiquetada consiste en asignar una etiqueta normal o anómala a cada instancia.

Tipo de datos	***Arquitectura de detección de anomalías***	***Ejemplos de datos***
Datos secuenciales	CNN, RNN, LSTM	Video, Series Temporales
Datos no secuenciales	CNN, Autoencoder	Imágenes, datos

Tabla 2.2: Arquitectura propuesta para la detección de anomalías según el tipo de datos de entrada

Las técnicas de detección de anomalías tienen un amplio uso en diferentes sectores y con distintos propósitos(Figura 2.3). Algunos sectores y campos de aplicación de estas técnicas serían: (1). Detección de intrusos, (2). Detección de intrusos

en la red, (3). Detección de fraude, (4). Detección de malware, (5). Detección de anomalías en redes sociales: Identificar los comportamientos ilícitos de los individuos es un factor fundamental ya que las consecuencias que podrían traer las acciones de estos podrían tener un impacto social muy grande. Esta técnica está muy bien conocida y documentada en la literatura [23]. Hay una variedad de técnicas que se han demostrado ser eficientes en la detección de dichas anomalías. La técnica en la que vamos a centrarnos es el Autoencoder [54, 8].

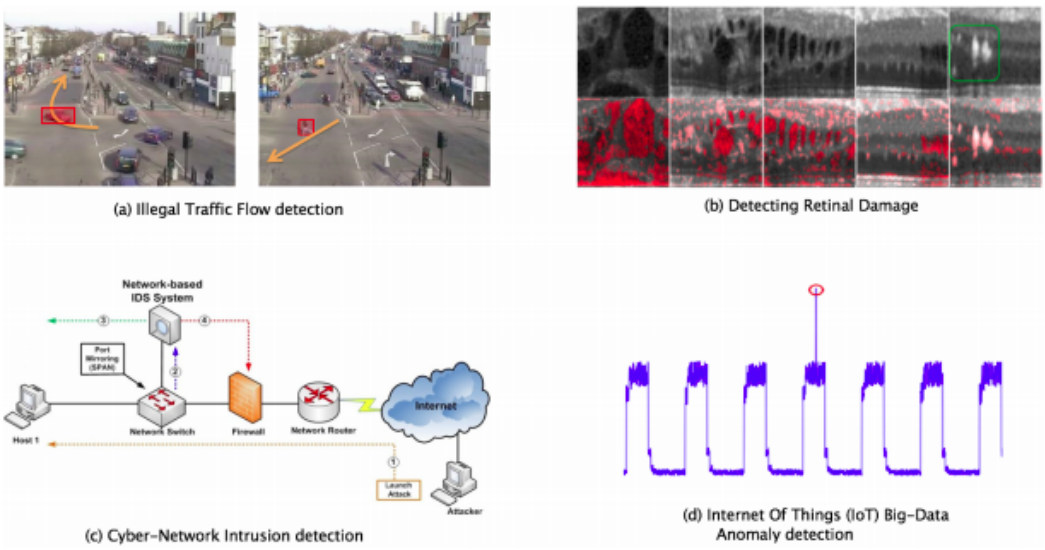


Figura 2.3: a) Análisis de imágenes [49] b) Medical [38] c) Redes [20] d) Detección de ánomalias en Big-data [26]

2.5. Arquitectura del Autoencoder

El Autoencoder es un tipo de red neuronal artificial que se utiliza para aprender codificaciones de datos eficientes de manera no supervisada. El principal objetivo de un Autoencoder es aprender una representación (encoding) de un grupo de datos, normalmente para la reducción de dimensionalidad. Aparte de la función de reducción, está la función de reconstrucción, en donde el Autoencoder es capaz de generar a partir de la codificación reducida una representación lo más cercana posible al dato inicial.

En cuanto a la arquitectura que presentan los Autoencoders, destacamos que, la forma más simple es una red neuronal no recurrente que tiene una capa de entrada, una capa de salida y una o más capas ocultas que las conectan, con la condición de que la capa de salida debe tener el mismo número de nodos que la capa de entrada, con el propósito de reconstruir las entradas (Figura 2.4). Esta arquitectura se podría descomponer en dos partes: una función encoder $E = f(X)$ que transforma las entradas X, en E codificadas; y una función decoder $X' = g(E)$ que retorna una reconstrucción de las entradas X'. Para aprender los pesos de las

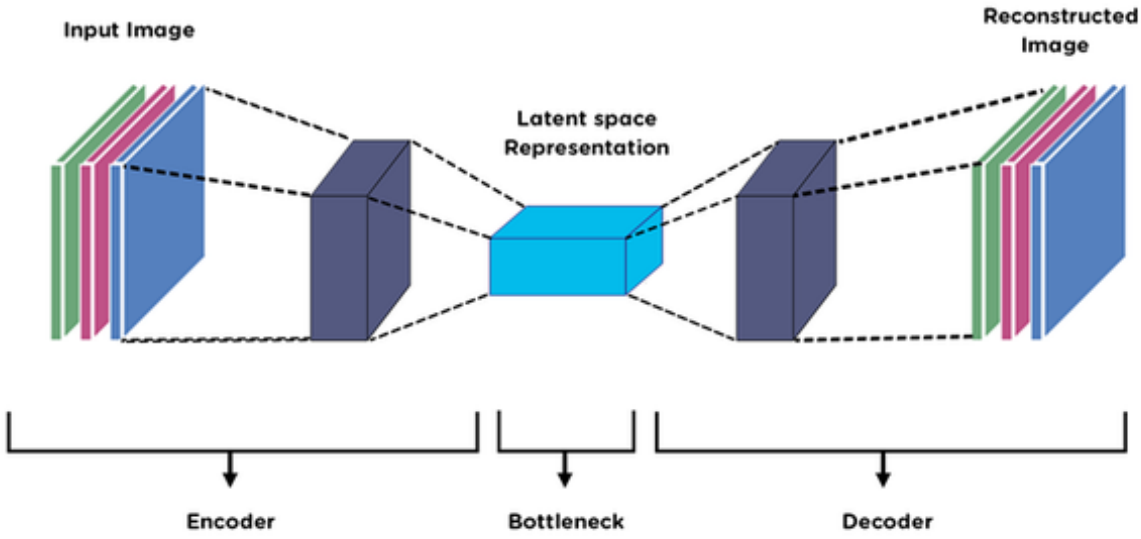


Figura 2.4: Autoencoder

neuronas y, por tanto, las codificaciones, el Autoencoder busca minimizar alguna función de pérdida, como el mean squared error (MSE), que penaliza a X' por ser diferente de X [6]:

$$\mathcal{L}(\mathbf{x}, \mathbf{x}') = |\mathbf{x} - \mathbf{x}'||^2 = \|\mathbf{x} - \sigma'(\mathbf{W}'(\sigma(\mathbf{W}\mathbf{x} + \mathbf{b})) + \mathbf{b}')\|^2 \tag{2.3}$$

2.6. Aplicaciones del Autoencoder

Al comienzo, en los años 80, las principales aplicaciones de los Autoencoders eran la reducción de la dimensionalidad y la recuperación de información. Los usos de esta arquitectura han ido evolucionando a lo largo de los años y hoy en día, se utiliza en varios ámbitos: (1). Reducción de la dimensionalidad, (2). Procesamiento de imágenes, (3). Interpretación automática, (4). Detección de anomalías: Al aprender a replicar las características más destacadas en los datos de entrenamiento, se alienta al modelo a aprender cómo reproducir con precisión las características más comunes de los datos. Ante datos que no presentan estas características, el modelo debería agravar su rendimiento de poder reconstruir los datos de forma parecida a los inicales. De forma general, los modelos suelen utilizar datos con instancias normales para el entreno del Autoencoder. También se puede dar el caso de que el número de anomalías del conjunto de datos sea tan reducido con respecto al conjunto total de datos, que su aporte al aprendizaje del modelo sea ignorarable. Después del entrenamiento, el Autoencoder será capaz de reconstruir de forma precisa los datos normales mientras que, tendrá dificultades al reconstruir los datos anómalos. Por lo tanto, de forma general se utiliza como referencia para la detección de anomalías el error entre el dato reconstruido y el dato original.

2.7. Metodología

Por lo tanto, para cumplir nuestro objetivo final, es decir, el de la creación de un modelo capaz de detectar opiniones fraudulentas a partir de un dataset de opiniones de usuarios reales, vamos a seguir un cierto procedimiento para su realización.

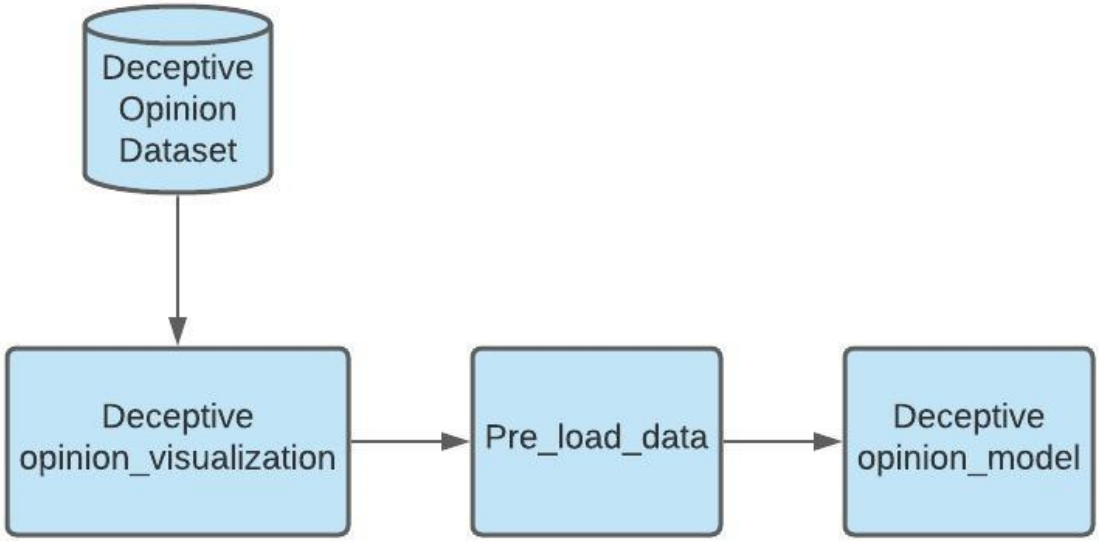


Figura 2.5: Procedimiento general que se va a seguir para cumplir con el objetivo general

Tal y como se puede apreciar en la Figura 2.5, la primera parte del procedimiento general va a consistir en el análisis y elección, según varios criterios objetivos, del dataset más propicio para cumplir con nuestro objetivo final. Esta fase es probablemente una de las partes más importantes del ciclo general ya que sin datos o con unos datos inconsistentes, no se podrá crear un modelo, o los resultados del modelo creado no serán satisfactorios. Esta fase se desarrollará y se detallará en más profundidad a lo largo del Capítulo 3.

Tras este primer ciclo, una vez que se haya encontrado un dataset que creamos que nos sirva para nuestro propósito final, vamos a dividir la metodología general en tres scripts distintos para facilitar la lectura y agilizar el procedimiento general:

- Deceptive opinion-visualization: Este primer script contendrá la fase del estudio exploratorio visual de nuestro dataset, viendo cómo se relacionan los datos y extrayendo ciertas conclusiones. Estos aspectos se detallarán en el Capítulo 4 en la Sección 4.1. Esta fase aunque no sea obligatoria en un proceso de aprendizaje profundo, ya que no afecta y no determina el resultado final del modelo, es interesante realizarla ya que se podrían sacar ciertos términos que nos ayuden a entender mejor los datos existentes y la afinidad entre los distintos campos del dataset.

- Pre-load-data: El segundo script, una vez finalizada la etapa de visualización de los datos, contendrá la fase inicial, previa a la modelización. Esta fase, que se puede ver más detallada en el Capítulo 4 en la Sección 4.2, consiste en la preparación de los datos de nuestro dataset mediante múltiples técnicas para su posterior uso en el modelo que vamos a definir a continuación. Esta parte se encarga de evitar las inconsistencias que pueden haber en los datos. Es fundamental ya que afectaría de modo directo los resultados finales de nuestro modelo.

- Deceptive opinion-model: El último script se encargará de la creación de un modelo siguiendo la arquitectura general de una red basada en Autoencoder y que a su vez contendrá capas basadas en RNNs del tipo LSTM. Tras la creación del primer modelo se emplearán múltiples métricas para obtener la 'eficacidad' del modelo y ver cómo se comporta nuestro modelo. Luego, a partir de ese modelo inicial se van a crear otros modelos similares modificando ciertos parámetros de la red con el objetivo de encontrar unos modelos que actúen mejor que el modelo original. Por último, se van a comparar los resultados obtenidos siguiendo varias métricas, para concluir cuál es el mejor modelo.

En este capítulo se ha hecho una breve introducción al deep learning, redes neuronales recurrentes, la detección de anomalías y por último, la técnica no supervisada que se va a usar para implementar este modelo, el Autoencoder. Finalmente, se ha expuesto la metodología a seguir.

Capítulo 3

Selección del conjunto de datos

En el capítulo anterior se ha hecho una breve introducción al deep learning, redes neuronales recurrentes, la detección de anomalías y por último, la técnica no supervisada que se va a usar para implementar este modelo, el Autoencoder. Finalmente, se ha expuesto la metodología a seguir.

En este capítulo vamos a explorar diferentes conjuntos de datos (datasets), reales y de fuente abierta. Analizamos y elegimos en base a ciertos criterios el dataset más adecuado para cumplir con nuestro objetivo final.

3.1. Búsqueda de diferentes conjuntos de datos

Antes de comenzar con la implementación del modelo necesitamos centrarnos en los datos con los cuales vamos a trabajar. Para ello, identificamos datos reales que sirvan nuestro propósito.

Por ello, se ha empezado a buscar datasets reales en fuentes como bibliografías especializadas, portales científicos y revistas de alto impacto científico (Tabla 3.1).

3.2. Criterios para la elección del conjunto de datos

Antes del análisis y la elección del dataset óptimo, se procede a la observación de las pautas determinantes en la elección del conjunto de datos:

- Opiniones de usuarios reales: Nuestro dataset debe contener opiniones de diferentes usuarios reales sobre productos o servicios. Estos datos deberían ser en un idioma internacional y de uso cotidiano. Nos vamos a centrar sobretodo en idiomas basados en el alfabeto latino, ya que otros idiomas, como el chino supondrán problemas adicionales [42].

  Además, aunque existan técnicas para el procesamiento de estos tipos de lenguajes naturales, la técnica que emplearemos, el Doc2Vec [15], es más eficaz al trabajar con idiomas basados en el alfabeto latino. Por lo tanto, valoraremos positivamente que las opiniones sean de este tipo.

- Etiquetado de las opiniones: Un tipo de dato que nos exponga de alguna forma si la opinión en cuestión es normal o anómala. En caso de que el

Corpus	***Descripción***	***Tamaño***	***Fecha***	***Referencia***
Opinion Fraud Detection	Opiniones sobre distintos productos de Amazon China.	∼190mb ∼1.2 millones de registros	2013	[50]
Deceptive Opinion Spam Corpus	Opiniones obtenidas de múltiples fuentes, sobre hoteles del área de Chicago.	∼1.3mb 1600 registros	2011, 2013	[31, 30]
Trip Advisor Hotel Reviews	Opiniones recogidas de Trip Advisor sobre diferentes hoteles.	∼15mb 20491 registros	2016	[5]
Yelp Reviews Polarity	Opiniones recogidas de Yelp sobre distintos servicios tales como: restaurantes, bares, y otros servicios públicos.	∼437mb 598.000 registros	2015	[53]
Amazon Review Data	Opiniones recogidas de Amazon US, sobre distintos productos.	∼34gb ∼233 millones de registros	2018	[27]

Tabla 3.1: Comparación de los diferentes datasets encontrados

etiquetado de las opiniones no esté presente, necesitaremos usar un modelo clasificador semi-supervisado de inteligencia artificial para conseguirlo. Al tratarse de un modelo semi-supervisado sería necesario el conocimiento humano especializado para poder clasificar una pequeña muestra de las opiniones y a partir de estas clasificar el resto de las opiniones.

Lo ideal sería obtener un dataset que presente el atributo discriminador entre las distintas opiniones.

- Tamaño: Otro aspecto a tener en cuenta es la cantidad de registros del dataset. Aunque un conjunto de datos más grande supondrá el uso de una cantidad considerable de recursos, también es cierto que el tamaño de este conjunto influirá de forma positiva en los resultados a obtener.

Se debe notar que podrían haber otros criterios a valorar para la elección del dataset óptimo, entre otros, siendo el de la proporción entre las opiniones normales y las anómalas.

3.3. Análisis de los datasets

Lo siguiente que se ha hecho fue el análisis y la elección del dataset más propicio para cumplir con nuestro objetivo, el de la detección de opiniones fraudulentas. Una vez expuestos los criterios por los cuales nos vamos a basar en la elección del dataset, vamos a proseguir a analizar los diferentes datasets (Tabla 3.1):

- 'Opinion Fraud Detection': Este dataset presenta distintas opiniones sobre diferentes productos de Amazon China. Los registros presentan los siguientes atributos: id del producto, nombre del usuario, título de la opinión, contenido de la opinión, categoría del producto. Aunque este dataset presenta algunos atributos que podrían mostrar cierto interés sobre todo en el análisis previo exploratorio, también presenta una gran deficiencia que es el uso del idioma china en las opiniones. Además no presenta el atributo que permita distinguir si una opinión es anómala o no, y esto implicaría, como se ha mencionado anteriormente, el uso de un conocimiento humano y/o el uso de un modelo clasificador capaz de distinguir entre los dos tipos de opiniones. Esto hace que este dataset no sea valorado positivamente, al cumplir solo con el criterio de la cantidad de registros adecuados.

- 'Deceptive Opinion Spam Corpus': Se trata de un dataset con opiniones obtenidas a partir de múltiples fuentes sobre hoteles del área de Chicago. Está compuesta por múltiples atributos tales como: nombre del hotel, polaridad de la opinión(positiva o negativa), fuente, contenido de la opinión del usuario y por último, un atributo que indica si la opinión es 'fraudulenta' o 'verídica'. A pesar de que este dataset no sea muy grande, es un muy buen candidato ya que cumple con los dos primeros requisitos mencionados anteriormente y presenta además unos atributos adicionales que nos podrían ayudar en un análisis exploratorio inicial del dataset.

- 'Trip Advisor Hotel Reviews': Se trata de un dataset que muestra información de las opiniones de distintos usuarios sobre hoteles obtenidos a partir de la plataforma TripAdvisor. Consta de solo dos atributos: el contenido de la opinión del usuario y la calificación puesta por el usuario al hotel. Se trata de un dataset medianamente grande que, sin embargo, no cumple con el segundo criterio. Este tipo de dataset, aunque se podría utilizar para nuestro propósito, se comporta mejor en un modelo predictivo capaz de obtener la calificación de una opinión.

- 'Yelp Reviews Polarity': Este dataset contiene información sobre las opiniones de diferentes usuarios sobre múltiples servicios o productos de la plataforma Yelp. Se caracteriza por sus dos principales atributos: contenido de la opinión del usuario y grado de polaridad de la opinión. Este último atributo tiene el siguiente significado:

    - '1': Opiniones clasificadas como negativas, marcadas con una o dos estrellas de puntuación.

- '2': Opiniones clasificadas como positivas, marcadas con tres o cuatro estrellas de puntuación.

Este dataset se encuentra en la misma situación que el dataset anterior, y aunque hubiera sido interesante analizarlo ya que presenta una cantidad extensa de datos, no presenta el atributo discriminador entre las distintas opiniones. Esto hace que este dataset, tal como el anterior, presente unas características que lo hagan idóneo para otros estudios y otros modelos tales como la predicción de la polaridad de las opiniones de los usuarios.

- 'Amazon Review Data': Este dataset se sustenta en información sobre distintas opiniones y metadatos de varios productos de la plataforma de Amazon Global. Al tratarse de un dataset muy grande, cerca de los 233 millones de opiniones distintas, se han tenido que separar estas opiniones por categorías, habiendo alrededor de 29 categorias. La cantidad de opiniones por cada categoría se ve fluctuante, según la importancia de la categoría yendo desde los 90.000 opiniones hasta los 50 millones de opiniones. Cada uno de estos datasets, que representan cada categoría, tiene información sobre el usuario, la opinión del usuario e información del producto. Por lo tanto, la información que presenta cada dataset sería la siguiente: id del usuario, nombre del usuario, contenido de la opinión del usuario, título del producto, descripción del producto, precio del producto, etc. En consecuencia, podemos decir que se trata de un dataset detallado, a partir del cual se podrían hallar y hacer múltiples estudios, pero, tal como algunos de los datasets anteriores, no presenta uno de los criterios importantes para la realización de nuestro modelo.

El dataset que hemos elegido ha sido el segundo analizado [31, 30], titulado 'Deceptive Opinion Spam Corpus', ya que tras examinarlo hemos concluido que es el más adecuado puesto que cumple con los criterios fijados inicialmente y presenta unos datos interesantes para nuestro modelo. Vamos a proseguir a examinarlo un poco más en detalle y ver como nos ayudaría a solventar y realizar nuestro objetivo.

3.4. Análisis del *Deceptive Opinion Spam Corpus*

El corpus del dataset elegido contiene la siguiente informacion: distintas opiniones etiquetadas como 'fraudulentas' o como 'verídicas', el hotel sobre el cual se opina, la polaridad de la opinión, es decir, negativa o positiva, la fuente de la cual se ha obtenido la opinión y por último, el contenido de la opinión (Tabla 3.2). Tiene un tamaño total de 1600 registros, que se podría dividir de la siguiente manera:

- 400 opiniones verídicas y positivas de TripAvisor.
- 400 opiniones fraudulentas y positivas de MTurk.
- 400 opiniones verídicas y negativas de otras páginas web tales como: Expedia, Hotels.com, Orbitz.

- 400 opiniones fraudulentas y negativas de MTurk.

Cada una de estas categorías contiene un total de 20 opiniones de cada uno de los 20 hoteles más populares de Chicago.

deceptive	***hotel***	***polarity***	***source***	***text***
deceptive	palmer	positive	MTurk	Amazing! I was swept away when I walked into the hotel it was gorgeous. The staff was very nice and helpful the hotel was very clean, and the food was delicious. We stayed two nights and the first night we went to the Lockwood restaurant and we were blown away by the food. It has to be the best food in Chicago the atmosphere was very classy and warm and the wait staff was very precise. The decor in our rooms did not look like any hotel I had been at they were very elegant and inviting. I cannot wait to go back it is a true escape from daily life.
truthful	hyatt	positive	TripAdvisor	Triple A rate with upgrade to view room was less than $200 which also included breakfast vouchers. Had a great view of river, lake, Wrigley Bldg. & Tribune Bldg. Most major restaurants, Shopping, Sightseeing attractions within walking distance. Large room with a very comfortable bed.

Tabla 3.2: Muestra del dataset Deceptive Opinion Spam Corpus

Por último, vamos a detallar la estructura del dataset, viendo los diferentes campos con sus posibles tipos y valores (Table 3.2):

- Deceptive: Campo que indica si la opinión es verídica o fraudulenta, siendo un campo de tipo 'String' con solo dos tipos de valores posibles: deceptive y truthful. Es un campo esencial para el modelo ya que necesitamos entrenar al modelo con las opiniones y para ello necesitamos saber si las opiniones son sinceras o engañosas.

- Hotel: Campo de tipo 'String' que indica el nombre del hotel sobre el cual se ha hecho la opinión. Hay un total de 20 posibles valores, algunos de los cuales serían: affinia, hyatt, conrad, omni, etc. Es un atributo que a priori no llegaría a usarse en el entrenamiento pero que puede resultar útil en la exploración inicial del dataset.

- Polaridad: Campo que al igual que los otros campos es de tipo 'String' con solo dos valores posibles: positivo y negativo. Puntualiza en el tipo de la opinión, informando si se trata de una opinión positiva o negativa. Este atributo tampoco se vería incluido directamente en el entrenamiento del modelo pero tal como el atributo anterior podría resultar interesante observar ciertas agrupaciones visuales del mismo en un análisis exploratorio.

- Fuente: Campo de tipo 'String' que caracteriza el tipo de fuente de la cual se ha extraído la información. Sus posibles valores serían: MTurk, TripAdvisor y Web. Dentro del valor 'Web' hay varias subpáginas(Expedia, Hotels.com, etc) que se han consultado para sacar esta información. Tal como los dos anteriores atributos, este tampoco influirá de forma directa en el entrenamiento del modelo y solo será útil en un análisis exploratorio que se verá en el siguiente capítulo.

- Texto: Último campo del dataset que consta de la opinión del usuario, siendo de tipo 'String' tal como los otros atributos. Es un campo de longitud variable que cumple con el criterio que hemos impuesto del uso de un lenguaje de uso internacional basado en el alfabeto latino. Además resultará fundamental en el entrenamiento de nuestro modelo, ya que el modelo se basará en la relación entre las palabras para poder distinguir si se trata de una opinión fraudulenta o verídica. Sin embargo, para poder trabajar con este tipo de dato será necesaria su transformación en un vector de características para poder usarlo en nuestro modelo.

Capítulo 4

Desarrollo del modelo de detección de opiniones fraudulentas

En el capítulo anterior se han explorado y analizado diferentes datasets posibles para nuestro modelo. En base a diferentes criterios se ha elegido el dataset más favorable para el cumplimiento de nuestro objetivo.

En este capítulo, haremos hincapié en el proceso de creación de un modelo capaz de detectar opiniones fraudulentas. Este proceso se ha dividido en varias subetapas. Cada uno de estos pasos es importante para lograr nuestros objetivos. Un diagrama genérico de este proceso se muestra en la (Figura 4.1).

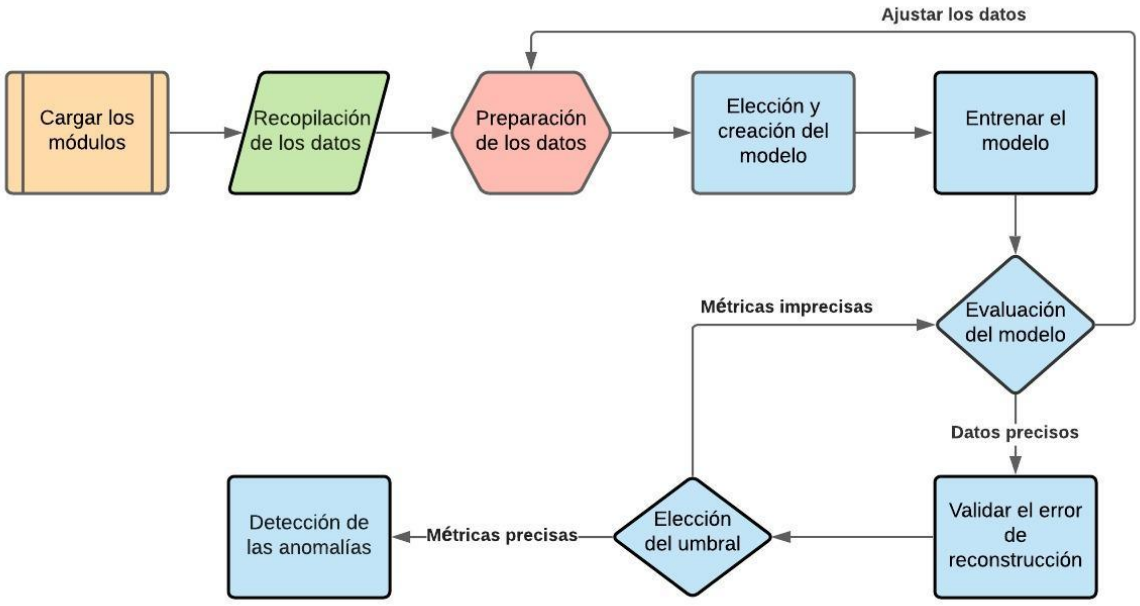


Figura 4.1: Proceso de aprendizaje automático

Tanto el código completo de las fases que se expondrán a continuación y el desarrollo general del proyecto se podrá ver en: Github(TFG-DL-Autoencoder)[44].

Tal y como se ha visto anteriormente en la Sección 1.1, queremos reiterar y

hacer énfasis en nuestro objetivo: la creación de un modelo basado en redes de aprendizaje profundo centrado en la técnica no supervisada llamada *Autoencoder* con la finalidad de detectar datos anómalos o outliers.

En nuestro supuesto caso, tras haber elegido el dataset óptimo, queremos implementar un modelo que, a partir de las opiniones de unas personas reales, etiquetadas como engañosas o sinceras, sea capaz de detectar aquellas opiniones anómalas, es decir, las opiniones engañosas.

4.1. Exploración de los datos

Una etapa que aunque no afecta directamente el resultado del modelo, pero puede informarnos de ciertos patrones en los datos es el análisis exploratorio visual de los datos. Con esto pretendemos mostrar de forma visual los datos con los cuales trabajaremos y las relaciones que existen entre ellos. Esto nos ayudará a tomar ciertas decisiones desde el punto de vista de la implementación del modelo más adelante en la Sección 4.3.

Empezamos viendo la distribución de las opiniones por la fuente, tal y como se puede observar en la Figura 4.2. De las 1600 opiniones que tenemos recogidas en nuestro dataset, la distribución de la misma en función de las tres fuentes disponibles parece ser más o menos uniforme, habiendo 800 opiniones de MTurk, 400 de TripAdvisor y 400 de otras fuentes Web.

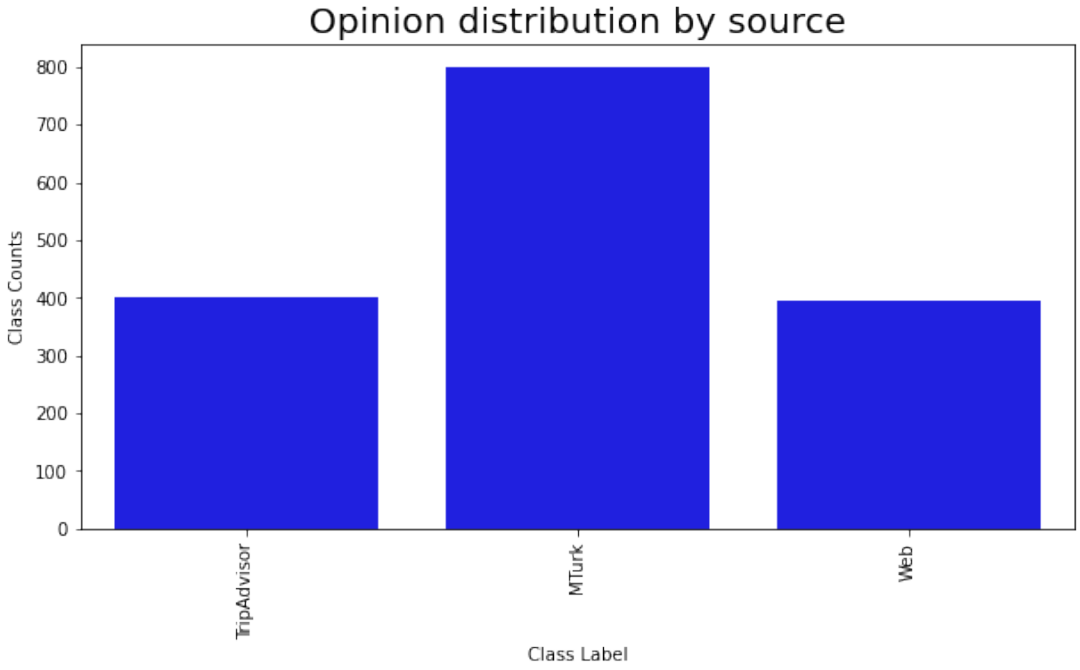


Figura 4.2: Distribución de las opiniones por fuente

Pasando a mirar la distribución de las opiniones de los usuarios por tipo de hotel (Figura 4.3) vemos que para los diferentes hoteles que aparecen en nuestro dataset, habiendo un total de 20 hoteles, la distribución es homogénea, es decir hay un

total de 80 opiniones para cada hotel existente en el dataset y además para cada uno de esos hoteles las opiniones están distribuidas de la siguiente manera: 40 opiniones de MTurk, 20 opiniones de otras fuentes Web y 20 opiniones de TripAdvisor. Esta distribución por hoteles sigue el mismo patrón general de distribución por fuente del dataset general y hasta aquí podemos concluir que el dataset se ha realizado con el propósito de que las personas que lo estén trabajando no tengan ninguna incógnita sobre si, en caso de que las opiniones no están bien distribuidas esto haga que una clase u otra sea más propensa a ser de un tipo u de otro, en nuestro caso sea más fraudulenta o más sincera.

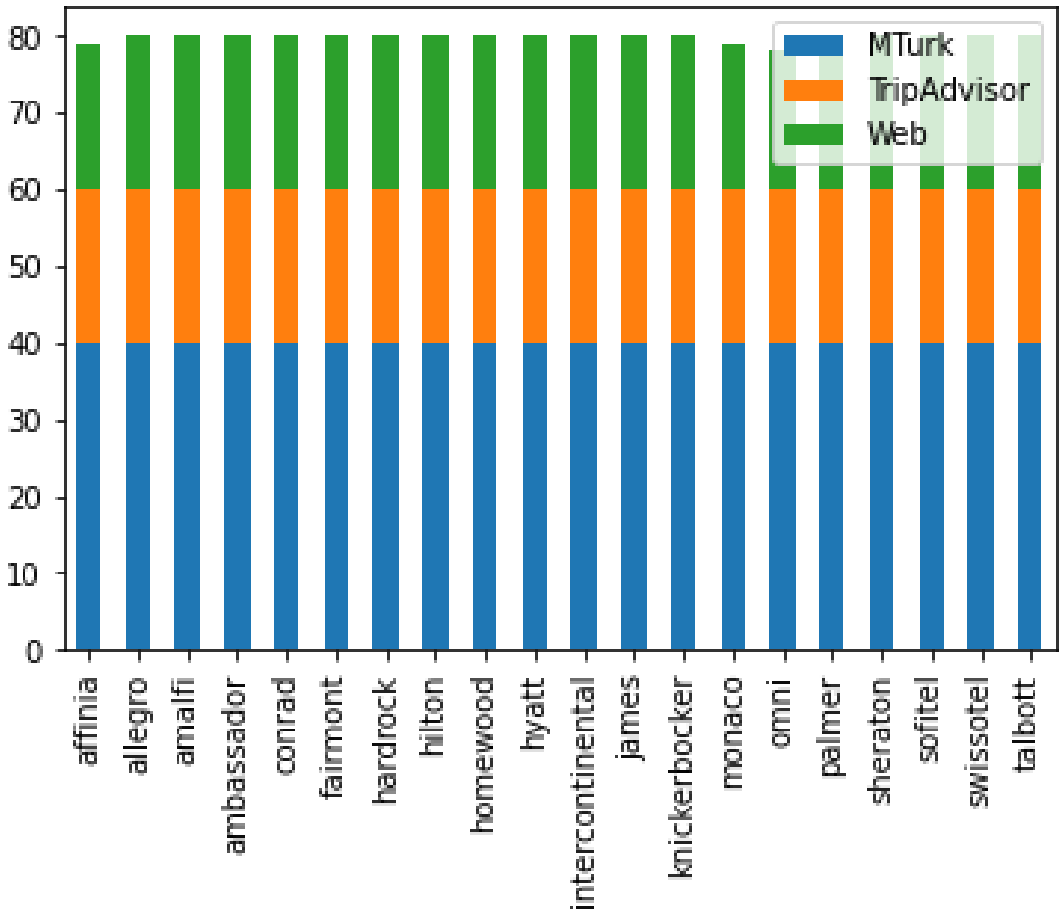


Figura 4.3: Distribución de las opiniones por fuente

Del mismo modo analizaremos la distribución de la polaridad, es decir positivas o negativas, de estas opiniones según las diferentes fuentes (Figura 4.4). Como se puede ver, esta distribucion tiene el siguiente patrón: para la fuente MTurk, que como vimos tenía 800 opiniones, se distribuyen de forma igual las opiniones según las dos polaridades, y para las otras dos fuentes que tenían solo 400 opiniones cada una, vemos que para cada una de las fuentes hay solo opiniones de una sola polaridad. Para la fuente TripAdvisor hay 400 opiniones positivas y ninguna negativa, mientras que para la fuente Web se ha hecho lo contrario, es decir, hay 400 opiniones negativas y ninguna positiva. La distribución de las fuentes TripAdvisor y Web es un tanto impropia ya que no se ha optado por una distribución igua-

litaria, es decir, 200 opiniones positivas y 200 opiniones negativas con el fin de obtener una distribución perfectamente distribuida. Este hecho se pudo dar por la imposibilidad de encontrar tantas opiniones negativas (como positivas) de la fuente TripAdvisor (sobre los hoteles en cuestión), o no se han encontrado tantas opiniones positivas(como negativas) de la fuente Web para llegar a una distribución equilibrada.

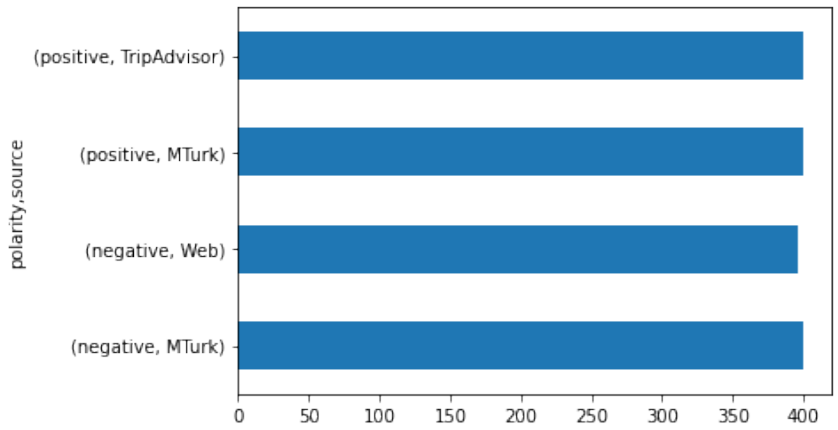


Figura 4.4: Distribución de la polaridad de las opiniones por fuente

A continuación, mirando la Figura 4.5, nos damos cuenta de que para las fuentes anteriores que hemos analizado, que solo tenían opiniones de una sola polaridad, positivas en caso de la fuente de TripAdvisor y negativas de la fuente Web, las opiniones son todas sinceras. En cambio para la fuente MTurk, la totalidad de las opiniones consta de opiniones fraudulentas.

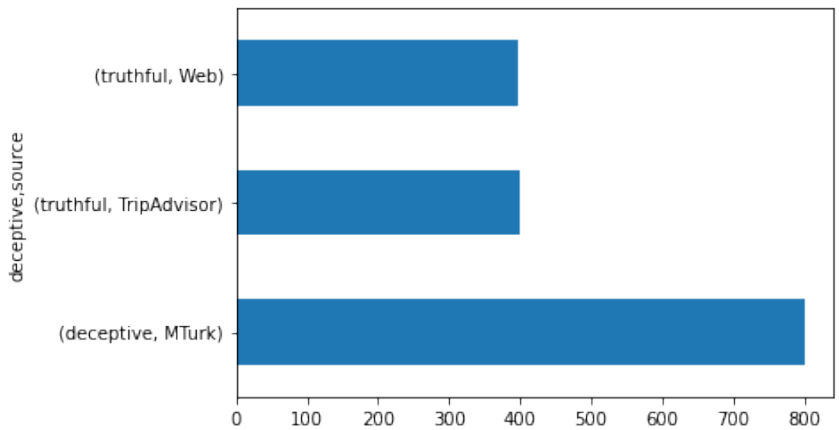


Figura 4.5: Distribución de las opiniones sinceras/engañosas por fuente

Este hecho hace que vamos a entrenar nuestro modelo utilizando datos sinceros de las dos fuentes que tengan estos tipos de datos, TripAdvisor y Web, prosiguiendo a evaluar nuestro modelo, utilizando datos tanto de las fuentes que tengan datos sinceros (TripAdvisor y Web), como de la fuente que tenga datos fraudulentos, MTurk. Este hecho va a dificultar un poco la precisión de nuestro modelo ya

que implicaría el uso de datos de la fuente MTurk en la evaluación del modelo pero no en su entreno. En caso de que las opiniones sean escritas de una manera muy diferente de una fuente a otra, este hecho dará lugar a un proceso más dificultoso para la detección de aquellas opiniones fraudulentas.

Tras haberse mirado la distribución de las opiniones y como esto va a afectar el posible resultado final de nuestro modelo, se ha pasado a analizar el comportamiento del tamaño de la opinión del usuario, sea negativa o positiva, en relación con la polaridad (Figura 4.6) o el grado de engaño de la opinión (Figura 4.7).

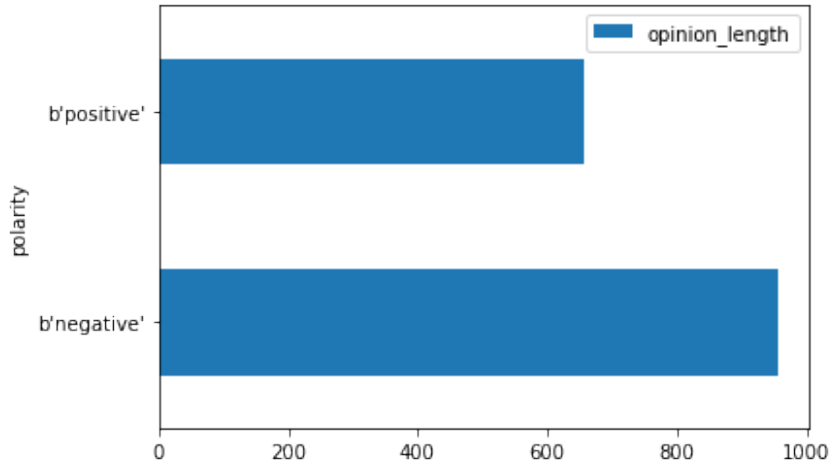


Figura 4.6: Relación entre el tamaño de la opinión y la polaridad de la misma

Como se puede ver el tamaño de las opiniones negativas suele ser aproximadamente 1.5 veces mayor que el tamaño de las opiniones positivas, hecho que a priori no influirá de forma directa en nuestro modelo, pero que se trata de un aspecto interesante de conocer. En cuanto a la relación entre el tamaño de la opiniones y el grado de engaño de las mismas, podemos ver que tienen una longitud aproximadamente similar, de media, siendo las opiniones fraudulentas más cortas que las opiniones sinceras. Este hecho es plausible ya que por lo general las opiniones sobre cualquier producto o servicio que sean fraudulentas suelen ser de un tamaño inferior a las opiniones sinceras ya que no ofrecen explicaciones tan detalladas y extensas como lo hacen los opinantes verídicos.

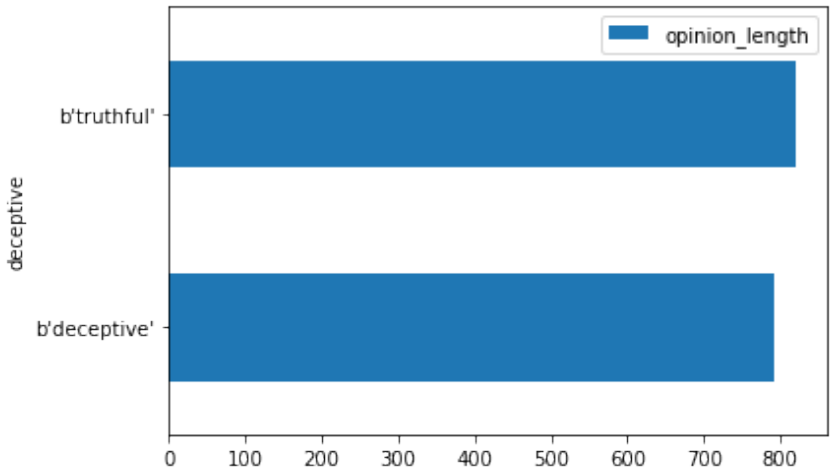


Figura 4.7: Relación entre el tamaño de la opinión y la clase engaño

También podemos ver en un grafo tipo violin (Figura 4.8 la relación entre el tamaño de las opiniones y la fuente desde la cual se ha opinado. De media, la fuente Web al tratar solo opiniones negativas, tiene un tamaño promedio superior al de los otros, hecho que confirma la veracidad de la Figura 4.6.

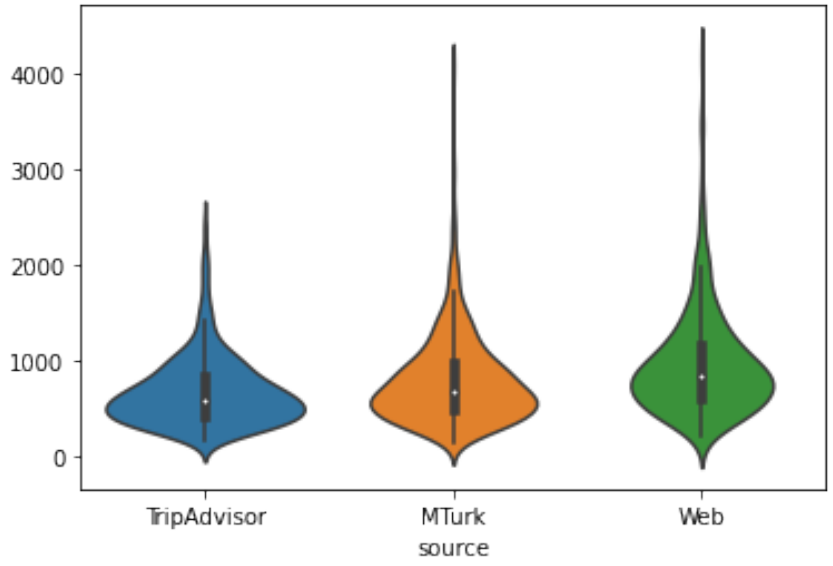


Figura 4.8: Relación entre el tamaño de la opinión y la fuente

Por último, vamos a utilizar la librería de WordCloud [1], una librería open-source que nos permite crear nubes de palabras sobre nuestro dataset. Su uso y los resultados obtenidos se pueden observar en la Sección A.1.

4.2. Preprocesamiento del *Deceptive Opinion Spam Corpus*

Tras la visualización de los datos, tendríamos que empezar a valorar la cualidad y cantidad de los datos con los cuales vamos a trabajar. El análisis de dichos datos

y la elección preliminar de estos se ha realizado en el capítulo anterior. Tras la elección del dataset más adecuado, dividiremos el proceso general en diferentes subetapas con la finalidad de obtener un dataset más limpio y más preciso para conseguir el objetivo planteado.

4.2.1. Limpieza de los datos

Es el proceso que permite identificar datos incompletos, incorrectos, duplicados y luego, sustituir, modificar o eliminar dichos datos. Esta fase es necesaria para prevenir que el modelo falle o tenga unos resultados deficientes. Los datos deben cumplir con las siguientes características:

- Exactitud: Deben ser íntegros, consistentes y densos.
- Integridad: Satisfacer las restricciones de integridad de los datos.
- Consistencia: Corrección de contradicciones entre los datos.
- Densidad: Número de valores omitidos sobre el número de valores totales.
- Unicidad: Evitar la duplicidad de los datos.

Para ello hemos aplicado algunas funciones propias de Panda [29] para la realización de esta tarea:

```
missing_values = ["n/a", "na", "--"]
df.isnull().sum()
df.isnull().values.any()
df[df.duplicated(keep=False)]
df = df.drop_duplicates()
```

Mediante esta secuencia de parámetros nos encargaremos de eliminar aquellos valores que sean de tipo 'extraño', es decir, 'NaN', 'Null', 'N/A', etc. Estos datos no nos ayudarían a resolver nuestra problemática, de hecho, podrían empeorar el resultado y tendríamos que modificarlos o sustituirlos por unos valores determinados. En nuestro caso concreto no se han encontrado registros de estos tipos y por lo tanto no se han tenido que tomar acciones para la modificación de los mismos.

A continuación, se ha comprobado también la redundancia de los datos, mirando a ver los posibles duplicados que haya en el dataset. En nuestro dataset se han encontrado múltiples ocurrencias de los mismos registros (Figura 4.9) y por lo tanto, se han tenido que descartar dichos registros para evitar una posible reiteración en los datos y mantener la consistencia e integridad de los datos.

	deceptive	hotel	polarity	source	text
803	truthful	omni	negative	Web	My daughter and I woke in the morning wanting ...
847	truthful	omni	negative	Web	The Omni was chosen for it's location whichwor...
853	truthful	omni	negative	Web	My daughter and I woke in the morning wanting ...
862	truthful	omni	negative	Web	The Omni was chosen for it's location whichwor...
995	truthful	affinia	negative	Web	I'd been searching for a cool, non-chain hotel...
1014	truthful	affinia	negative	Web	I'd been searching for a cool, non-chain hotel...
1085	truthful	monaco	negative	Web	Very disappointed in our stay in Chicago Monoc...
1109	truthful	monaco	negative	Web	Very disappointed in our stay in Chicago Monoc...

Figura 4.9: Valores duplicados del dataset

4.2.2. Reducción de los datos

La reducción de la dimensionalidad de los datos es el proceso de transformación de los datos de un espacio dimensional grande en un espacio dimensional pequeño tal que esta nueva representación mantenga las propiedades más determinantes de los datos originales. Trabajar con datos en un espacio dimensional grande implicaría un uso de cómputo y almacenamiento más grande y el análisis de estos datos con dimensiones grandes puede volverse laborioso desde el punto de vista computacional. Para nuestra problemática, partimos de un dataset con un conjunto de registros formado por cinco atributos: 'deceptive', 'hotel', 'polarity', 'source', 'text'; de los cuales, como ya hemos analizado en el capítulo anterior, solo los atributos 'deceptive' y 'text' nos ayudarían a modelar nuestro objetivo final, el de la detección de opiniones fraudulentas, ya que tenemos la opinión del usuario por un lado y la etiqueta del tipo de opinión por otro. Los otros atributos, aunque han sido útiles para hacer un análisis exploratorio inicial, como se ha podido ver en la sección anterior, no influyen directamente en la detección de opiniones fraudulentas y por lo tanto se procedería con su eliminación.

4.2.3. Transformación de los datos

La siguiente etapa en el proceso general del preprocesamiento de los datos, consta de una parte fundamental y obligatoria, alterar los datos de tal manera que se puedan trabajar con ellos de una forma consistente.

En nuestro caso, tras haber reducido la dimensionalidad de nuestro dataset, vamos a tener que trabajar con la opinión del usuario y la etiqueta de la opinión. Por lo tanto, al tener que trabajar con dos atributos de tipo 'String', es necesaria su conversión en una secuencia numérica para su uso posterior por parte del modelo de aprendizaje profundo. Las siguientes subetapas de este proceso consistirán en realizar esta transformación para poder manejar estos datos de una forma ideal y tener un modelo funcional.

4.2.4. Limpieza del texto

La primera subetapa consistiría en limpiar el texto de tal manera que no contenga palabras redundantes o que puedan afectar el comportamiento general del algoritmo que vamos a usar en la siguiente sección para transformar el texto en un vector numérico.

La limpieza del texto es un paso necesario, sobre todo dentro del ámbito del procesamiento de lenguaje natural. Mediante el uso del procesamiento de lenguaje natural, vamos a obtener que nuestro modelo entienda los contenidos de los documentos iniciales, en nuestro caso las opiniones de los usuarios, incluyendo las relaciones contextuales que se puedan dar dentro de cada documento. Por lo tanto, mediante este método se van a extraer ciertas características esenciales que permitan distinguir los diferentes documentos y con la ayuda de los cuales categorizaremos las diferentes opiniones. Sin el proceso de limpieza del texto, nuestro dataset es un cúmulo de palabras que nuestro modelo no va a entender.

Los documentos en lenguaje natural normalmente contienen palabras que solo aparecen una vez, también conocidos como 'Hapax Legomenon'. Incluyendo todas las palabras que aparecen en un dataset puede resultar en una cantidad excesiva de características, hecho no muy deseable, y por lo tanto, emplearemos unas técnicas de procesamiento de lenguaje natural para eliminar y reducir el tamaño de este espacio de características.

A continuación, vamos a ver los pasos más comunes en un proceso normal de limpieza de texto:

- Conversión del texto a minúsculas: Esto se realiza ya que ayudaría en el futuro proceso de preprocesamiento en cuanto al análisis y formateo del texto.
- Eliminación de la puntuación, números, múltiples espacios, etiquetas HTML y palabras cortas : Estas palabras son muchas veces innecesarias ya que no añadirían ningún valor o significado a nuestro modelo final.
- Eliminación de 'stop words' o palabras vacías: Se trata de palabras típicamente cortas y comunes tales como: 'at', 'is', 'the', etc. Este paso es importante probarlo y analizar los modelos finales con y sin este proceso, para darnos cuenta de si influye o no en el resultado final.
- Lematización y Stemming: Es el proceso por el cual se reduce una palabra a su raíz para evitar las variaciones de la misma. Más información en la Sección A.2.

```
1 Resultado del stemming para cualquier variación:
2 print(ps.stem('believe'))
3 print(ps.stem('believing'))
4 print(ps.stem('believed'))
5 print(ps.stem('believes'))
6 >> believ
```

Una vez explicado todo el proceso necesario para la limpieza procederemos a utilizar la librería Gensim [35], una librería open-source, utilizada para el procesamiento del lenguaje natural que aplica el uso del aprendizaje automático estadístico moderno. Está implementada en Python/Cython y ha sido diseñada principalmente para trabajar con colecciones de textos grandes utilizando transmisión de datos y algoritmos incrementales en línea, a diferencia de la mayoría de las otras librerías de aprendizaje automático que se enfocan solo en el procesamiento en memoria. Mediante el uso de esta librería hemos aplicado los filtros necesarios, anteriormente explicados para la limpieza del texto.

Tras este proceso, una última subetapa necesaria antes de la conversión del texto en un vector de características sería la de 'Tokenización'. Este proceso consiste en separar la cadena en una lista de palabras haciendo uso de expresiones regulares para identificar y separar dichas palabras. Por último, una vez acabado este proceso, convertiremos dichas lista de palabras en un tipo especial de objeto llamado TaggedDocument, cuyos dos parámetros serían: la lista de palabras de una opinión del usuario junto a un tag, que representaría el valor 'truthful' o 'deceptive' de la opinión, en caso de que se trate de una opinión fraudulenta o sincera (Figura 4.10).

```
TaggedDocument(['spent', 'wonder', 'night', 'the', 'amalfi', 'with',
'friend', 'the', 'even', 'recept', 'wa', 'veri', 'nice', 'and', 'th
e', 'staff', 'wa', 'veri', 'attent', 'breakfast', 'wa', 'plu', 'roo
m', 'were', 'cozi', 'and', 'veri', 'clean', 'would', 'not', 'hesit',
'back', 'again'], ['truthful'])
```

Figura 4.10: Ejemplo de un TaggedDocument

4.2.5. Transformación del texto

Una vez acabado con este proceso exhaustivo de limpieza del texto, utlizaremos el modelo de Doc2Vec [15] para la conversión del documento en un vector de características. Se trata de un modelo generalizado del Word2Vec [16]. Doc2Vec genera unos vectores distribuidos eficientes capaces de recoger las relaciones sintácticas y semánticas entre las palabras del documento.

Tal y como se puede leer en la Sección A.3, aunque el algoritmo PV-DBOW tiene varias características interesantes y que además, consume menos almacenamiento al no tener que guardar los vectores de palabras en memoria, se ha concluido que el modelo PV-DM es mucho más estable y superior en ciertas condiciones, y que normalmente obtiene resultados de última generación por si sólos. Esto se ha determinado en varias publicaciones que analizan su eficacidad para la modelización en el ámbito del análisis de sentimientos [22].

Para nuestro caso hemos decido lanzar el modelo con los siguientes parámetros (A.4):

```
1 dbow = Doc2Vec(dm=1, vector_size = 300, window = 5, min_count = 3,
2 negative=5, workers = cores, alpha=0.025, min_alpha=0.001)
```

Tras definir el modelo, lo entrenamos fijando cierta tasa de aprendizaje y finalmente inferimos los vectores de características de cada uno de nuestro párrafos tanto para el dataset de training como para el de testing:

```
from sklearn import utils

dbow.train(utils.shuffle([x for x in tqdm(train_dec.values)]),
total_examples= len(train_dec.values), epochs=200)

def vec_for_learning(model, input_docs):
    sents = input_docs
    targets, feature_vectors = zip(*[(doc.tags[0],
    model.infer_vector(doc.words, steps=50)) for doc in sents])
    return targets, feature_vectors

y_train, x_train = vec_for_learning(dbow, train_dec)
y_test, x_test = vec_for_learning(dbow, test_dec)
```

Una vez hecho esto, tenemos nuestros datos transformados y listos para ser utilizados en el modelo final de detección de opiniones fraudulentas.

4.3. Selección, modelización y entreno de modelos de detección de opiniones fraudulentas

En este apartado, realizaremos la parte de elección y creación de los diferentes modelos. Una vez que se hayan limpiado y preprocesado los datos que vamos a utilizar en nuestro modelo, definiremos los diferentes modelos que se encargarán de la detección de opiniones fraudulentas.

Como arquitectura elegida para nuestros modelos, nos basaremos en un tipo particular de RNN, ya que este tipo de red presenta ventajas peculiares sobre los CNN, tal y como se ha visto a lo largo del Capítulo 2. El tipo específico de RNN que vamos a utilizar es LSTM (Long short-term memory), por sus habilidades específicas de 'mirar hacia atrás', y mantener una cierta memoria a través del transcurso del tiempo. Este tipo de red se suele utilizar más a menudo para datos basados en series temporales. Aunque nuestros datos no dependen de forma continua uno del otro, como en una serie temporal, se debe recalcar el hecho de que nuestros datos presentan valores textuales. Es decir, nuestros modelos van a trabajar con opiniones que presentan un concepto similar al de los datos temporales: las palabras de un texto tienen cierta temporalidad en que la secuencia de las palabras determinan el significado del texto; y por lo tanto este es el principal aspecto por el cual nos hemos guiado en la elección de este tipo de RNN para nuestros modelos.

4.3.1. Primer modelo

Definimos y creamos el primer modelo basado en redes LSTM siguiendo la técnica del Autoencoder. Para ello, hemos separado previamente el dataset de entreno y el de evaluación. Para que estos datos puedan ser usados como entrada

en una red de este tipo necesitamos remodelarlos de la siguiente manera:

```
# reshape inputs for LSTM [samples, timesteps, features]
x_train = x_train.reshape(x_train.shape[0], 1, x_train.shape[1])
x_test = x_test.reshape(x_test.shape[0], 1, x_test.shape[1])
```

Ambos objetos, tanto los datos de entreno como los datos de evaluación, van a tener la forma adecuada para ser usada por redes LSTM: (número de registros, timesteps, número de características). Dentro de nuestro modelo, el 'número de registros' se corresponde al tamaño del dataset de entreno y el de la evaluación, el 'timesteps' sería un valor constante, en este caso siendo 1, y el 'número de características' se relacionaría con el tamaño del vector que representa cada opinión, parámetro que se ha definido en la sección anterior. El número de características representa la dimensionalidad de nuestro vector de opinión.

Emplearemos la librería de Keras [13] para crear las capas del modelo y los tensores de Keras necesarios, tal que nuestro modelo quedará de esta forma:

```
def autoencoder_model(X):
    inputs = Input(shape=(X.shape[1], X.shape[2]))
    L1 = LSTM(16, activation='relu',
              kernel_regularizer=regularizers.l2(0.00))(inputs)
    output = (Dense(X.shape[2])) (L1)
    model = Model(inputs=inputs, outputs=output)
    return model
```

Empezamos definiendo un objeto de tipo tensor de la forma:(timesteps = 1, número de características); este objeto se le pasa a la única capa LSTM que hemos definido, una capa de 16 nodos que tiene una función de activación (ReLU), que se encargará de emitir la entrada directamente en caso de que sea positiva. Esta es una de las funciones de activación predeterminadas para muchos tipos de redes neuronales. Se emplean regularizadores para aplicar sanciones a las capas durante la optimización. Por último, se define una capa de tipo Dense que recibirá como entrada la salida de la capa anterior LSTM, y tendrá como salida el tamaño de nuestro tipo de dato anterior, es decir, el número de características del vector de opinión. Con esto se crea un objeto de tipo 'Model', que recibe los datos de entrada y los datos que se deberán obtener como salida.

Vamos a hacer uso del método 'compile' sobre el objeto 'model', que hemos creado anteriormente, para configurar nuestro modelo fijando varios parámetros, tales como: el optimizador, la función objetivo, y una lista de métricas que van a ser evaluadas por el modelo durante el entreno y el evaluación. Posteriormente, entrenamos dicho modelo de la siguiente manera:

```
nb_epochs = 300
batch_size = 128

history = model.fit(x_train, x_train, epochs=nb_epochs, batch_size=
batch_size, validation_split = 0.2, verbose= 0).history
```

Al emplear la técnica del Autoencoder, tanto los datos de entrada como los datos que se pretenden obtener, tienen que ser los mismos para cumplir con los principios fijados por esta técnica.

Seguidamente, se utiliza el método 'evaluate', para obtener el valor de la pérdida y los distintos valores de las métricas, que anteriormente se hayan fijado al configurar el modelo.

Tras obtener estos valores, podemos crear varias gráficas y diagramas para ver la pérdida del modelo y la distribución de la misma en el entreno, la validación y la evaluación.

4.3.2. Segundo y tercer modelo

Una vez acabado el primer modelo, intentaremos mejorar nuestro modelo para que sea más eficaz. Esto lo hacemos de la siguiente manera:

```
def autoencoder_model(X):
    inputs = Input(shape=(X.shape[1], X.shape[2]))
    L1 = LSTM(32, activation='relu', return_sequences=True,
              kernel_regularizer=regularizers.l1(0.01))(inputs)
    L2 = Dropout(0.2)(L1)
    L3 = LSTM(4, activation='relu', return_sequences=False,

    kernel_regularizer=regularizers.l1(0.01))(L2)
    L4 = RepeatVector(X.shape[1])(L3)
    L5 = LSTM(4, activation='relu', return_sequences=True)(L4)
    L6 = LSTM(32, activation='relu', return_sequences=True,
    kernel_regularizer=regularizers.l1(0.01),
    activity_regularizer=regularizers.l2(0.01))(L5)
    output = TimeDistributed(Dense(X.shape[2]))(L6)
    model = Model(inputs=inputs, outputs=output)
    return model
```

Procederemos a añadir más capas LSTM, empezando con una capa de 32 nodos, e iremos reduciendo la dimensionalidad de los datos para que se puedan representar con una capa de 4 nodos, y luego iremos recuperando poco a poco la dimensionalidad original. De forma similar al modelo anterior, utilizaremos regularizadores en algunas capas para poner ciertas penalizaciones a los parámetros de las capas para mejor la distribución de la pérdida. Otro método que emplearemos con una finalidad regularizadora es la capa Dropout, que nos ayudaría a evitar el sobreajuste del modelo.

El tercer modelo que hemos creado se asemeja mucho al segundo y representa solo un paso más hacia el objetivo de obtener unos mejores resultados. En este modelo lo que se ha hecho fue añadir aún más capas, pasando de una dimensionalidad original de 128 nodos e ir bajando progresivamente tal que así:

```
def autoencoder_model(X):
    inputs = Input(shape=(X.shape[1], X.shape[2]))
    L1 = LSTM(128, activation='relu', return_sequences=True,
```

```
4                 kernel_regularizer=regularizers.l2(0.00))(inputs)
5     L2 = LSTM(32, activation='relu', return_sequences=True)(L1)
6     L3 = LSTM(4, activation='relu', return_sequences=False)(L2)
7     L4 = RepeatVector(X.shape[1])(L3)
8     L5 = LSTM(4, activation='relu', return_sequences=True)(L4)
9     L6 = LSTM(32, activation='relu', return_sequences=True)(L5)
10    L7 = LSTM(128, activation='relu', return_sequences=True)(L6)
11    output = TimeDistributed(Dense(X.shape[2]))(L7)
12    model = Model(inputs=inputs, outputs=output)
13    return model
```

Estos dos últimos modelos van a utilizar las mismas configuraciones, métricas, proceso de entreno y predicción, definidos en la Sección 4.3.1 para el primer modelo.

Se ha llegado a definir otro modelo, que sigue la misma estructura que el tercer modelo, definido en la Sección 4.3.2 pero que, en vez de utilizar capas LSTM emplea capas de otro tipo de RNN, llamado GRU. Este modelo se ha definido con el objetivo de mejorar los resultados iniciales de los modelos previos.

4.4. Métricas

En este apartado presentamos las métricas y métodos que permiten evaluar la calidad de nuestros modelos. Dentro del proceso de evaluación de un modelo la elección de las métricas es muy importante ya que va a influir en la eficacia de los algoritmos del modelo para medir y comparar los resultados.

En la clasificación binaria, tal como en nuestro caso, hay una amplia variedad de métricas que se pueden emplear. Aunque, también dependen del ámbito en el que el modelo se emplea. Para modelos del ámbito de la computación científica se suelen utilizar las métricas de Precisión y Recall mientras que en el ámbito médico, se emplea más las métricas de Sensibilidad y Especificidad [2].

A continuación definiremos algunas métricas en las que nos basaremos a la hora de tomar decisiones para comprobar la calidad de nuestro modelo:

- Matriz de confusión: Es una de las métricas más intuitivas y fácil de entender para encontrar cuánto de 'correcto' se comporta nuestro modelo y la precisión del mismo. Se trata de una tabla de dos dimensiones: 'Verdadero' y 'Predicho' y una serie de clases para ambas dimensiones: positivas y negativas. En nuestro caso, estas clases se corresponden con las opiniones sinceras y fraudulentas. Esta métrica por sí sola no es una medida de la precisión de un modelo, pero las otras métricas se basan en los valores de la misma. La matriz de confusión (Figura 4.11) tiene los siguiente términos asociados:

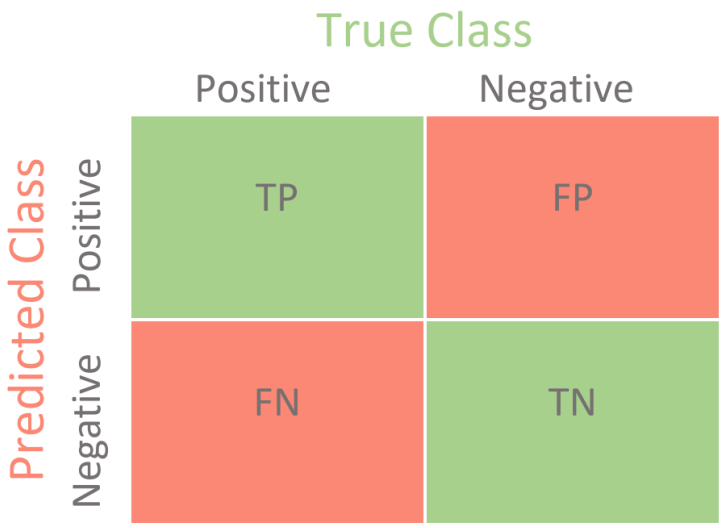


Figura 4.11: Diagrama general de la Matriz de Confusión

- TP (Verdaderos Positivos): Son aquellos valores verdaderamente positivos que se hayan clasificado como positivos. En nuestro caso, aquellas opiniones sinceras que se hayan clasificado como sinceras.
- TN (Verdaderos Negativos): Son aquellos valores verdaderamente negativos que se hayan clasificado como negativos. En nuestro caso, aquellas opiniones fraudulentas que se hayan clasificado como fraudulentas.
- FP (Falso Positivo): Son aquellos valores verdaderamente negativos que se hayan clasificado como positivos. En nuestro caso, aquellas opiniones fraudulentas que se hayan clasificado como sinceras.
- FN (Falso Negativo): Son aquellos valores verdaderamente positivos que se hayan clasificado como negativos. En nuestro caso, aquellas opiniones sinceras que se hayan clasificado como fraudulentas.

Para nuestro objetivo, la métrica ideal sería maximizar el número de TN o que es lo mismo minimizar el número de FN. Aunque, este sea el objetivo principal, también nos interesaría que haya más valores clasificados como TP frente a los FP.

- Tasa de verdadero positivo o Recall: Esta métrica mide cuántas observaciones positivas se hayan clasificado como positivas de todas las observaciones verdaderamente positivas. En un problema de clasificación binaria no balanceado, la fórmula para el cálculo de esta métrica será la siguiente: $Recall = (TP/TP + FN)$. En nuestro caso, esta métrica significaría la probabilidad de que una opinión salga como sincera de todas las opiniones sinceras que se hayan clasificado o bien como sinceras o como fraudulentas. Está métrica nos ofrece información respecto a los FN, así que nos interesaría para minimizar los FN.

Esta métrica normalmente no se usa por sí sola sino que se emplea junto a otras métricas.

- Valor predictivo positivo o Precisión: Esta métrica mide cuántas observaciones positivas se hayan clasificado como positivas de todas las observaciones tanto positivas como negativas clasificadas como positivas. En un problema de clasificación binaria no balanceado, la fórmula para el cálculo de esta métrica sería la siguiente: $Precisin = (TP/TP + FP)$. Para nuestro modelo, esta métrica significaría la probabilidad de que una opinión salga como sincera de todas las opiniones sinceras o fraudulentas que se hayan clasificado como sinceras. Por lo tanto, esta métrica nos interesa para minimizar los FP. Tal como la métrica anterior de Recall, esta métrica no se suele usar por sí sola sino que se emplea junta a otras métricas.

- Tasa de verdadero negativo o Specificity: Se define como la proporcion de observaciones negativas que se hayan clasificado como negativas entre todas las observaciones clasificadas como positivas o negativas. En un problema de clasificación binaria no balanceado, la fórmula para el cálculo de esta métrica sería la siguiente: $Specificity = (TN/TN + FP)$. Para nuestro modelo, esta métrica significaría la probabilidad de que una opinión salga como verdaderamente fraudulenta de todas las opiniones fraudulentas que se hayan clasificado como sinceras o fraudulentas.

  Un valor más alto de Specificity significaría una mejor tasa de TN y una menor tasa de FP.

- F-score: Se trata de una métrica que combina la Precisión y la Recall en una sola métrica. Normalmente, cuanta mayor sea la puntuación de esta métrica mejor se comporta nuestro modelo. Para nuestro problema de clasificación binaria, la fórmula para el cálculo de esta métrica sería la siguiente:

$$F_{beta} = (1 + \beta^2)\frac{\text{precision} * \text{recall}}{\beta^2 * \text{precision} + \text{recall}} \tag{4.1}$$

  Normalmente el parámetro β de esta métrica se elige en función de qué métrica queremos que tenga más peso, Precisión o Recall. Para un beta con valor `1`, nuestra métrica tendrá peso igual tanto para Precision y Recall, es decir, no vamos a preferir ninguna métrica sobre la otra. En caso de que beta tenga valor 2, esto prioriza el Recall al doble sobre Precisión. Para un valor beta entre 0 y `1`, el valor de Precisión tendrá mayor peso que Recall.

  Vamos a utilizar la métrica `F1`, ya que queremos buscar un cierto balance entre las métricas Precision y Recall y dado que esta métrica se puede utilizar en cualquier problema de clasificación binaria, sobre todo en problemas donde hay una gran distribución desigual, como es nuestro caso, puede que sea una muy buena métrica a utilizar.

En la determinación de las métricas emplearemos la librería Scikit [39], una librería de aprendizaje automático hecha Python [34]. Emplea diferentes algoritmos para la clasificación, regresión y clustering de modelos de aprendizaje profundo. Es simple y eficiente para el análisis de predicción de datos. Se puede usar en diferentes contextos y es compatible con otras librerías que estuvimos usando, tales como NumPy [28] o Pandas [29].

El módulo disponible de Scikit, llamado **sklearn.metrics** [40], implementa varias funciones que permiten evaluar el error de predicción. Utilizaremos los siguientes métodos (Tabla 4.1).

Método	***Descripción***
precision_score	Calcula el valor de la Precisión. Esta métrica determina la habilidad del clasificador de no etiquetar de forma errónea un positivo como un negativo.
recall_score	Calcula el valor de Recall. Esta métrica determina la habilidad del clasificador de detectar todas las muestras positivas.
fbeta_score	Computa el valor de Fbeta. Esta puntuación es la media armónica ponderada entre Precisión y Recall, alcanzando su valor óptimo en 1 y su peor valor en 0.
confusion_matrix	Métrica que computa la matriz de confusión de nuestro modelo tal como la de la Figura 4.11, y que nos ayuda de una forma más gráfica evaluar el Accuracy de la clasificación
classification_report	Método que construye un informe en formato texto mostrando las principales métricas de clasificación.
accuracy_score	Calcula la puntuación de Accuracy de un modelo.
precision_recall_curve	Calcula pares de valores (Precision, Recall) para diferentes umbrales.
roc_curve	Calcula la curva ROC, diagrama visual que muestra la capacidad de un sistema clasificador binario, a medida que varía su umbral de discriminación.

Tabla 4.1: Métodos empleados para la evaluación del modelo

4.5. Experimentación y resultados

En esta sección nos basaremos en las métricas anteriormente mencionadas para evaluar nuestro modelo. Utilizaremos los métodos de: *Classification report*, *Roc curve*, *Precision-recall curve*, entre otros descritos anteriormente, para comparar los valores obtenidos para diferentes umbrales y para los diversos modelos que hemos creado.

Tanto el código completo como los resultados obtenidos se podrán consultar en el siguiente repositorio: Github(TFG-DL-Autoencoder) [44].

El proyecto ha sido realizado en un ámbito con las siguientes características (Tabla 4.2):

Máquina empleada	Procesador	RAM	Almacenamiento	Sistema operativo	Software empleado
Asus-GL552JX (x86_64)	Intel i7-4720HQ (8 x 2.1 MHz)	8 GB	128 GB SSD (SAMSUNG MZNLF128)	Ubuntu 20.04.1 LTS	JupyterLab 3.0.7 Python 3.8.5 Keras 2.4.3

Tabla 4.2: Características del ámbito empleado para el desarrollo del trabajo

Empezaremos viendo cómo se comportan los diferentes modelos definidos anteriormente en la Sección 4.3 mediante la curva ROC (Tabla 4.12).

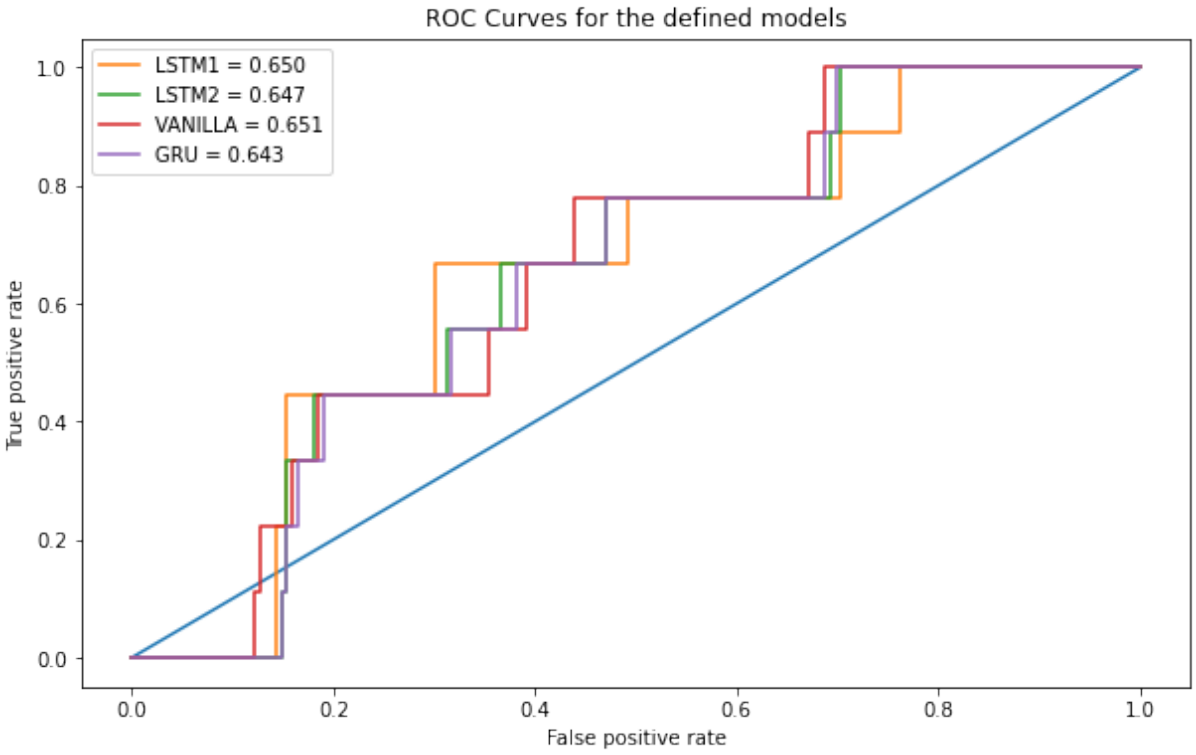


Figura 4.12: Diagrama de las curvas ROC para los diferentes modelos definidos

Tal como se ha dicho anteriormente es una medida de rendimiento para los problemas de clasificación en varios valores de umbral. Representa una curva de probabilidad que indica la capacidad del modelo de distinguir entre clases (sinceras y fraudulentas). En nuestro caso, los cuatro modelos definidos tienen una capacidad bastante similar en la evaluación del modelo.

Por lo tanto, hemos logrado crear diferentes modelos que se comportan de manera similar y cuyas capacidades de distinguir y clasificar las opiniones de forma correcta(clasificar las opiniones sinceras como sinceras y las opiniones fraudulentas como fraudulentas) se han centrado en torno al 64.3 %-65.1 % .

Otros aspectos que habrían que considerar para evaluar los diferentes modelos serían: el tiempo de entreno, época empleadas, algoritmo de optimizacion, función de pérdida utilizada, arquitectura empleada, para cada uno de los modelos (Tabla 4.3).

Se observa como, dada las diferentes condiciones mencionadas anteriormente, los tiempos suelen variar bastante. El menor tiempo de entreno se ha sacado utilizando el segundo modelo, definido en la Sección 4.3.2. También podemos observar que el modelo definido con capas GRU tiene un menor tiempo de entreno frente al mismo modelo definido(empleando las mismas condiciones) con las capas LSTM, modelo explicado en la Sección 4.3.2.

Modelo	**Tiempo de entreno**	**Épocas usadas en el entreno**	**Algortimo optimización**	**Función de perdida**	**Arquitectura empleada**
Vanilla LSTM	71.10 s	300	adamx	MAE	16 nodos (kernel_regularizer)
Basic LSTM	33.13 s	300	adamx	MAE	72 nodos (Dropout + activity_regularizer + kernel_regularizer)
Extended LSTM	63.33 s	300	adamx	MAE	328 nodos (kernel_regularizer)
GRU	55.21 s	300	adamx	MAE	328 nodos (kernel_regularizer)

Tabla 4.3: Descripción de los diferentes modelos definidos

Nos basaremos en el uso del diagrama Precision-Recall, ya que se trata de una métrica muy útil sobre todo, al realizar predicciones en clases desbalanceadas. Para nuestro caso, al trabajar principalmente sobre la optimización de la clase minoritaria o negativa, tendremos un bajo porcentaje de Precisión negativa(Precisión de la clase minoritaria) o 'False Discovery Rate' y un porcentaje más alto de Recall sobre la clase negativa(Specificity). Para nuestro caso(Figura 4.13), los modelos se comportan de una forma similar. El único modelo que destacaría un poco sería el segundo modelo definido en la Sección 4.3.2. Con esto se intenta que, al tratarse de un clasificador binario desbalanceado, la Precisión negativa(False Discovery Rate) sea lo más bajo posible para evitar que opiniones fraudulentas sean clasificadas como sinceras y por otro lado, con el alto porcentaje de Recall de la clase minoritaria, se intenta maximizar el número de anomalías clasificadas de forma correcta, es decir, como opiniones fraudulentas.

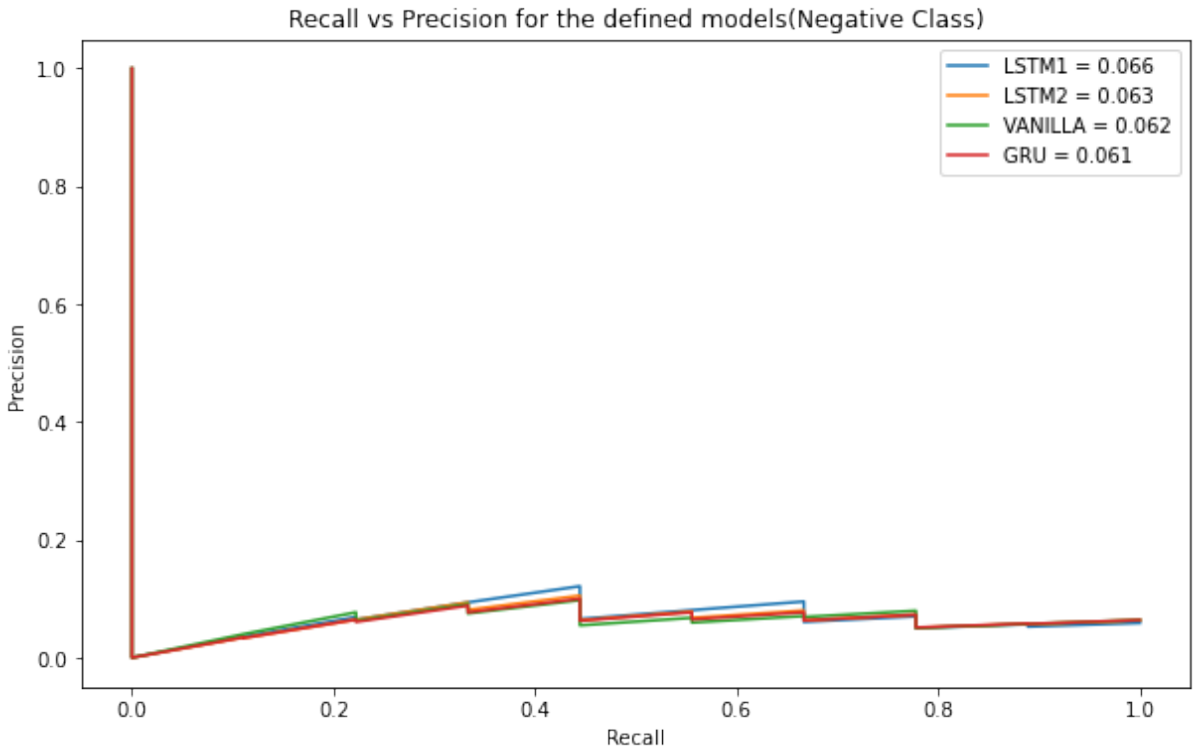


Figura 4.13: Diagrama de las curvas Precision-Recall para los diferentes modelos definidos (Clase Negativa)

Una vez visto, según los diagramas anteriores, los comportamientos que presentan los diferentes modelos, nos centraremos en la elección de uno o varios umbrales o thresholds.

El criterio de elección del threshold va a ser uno arbitrario, ya que no tenemos(a priori no podemos saberlo), el threshold ideal para poder distinguir entre una clase y la otra. Por lo tanto, lo que se ha realizado fue la elección de múltiples umbrales y la visualización de los resultados que presentan los diferentes modelos creados según varias métricas (Tabla 4.4). Las métricas que se hayan elegido y que se muestran en la tabla son las siguientes: **P**: Precision, **R**: Recall, **Fb**: F-beta.

Modelo\ Umbral	**0.4**			**0.65**			**0.7**			**0.8**		
Métricas	**P**	**R**	**Fb**	**P**	**R**	**Fb**	**P**	**R**	**Fb**	**P**	**R**	**Fb**
Vanilla LSTM	0.52	0.52	0.08	0.52	0.62	0.42	0.52	0.59	0.48	0.47	0.45	0.46
Basic LSTM	0.52	0.51	0.07	0.52	0.60	0.35	0.52	0.63	0.44	0.47	0.43	0.45
Extended LSTM	0.52	0.52	0.08	0.52	0.63	0.39	0.51	0.56	0.45	0.47	0.44	0.46
GRU	0.52	0.52	0.09	0.52	0.61	0.41	0.52	0.57	0.46	0.47	0.44	0.46

Tabla 4.4: Métricas de los diferentes modelos definidos para ciertos umbrales

Por último, vamos a presentar el error de reconstrucción(Figura 4.14) de los da-

tos del segundo modelo creado(detallado en la Sección 4.3.2). Se puede concluir que el modelo es capaz, con cierta eficacidad, de diferenciar algunas opiniones fraudulentas frente a las opiniones sinceras. Establecemos un valor de threshold arbitrario para poder mostrar la matriz de confusión del modelo(Figura 4.15).

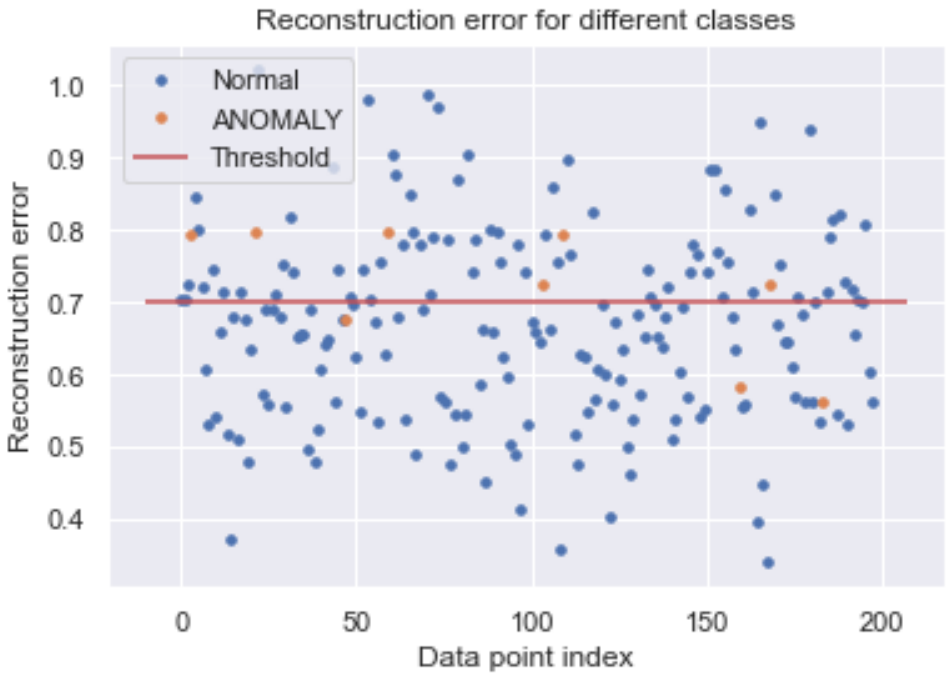


Figura 4.14: Error de reconstrucción para las diferentes clases del modelo

Los resultados que se muestran en la matriz de confusión indican que, el modelo ha sido capaz de clasificar de forma correcta seis de las nueve opiniones fraudulentas. Esto indicaría un porcentaje de acierto del 66 % para las opiniones fraudulentas mientras que para las opiniones sinceras, el porcentaje de acierto estaría en torno al 60 %.

Aunque el objetivo principal ha sido el de la detección de opiniones fraudulentas, también nos interesa que el porcentaje de acierto de las opiniones sinceras sea un valor íntegro.

Al analizar más en detalle el dataset que se ha concluido ser el más óptimo, dentro de los datasets analizados, se ha descubierto que podría presentar ciertas deficiencias que hacen que nuestro modelo no se comporte de la mejor forma posible. Estas deficiencias están relacionadas con las opiniones de los diferentes usuarios, datos necesarios, que están directamente implicados en el proceso de entreno de nuestro modelo. El primer aspecto determinante fue que, las opiniones de los usuarios podrían no presentar tantas palabras o secuencias de palabras distintivas, para que el modelo sea capaz de determinar y clasificar la opinión en la categoría correcta. Otro aspecto esencial fue el hecho de que, las distintas opiniones de los usuarios provienen de tres fuentes distintas: TripAdvisor, Web, MTurk. Nuestro modelo se entrenó con las opiniones de los usuarios obtenidas de las dos

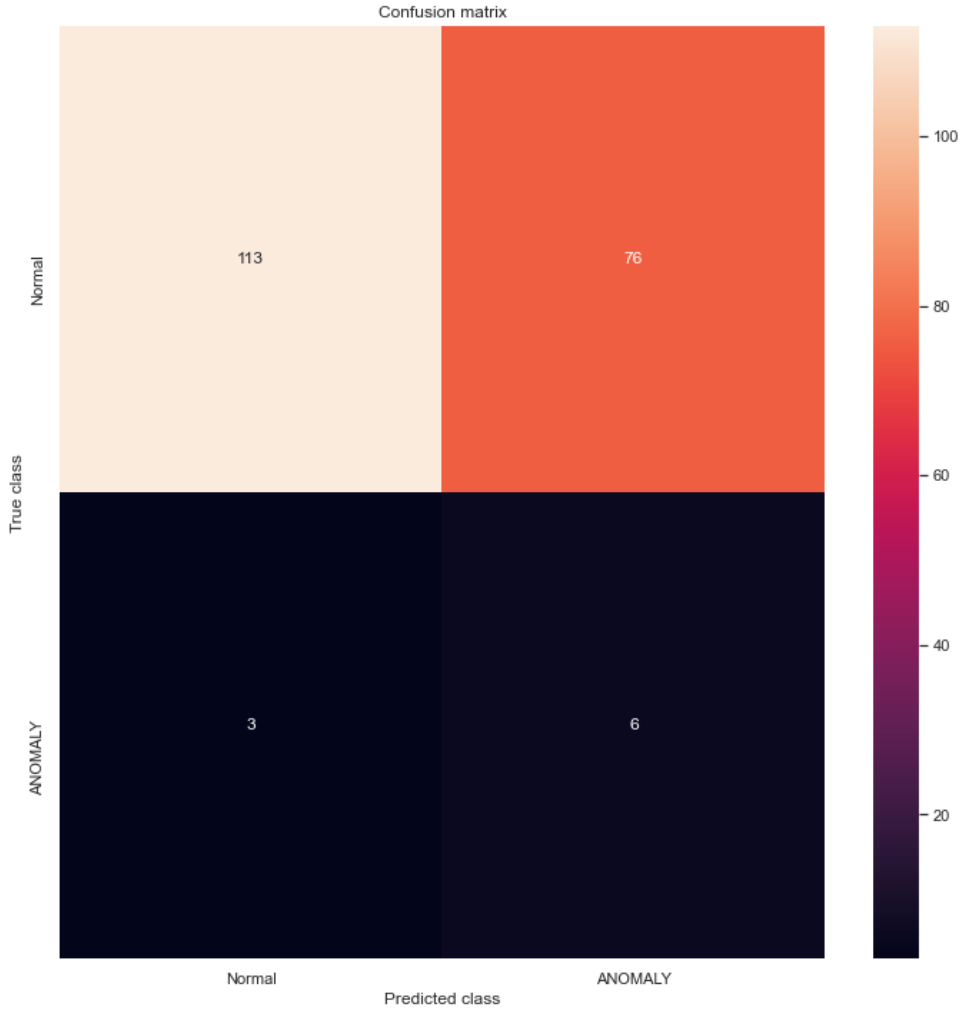


Figura 4.15: Matriz de confusión del segundo modelo LSTM definido

primeras fuentes, ya que estas fuentes solo presentan opiniones sinceras, mientras que la evaluación del mismo se realizó usando datos de las tres fuentes. La fuente MTurk, al presentar solo opiniones fraudulentas, no se ha utilizado en el entreno del modelo. Al redactar una opinión en una de estas plataformas anteriormente mencionada, la forma de escribir, el vocabulario empleado y otros aspectos podrían cambiar de forma drástica. Teniendo esto en cuenta, este aspecto podría ser otra de las deficiencias anteriormente mencionadas, que hayan afectado el rendimiento general del modelo.

Tras los resultados obtenidos anteriormente, podemos argumentar que se ha encontrado un modelo capaz de distinguir con cierta eficacidad las opiniones fraudulentas dentro de un dataset de opiniones mixtas(sinceras y fraudulentas) de usuarios reales.

Capítulo 5

Conclusiones y líneas futuras

En este capítulo presentamos las conclusiones obtenidas a partir del desarrollo del proyecto basado en redes de aprendizaje profundo centrado en la técnica no supervisada del Autoencoder y las líneas futuras del trabajo.

5.1. Conclusiones

Al finalizar el proyecto hemos llegado a las siguientes conclusiones:

- Se ha desarrollado un modelo capaz de identificar con cierta eficacidad, tal como presentamos en los resultados, aquellas opiniones fraudulentas dentro de un dataset de opiniones mixtas, tanto opiniones sinceras como fraudulentas, de diferentes usuarios reales.
- Tras el estudio y el empleo de las RNNs con sus variantes LSTM y GRU, se puede afirmar que nos ayudaron a desarrollar con éxito nuestro modelo siguiendo la arquitectura del Autoencoder para la detección de anomalías. Se ha tomado la decisión de emplear estas variantes de RNN, LSTM y GRU, ya que presentan ciertas características constitutivas que la hacen más propicias para trabajar, en algunos casos, con secuencias de texto.
- El conjunto de datos que hemos concluido que es el óptimo, dentro de los datasets analizados, podría presentar ciertas deficiencias que hacen que nuestro modelo no se comporte de la mejor forma posible.
- Las tecnologías del Deep Learning con sus diferentes tipos de redes neuronales, son adecuadas para la identificación de anomalías presentes en las opiniones, en una multitud de ámbitos, tal como en el sector turístico.
- El Autoencoder es una herramienta tecnológica emergente, que se utiliza cada vez más a menudo en varios sectores teniendo múltiples usos. Esta herramienta tiene un amplio uso en el procesamiento de lenguaje natural(NLP) y tal como se ha visto en nuestro modelo, muestra ciertas habilidades en el campo de la detección de anomalías. Esto se da por sus características intrínsecas de aprender aquellas características comunes de los datos y diferenciar aquellos rasgos particulares como anomalías.

- Las métricas que se han empleado nos permitieron evaluar la calidad de nuestro modelo. Los resultados obtenidos indican que nuestro modelo tiene una capacidad o efectividad de alrededor del 65 %. Esto hace que no se trate de un modelo especialmente bueno, sino un modelo aceptable que cumple con el objetivo inicial propuesto.

5.2. Líneas futuras

A continuación se expondrán algunos aspectos que potencialmente serán útiles para continuar con nuestro trabajo:

- Nuestro modelo se ha basado en el uso de la tecnología más predominante en este campo, y la que se haya considerado la más adecuada para cumplir con el objetivo propuesto. Dado el continuo crecimiento de estas tecnologías y el uso de la misma en el sector del NLP, no se podría descartar el desarrollo de modelos con la ayuda de redes generativas antagónicas (GANs) u otras técnicas relevantes.

- Otros aspectos que interesaría investigar en una línea futura serían: (1). El uso de un dataset con mayores dimensiones, y evaluar el desempeño de los modelos, (2). El uso de un dataset con una proporción de datos normales más extensa frente a la proporción de datos anómalos.

- Dados los beneficios que puede generar el uso de este tipo de modelo en múltiples ámbitos, se prevé la posible implementación de una aplicación centrada en este modelo. De esta aplicación se sacaría como provecho que, al introducir una nueva opinión, el modelo sea capaz de clasificarla como una opinión normal o anómala. De esto se pueden beneficiar múltiples categorías del sector hostelero, cuya actividad económica se ve determinada, aunque puede que de forma indirecta, por las opiniones que dejan los usuarios sobre sus diferentes productos o servicios.

Capítulo 6

Conclusions and future improvements

In this chapter we will describe the conclusions that have been obtained from the development of the project based on Deep Learning networks focused on the unsupervised technique of the Autoencoder and future lines of work.

6.1. Conclusions

At the end of the project we have reached the following conclusions:

- A model has been developed, capable of identifying with some efficiency, as we presented in the results, those fraudulent opinions within a dataset of mixed opinions, both sincere and fraudulent opinions, of different real users.
- After studying and using the RNNs with their LSTM and GRU variants, we may claim that they helped us to successfully develop our model following the Autoencoder architecture for the anomaly detection. The commitment to use these RNN variants, LSTM and GRU has been taken, as they have certain constitutive characteristics that make them more advantageous while working, in some cases, with text sequences.
- The dataset that we have concluded to be the most optimal, within the analyzed datasets, could present certain deficiencies that can make our model not to behave in the best possible way.
- Deep Learning technologies with their different types of neural networks, are suitable for the identification of anomalies that can be found in opinions, in a substantial amount of fields, such as in the tourism sector.
- The Autoencoder is an emerging technological tool, which is used more and more often in various sectors having numerous uses. This tool is widely used in natural language processing (NLP) and, as it has been seen in our model, shows certain skills in the field of anomaly detection. This is due to its intrinsic characteristics of learning those common characteristics of the data and discerning those particular features as anomalies.

- The metrics that have been used allowed us to evaluate the quality of our model. The results obtained indicate that our model has a capacity or effectiveness of around 65 %. This means that it is not a particularly good model, but rather an acceptable model that meets the initial objective proposed.

6.2. Future improvements

Hereafter are some aspects that will potentially be useful in order to refine our project:

- Our model has depended on the use of the most prevailing technology in this field, and the one that has been considered to be the most suitable in order to achieve the proposed objective. Given the continuous growth of this technologies and its use in the NLP area, the development of models with the help of generative adversarial network (GANs) or other relevant techniques can not be ruled out.
- Other aspects that would be interesting to explore in a future line of work would be: (1). The use of a dataset with larger proportions, and evaluate the performance of the models, (2). Using a dataset with a larger ratio of normal data to the ratio of anomalous data.
- Given the benefits that the use of this type of model can generate in multiple areas, the possible implementation of an application focused on this model is foreseen. The benefits obtained from this application would be that, when entering a new opinion, the model can classify it as a normal or an abnormal opinion. This can benefit multiple categories of the hospitality industry, whose economic activity is determined, although perhaps indirectly, by the opinions that the users leave about their different products or services.

Capítulo 7

Presupuesto

En este capítulo se presenta una estimación sobre el coste de la realización de este proyecto. También se presenta un coste hipotético de la implementación de este modelo en un entorno real.

7.1. Presupuesto de recursos humanos

En este apartado se muestra de forma desglosada el coste de los recursos humanos que ha sido invertido en el desarrollo del proyecto.

Tarea	***Cantidad de horas empleadas***	***Coste bruto por hora***	***Coste total***
Estudios relacionados al Machine Learning/Deep Learning	20	14€	480€
Estudio de las diferentes arquitecturas	15	14€	210€
Búsqueda y análisis de los conjuntos de datos adecuados para nuestro objetivo	18	14€	252€
Desarrollo del módulo de preprocesamiento y transformación de los datos para nuestro modelo	10	14€	140€
Visualización de los datos	8	14€	112€
Aprendizaje de las diferentes librerías y módulos a usar en la aplicación	25	14€	350€
Elección y creación del modelo	20	14€	280€
Documentación de la aplicación	10	14€	140€
Pruebas y tests de la aplicación	22	14€	308€
Ajustar los hiperparámetros del modelo	18	14€	252€
Total:	**166 horas**		**2533€**

Tabla 7.1: Presupuesto Personal

7.2. Presupuesto material

En este apartado se calculan los costes de los distintos componentes necesarios para el desarrollo de la aplicación.

Elemento	***Coste total***
Portátil	950€
Librerías / Módulos externos	0€
IDE / Editor	0€
Total:	**950€**

Tabla 7.2: Presupuesto de los materiales necesarios para el desarrollo del proyecto

7.3. Presupuesto para la implementación del proyecto

En caso de una hipotética ocasión en el que este modelo se quiera emplear en un entorno real, cabe decir que este modelo se podría utilizar dentro del sector hotelero para la detección de opiniones fraudulentas. Para ello, los componentes materiales mínimos necesarios serían las siguientes:

Elemento	***Cantidad***	***Coste***	***Coste total***
Terminales	5	650€	3250€
Servidor	1	600€	600€
Total:			**3850€**

Tabla 7.3: Presupuesto de los materiales necesarios para el supuesto caso

Cabe destacar que a parte de un servidor físico, también se podría emplear un servicio en la nube como el que ofrece Google Cloud [17]. Se trata de un servicio que propone poder de cómputo para modelos de aprendizaje automático, visualización de gráficos y computación científica. Esto nos libraría del hecho de tener un servidor físico y por lo tanto, basarnos en los servicios virtuales de Google.

Elemento	***Cantidad***	***Coste***	***Coste total***
Terminales	5	650€	3250€
Servidor	1	1609€/año	1609€/año
Total:			**4859€**

Tabla 7.4: Presupuesto de los materiales necesarios para el supuesto caso haciendo uso del Google Cloud

Aunque este servicio ofrecerá una mejora escalabilidad de la red general y presentará un tiempo de respuesta mejor, es recomendable el uso de la red de servidor local para garantizar un servicio estable y continuo. Con esto se logra tener un control general y absoluto de los datos.

7.4. Presupuesto final

El presupuesto final teniendo en cuenta el presupuesto del personal humano y utilizando el presupuesto de los materiales necesarios para la realización del modelo, sería el siguiente:

Elemento	***Coste total***
Coste humano	2533€
Coste de los materiales	950€
Total:	**3483€**

Tabla 7.5: Presupuesto Final

Apéndice A

Métodos y tecnologías complementarias

A.1. Uso de la librería WordCloud para la exploración de los datos

Procederemos a crear un WordCloud con las palabras más frecuentes obtenidas del dataset (Figura A.1). Como podemos ver la mayoría de las palabras tienen una relación clara muy directa con el ámbito de nuestro dataset: 'Chicago', 'room', 'stay', 'night', 'service', etc. Esto tiene mucho sentido ya que al tratarse de opiniones sobre hoteles del área de Chicago tiene interpretación que muchas palabras sean de este ámbito.

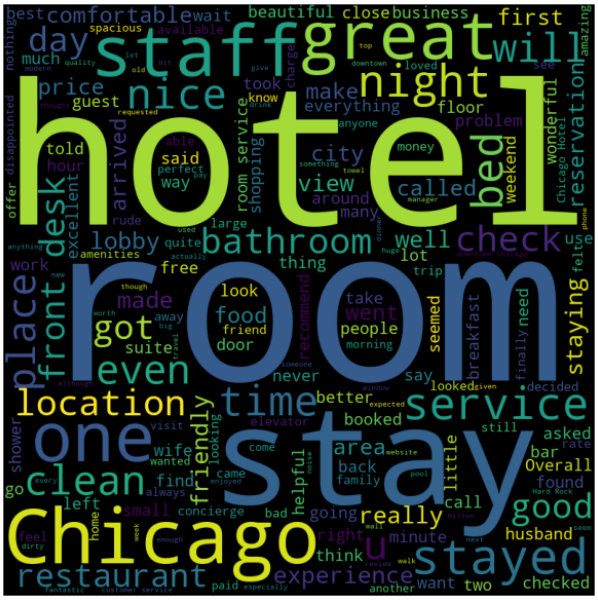


Figura A.1: WordCloud con las palabras más frecuentes del dataset

Luego, mirando la comparación entre el WordCloud de las opiniones sinceras y las opiniones fraudulentas (Figura A.2, Figura A.3), podemos concluir que las palabras más frecuentes no distinguen directamente entre una clase y la otra. Hay algunas palabras más frecuentes, entre las opiniones fraudulentas, que se repiten más que en las opiniones sinceras, tales como: 'experience', 'reservation', 'arrived';

pero este hecho no determina completamente que estas palabras u otras palabras que aparezcan con más frecuencia en las opiniones fraudulentas, precisan directamente el tipo de una opinión, sincera o fraudulenta. Esto hecho probablemente se dará con algunas palabras particulares y alguna secuencia de estas palabras. Es decir, una opinión se determina que es una opinión fraudulenta si contiene unas palabras específicas con o sin una secuencia bien establecida.

Figura A.2: WordCloud de opiniones sinceras

Figura A.3: WordCloud de opiniones fraudulentas

A.2. Lematización y Stemming

La diferencia fundamental entre los dos procesos es que el stemming corta el final de la palabra sin tener en consideración el contexto en el que se halla esta palabra, mientras que el proceso de lematización examina el contexto en el que se encuentra la palabra y recorta la palabra a su raíz siguiendo la definición de un diccionario predefinido. Por lo tanto, el compromiso entre los dos métodos sería que el stemming gana en rapidez de procesamiento mientras que la lematización tienen mejores resultados en la precisión del procesamiento de los datos.

A.3. Doc2Vec y sus principales técnicas de conversión

El objetivo de Doc2Vec es crear una representación numérica de un documento, independientemente de su longitud. Pero a diferencia de las palabras, los documentos no vienen en estructuras lógicas como las palabras, por lo que hay que encontrar otro método. El concepto fundamental que lo diferencia de los otros algoritmos, es el uso de otro vector llamado Paragraph ID, es decir añadir otro parámetro extra que permita identificar al documento. Además de entrenar los vectores de palabras, se aprende el vector de documento. Este concepto es una breve extensión del modelo CBOW utilizado en los modelos Word2Vec pero en vez de utilizar sólo palabras de un contexto para predecir la siguiente palabra, se emplea un vector de características, único para cada documento. Hay dos técnicas fundamentales de este modelo para obtener el vector del documento:

- PV-DM: Es una técnica similar al método CBOW del modelo Word2Vec, que intenta predecir el embedding de la palabra destino basándose en las palabras del contexto junto al Paragraph Id. En la Figura A.4, cada párrafo se asigna a un vector único, representado por una columna en una matriz D y cada palabra se relaciona con un único vector representado por una columna en otra matriz W. Por último, el vector párrafo y los vectores de las palabras se concatenan o promedian para predecir la siguiente palabra en el contexto dado. Para el ejemplo de la Figura A.4, suponiendo una secuencia del tipo: 'The cat sat on...', el modelo debería ser capaz de predecir la palabra 'on' a partir de las otras palabras del contexto: 'the', 'cat', 'sat'. Esencialmente, el algoritmo muestrea aleatoriamente palabras consecutivas de un párrafo, e intenta predecir una palabra del conjunto de palabras presentadas, tomando como entrada las palabras del contexto y un Paragraph Id.

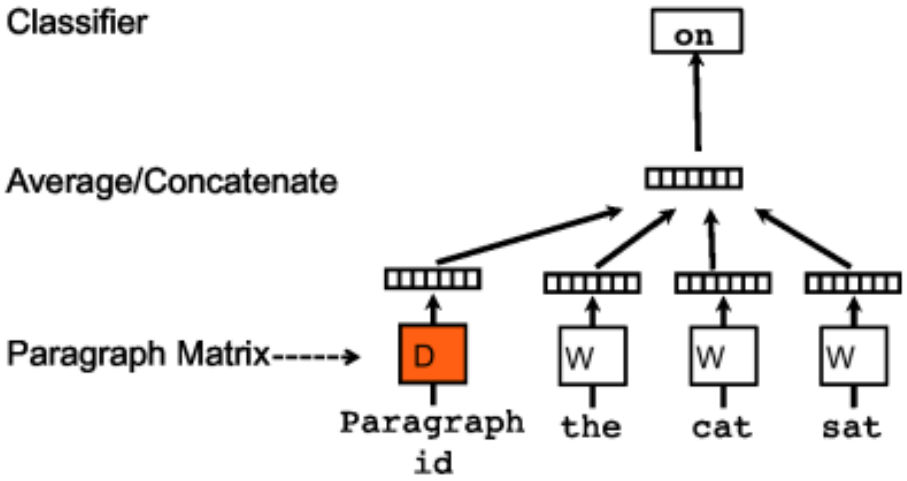


Figura A.4: Ejemplo del funcionamiento del algoritmo PV-DM

- PV-DBOW: Esta técnica es similar al método Skip Gram del modelo Word2Vec. Aquí, se ignora el contexto de las palabras como datos de entrada, pero obliga al modelo a predecir palabras muestreadas aleatoriamente de un párrafo. Es decir, se toma una ventana de palabras muestreando de forma aleatoria una palabra, y junto al Paragraph Id se intentan obtener las posibles palabras que pertenezcan al mismo contexto que el de la palabra elegida. En nuestro ejemplo anterior, tal y como se puede ver en la Figura A.5, dado un Paragraph Id y escogiendo aleatoriamente una palabra de ese párrafo se intentan predecir el resto de palabras que puedan pertenecer al contexto.

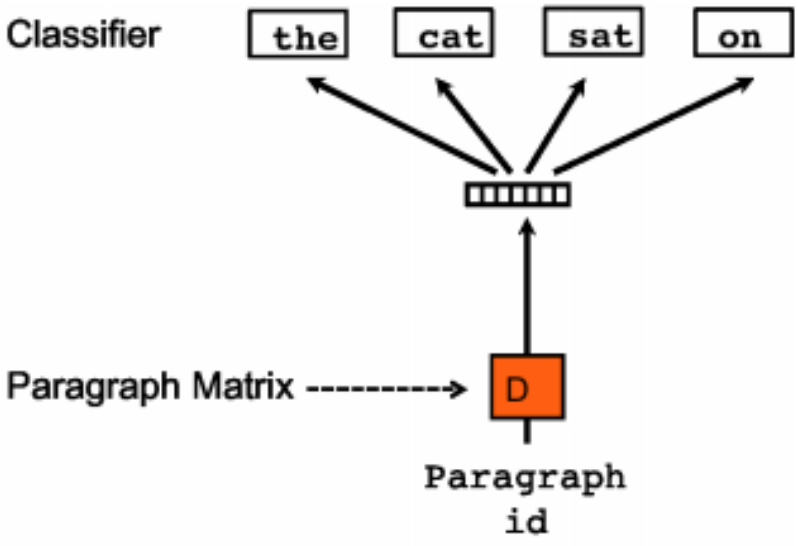


Figura A.5: Ejemplo del funcionamiento del algoritmo PV-DBOW

A.4. Principales parámetros del método Doc2Vec de la librería Gensim

Algunos de los parámetros que debemos fijar en el método Doc2Vec serían:

- documents: Es la lista de objetos de tipo **TaggedDocument** sobre la cual se va a entrenar. Para nuestro modelo, este objeto va a tener la estructura mostrada en la Figura 4.10.
- dm: El tipo de algoritmo que se va a utilizar. Vamos a entrenar nuestro modelo utilizando el algoritmo PV-DM, ya que, como se ha mencionado anteriormente, ha obtenido unos resultados excelentes en ámbitos similares a nuestro objetivo.
- vector_size: Fijar la dimensionalidad de nuestro vector de características. Es decir, precisar el tamaño de la dimensión de nuestro párrafo. Esto hace que a cuantas más características contenga el vector, más preciso será, aunque consumirá más almacenamiento.
- window: Ventana de distancia entre la palabra actual y la palabra que se va a predecir dentro de la misma sentencia.
- min_count: Ignora todas aquellas palabras con una frecuencia menor que este parámetro.
- sample: Umbral para configurar qué palabras de mayor frecuencia se reducen aleatoriamente.
- workers: Número de hilos que se utilizan para entrenar el modelo.

Los parámetros anteriormente mencionados son algunos de los más básicos pero no son los únicos que se podrían emplear en el uso del modelo Doc2Vec [15].

A.5. Hiperparámetros del modelo utilizando Keras Tuner

Keras Tuner [21] es una librería que ayudará en la elección del conjunto de hiperparámetros más óptimos para nuestro modelo. Se pueden distinguir dos principales hiperparámetros:

- Hiperparámetros del modelo: Aquellos que influyen directamente sobre la selección del modelo, es decir, número y características de las capas ocultas.
- Hiperparámetros del algoritmo: Los que afectan directamente la velocidad y la cualidad de entrenamiento del modelo y aprendizaje del modelo.

Para la utilización de esta librería y para la integración de la misma en nuestro modelo, aparte de la arquitectura del propio modelo, vamos a necesitar definir un espacio de búsqueda de hiperparámetros. Para ello, crearemos una función tipo 'builder' para definir nuestro modelo e incluir los hiperparámetros de Keras. Esta función va a devolver directamente el modelo compilado.

Bibliografía

[1] Andreas Mueller. Wordcloud, 2020. [Online; accessed 18-January-2021].

[2] Shervin Minaee. 20 popular machine learning metrics. part 1: Classification and regression evaluation metrics, 2019. [Online; accessed 3-March-2021].

[3] Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262, 06 2017.

[4] L. Akoglu, R. Chandy, and C. Faloutsos. Opinion fraud detection in online reviews by network effects. *Proceedings of the 7th International Conference on Weblogs and Social Media, ICWSM 2013*, pages 2–11, 01 2013.

[5] Md. Hijbul Alam, Woo-Jong Ryu, and SangKeun Lee. Joint multi-grain topic sentiment: modeling semantic aspects for online reviews. *Information Sciences*, 339:206 – 223, 2016.

[6] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders, 2020.

[7] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars, 2016.

[8] Raghavendra Chalapathy and Sanjay Chawla. Deep learning for anomaly detection: A survey, 2019.

[9] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

[10] Xing Chen, Li Ma, and Xiaoquan Yang. Stacked denoise autoencoder based feature extraction and classification for hyperspectral images. *J. Sensors*, 2016:3632943:1–3632943:10, 2016.

[11] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[12] Manqing Dong, Lina Yao, Xianzhi Wang, Boualem Benatallah, Chaoran Huang, and Xiaodong Ning. Opinion fraud detection via neural autoencoder decision forest. *Pattern Recognition Letters*, 132:21 – 29, 2020. Multiple-Task Learning for Big Data (MTL4BD).

[13] François Chollet. Keras, 2015. [Online; accessed 03-January-2021].

[14] Jianlong Fu and Yong Rui. Advances in deep learning approaches for image tagging. *APSIPA Transactions on Signal and Information Processing*, 6, 10 2017.

[15] Gensim. Doc2vec. [Online; accessed 03-January-2021].

[16] Gensim. Word2vec. [Online; accessed 03-January-2021].

[17] Google. Google cloud gpus pricing, 2015. [Online; accessed 03-January-2021].

[18] Google. Tensor flow, 2015. [Online; accessed 03-January-2021].

[19] Victoria Hodge and Jim Austin. A survey of outlier detection methodologies. *Artificial intelligence review*, 22(2):85–126, 2004.

[20] Ahmad Javaid, Quamar Niyaz, Weiqing Sun, and Mansoor Alam. A deep learning approach for network intrusion detection system. In *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*, pages 21–26, 2016.

[21] Keras Tuner. Keras tuner, 2015. [Online; accessed 17-December-2020].

[22] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, page II–1188–II–1196. JMLR.org, 2014.

[23] Yan Liu and Sanjay Chawla. Social media anomaly detection: Challenges and solutions. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2317–2318, 2015.

[24] Yvan Lucas and Johannes Jurgovsky. Credit card fraud detection using machine learning: A survey, 2020.

[25] Justin Malbon. Taking fake online consumer reviews seriously. *Journal of Consumer Policy*, 36, 03 2013.

[26] Mehdi Mohammadi, Ala Al-Fuqaha, Sameh Sorour, and Mohsen Guizani. Deep learning for iot big data and streaming analytics: A survey. *IEEE Communications Surveys & Tutorials*, 20(4):2923–2960, 2018.

[27] Jianmo Ni, Jiacheng Li, and Julian McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 188–197, Hong Kong, China, November 2019. Association for Computational Linguistics.

[28] NumPy. Numpy — scientific computing with python, 1991. [Online; accessed 17-December-2020].

[29] NumPy. Pandas — fast and flexible data analysis tool, 2008. [Online; accessed 17-December-2020].

[30] Myle Ott, Claire Cardie, and Jeffrey T Hancock. Negative deceptive opinion spam. In *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: human language technologies*, pages 497–501, 2013.

[31] Myle Ott, Yejin Choi, Claire Cardie, and Jeffrey T. Hancock. Finding deceptive opinion spam by any stretch of the imagination. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 309–319, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[32] Andre G. C. Pacheco and Renato A. Krohling. Recent advances in deep learning applied to skin cancer detection, 2019.

[33] Guansong Pang, Chunhua Shen, and Anton van den Hengel. Deep anomaly detection with deviation networks, 2019.

[34] Python Foundation. Python — general-purpose programming language, 1991. [Online; accessed 17-December-2020].

[35] Radim Řehůřek. Gensim. [Online; accessed 18-January-2021].

[36] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. Deep one-class classification. In *International conference on machine learning*, pages 4393–4402, 2018.

[37] Haşim Sak, Andrew Senior, Kanishka Rao, and Françoise Beaufays. Fast and accurate recurrent neural network acoustic models for speech recognition, 2015.

[38] Thomas Schlegl, Philipp Seeböck, Sebastian M Waldstein, Ursula Schmidt-Erfurth, and Georg Langs. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In *International conference on information processing in medical imaging*, pages 146–157. Springer, 2017.

[39] Scikit. Scikit — machine learning in python, 2006. [Online; accessed 17-December-2020].

[40] Scikit. Scikit metrics — evaluate model's predictions, 2006. [Online; accessed 17-December-2020].

[41] Simeon Kostadinov. Understanding gru networks, 2017. [Online; accessed 1-March-2021].

[42] Stanford University. Chinese nlp, 2015. [Online; accessed 17-December-2020].

[43] Shiliang Sun, Chen Luo, and Junyu Chen. A review of natural language processing techniques for opinion mining systems. *Information Fusion*, 36:10–25, 2017.

[44] Vlad Comanescu. Dl autoencoder, 2021. [Online; accessed 1-March-2021].

[45] Yequan Wang, Minlie Huang, Xiaoyan Zhu, and Li Zhao. Attention-based LSTM for aspect-level sentiment classification. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 606–615, Austin, Texas, November 2016. Association for Computational Linguistics.

[46] Yiming Wang, Tongfei Chen, Hainan Xu, Shuoyang Ding, Hang Lv, Yiwen Shao, Nanyun Peng, Lei Xie, Shinji Watanabe, and Sanjeev Khudanpur. Espresso: A fast end-to-end neural speech recognition toolkit, 2019.

[47] Yao Wu, Christopher DuBois, Alice X. Zheng, and Martin Ester. Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, WSDM '16, page 153–162, New York, NY, USA, 2016. Association for Computing Machinery.

[48] Drausin Wulsin, Justin Blanco, Ram Mani, and Brian Litt. Semi-supervised anomaly detection for eeg waveforms using deep belief nets. In *2010 Ninth International Conference on Machine Learning and Applications*, pages 436–441. IEEE, 2010.

[49] Xuemei Xie, Chenye Wang, Shu Chen, Guangming Shi, and Zhifu Zhao. Real-time illegal parking detection system based on deep learning. In *Proceedings of the 2017 International Conference on Deep Learning Technologies*, pages 23–27, 2017.

[50] Chang Xu, Jie Zhang, Kuiyu Chang, and Chong Long. Uncovering collusive spammers in chinese review websites. In *Proceedings of the 22nd ACM International Conference on Information and Knowledge Management*, page 979–988, 2013.

[51] Wenju Xu, Shawn Keshmiri, and Guanghui Wang. Adversarially approximated autoencoder for image generation and manipulation, 2019.

[52] Xiao Yang, Daren Sun, Ruiwei Zhu, Tao Deng, Zhi Guo, Jiao Ding, Shouke Qin, Zongyao Ding, and Yanfeng Zhu. Aiads: Automated and intelligent advertising system for sponsored search, 2019.

[53] Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *CoRR*, abs/1509.01626, 2015.

[54] Yan Zhang, Weiling Chen, Chai Kiat Yeo, Chiew Tong Lau, and Bu Sung Lee. Detecting rumors on online social networks using multi-layer autoencoder. In *2017 IEEE Technology & Engineering Management Conference (TEMSCON)*, pages 437–441. IEEE, 2017.