Universidad ORT Uruguay
Facultad de Ingeniería

Prueba de concepto del framework de OpenMined para modelos de Machine Learning

Entregado como requisito para la obtención del título de Máster en Big Data

Pablo Ampuero – 248295

Julio Sánchez – 92935

Tutor: Sergio Yovine

2021

Declaración de autoría

Nosotros, Pablo Ampuero y Julio Sánchez, declaramos que el trabajo que se presenta en esa obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos Proyecto Final del Master en Big Data;
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad;
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra;
- En la obra, hemos acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

Pablo Ampuero

2021-03-26

Julio Sánchez

2021-03-26

Abstract

En la actualidad mayoritariamente los modelos de ML / AI son generados en forma centralizada debido a limitaciones sobre el acceso a los datos sensibles de diferentes fuentes por distintos motivos de protección de la privacidad.

Para atacar estos dos problemas (privacidad y acceso a múltiples dataset) se propone el concepto de “Descentralización de la Información” en forma segura y con las garantías correspondientes sobre el mantenimiento tanto de la privacidad de los datos como de la calidad de los modelos. Las soluciones para esto se basan en la utilización de técnicas de Federated Learning (FL), Homomorphic Encryption (HE) y Decentralized Learning (DL).

Una posible solución para abordar esta nueva forma colaborativa de trabajo es el uso del framework provisto por OpenMined para compartir data en forma anónima y poder usarla para la generación de modelos de ML sin comprometer la integridad de los resultados.

Este proyecto pretende analizar y utilizar esta nueva tecnología para generar un modelo de clasificación de detección de ataques maliciosos, generando modelos para análisis de errores y comportamiento en logs de Apache Web Servers, los cuales tienen una estructura conocida pudiendo extraerse información de estos, pero que también contienen data sensible (IP, Usuario, Passwords, URI) que pueden ser usadas maliciosamente de no ser anonimizadas correctamente.

Se pretende estudiar la robustez y limitaciones de los algoritmos planteados por el framework en cuanto a cómo anonimizar la data y marcos de gobernanza mediante una prueba de concepto.

A partir de dicha prueba se concluye que framework garantiza que no se pueden identificar registros específicos a partir de las técnicas usadas de HE. La funcionalidad de creación de usuarios y un sistema de permisos sobre los datos publicados todavía no cuenta con un mecanismo robusto para la gobernanza de datos. En la implementación realizada, el framework no presenta mayor complejidad al momento de ser utilizado por

el Data Science con respecto al modelo de referencia utilizado. Tampoco se observaron impactos negativos en la calidad de los resultados.

Es un framework todavía en construcción, por lo que se espera continúe ampliando las funcionalidades disponibles, entre ellas el entrenamiento remoto, que todavía está en desarrollo, y en la función de activación de la red, que como se comentó se tuvo que aplicar la RELU.

Palabras clave

OpenMined, Federated Learning, Differential Privacy, Homomorphic Encryption, Decentralized Learning, Machine Learning, Client-Server Machine Learning, PySyft

Índice

Índice de tablas

Índice de ilustraciones

1. Introducción

En la actualidad mayoritariamente los modelos de ML / AI son generados en forma centralizada, donde una entidad consigue datasets con la información que desea entrenar sus modelos. Esa información ya sea propia o comprada es determinante al momento de la calidad de los modelos entrenados.

En contraparte, las entidades que poseen datos para poder entrenar modelos no siempre pueden compartir sus datasets. Las limitaciones actuales al momento de usar data de diferentes fuentes para la generación de modelos de ML con datos sensibles de diferentes empresas, de uso privado, o que pongan en compromiso la identificación de una persona, ya sean por temas de regulación gubernamental, cuidado de marca, violación de privacidad, o el costo de conseguir los mismos, hace que las restricciones de generación de tales modelos sean tan grandes que no se puedan generar.

Para atacar estos dos problemas (privacidad y acceso a múltiples dataset) se propone el concepto de "Descentralización de la Información" en forma segura y con todas las garantías el cual se resume en varios conceptos:

- Federated Learning. Esta técnica permite el entrenamiento de modelos localmente en las instalaciones de los dueños de la data.

- Homomorphic Encryption. La data es trabajada en forma encriptada por los modelos brindando garantías al poseedor de la data que el modelo que utilizó la misma para entrenarse no va a poder revelar ningún aspecto de privacidad de esta.

- Decentralized Learning. Los modelos son entrenados en los dispositivos donde se aloja la data que van desde servers en organizaciones, servicios en la nube o incluso hasta aplicaciones en celulares entrenando con data online.

Una posible solución para abordar esta nueva forma colaborativa de trabajo es el uso del framework provisto por OpenMined para compartir data en forma anónima y poder usarla para la generación de modelos de ML sin comprometer la integridad de los resultados.

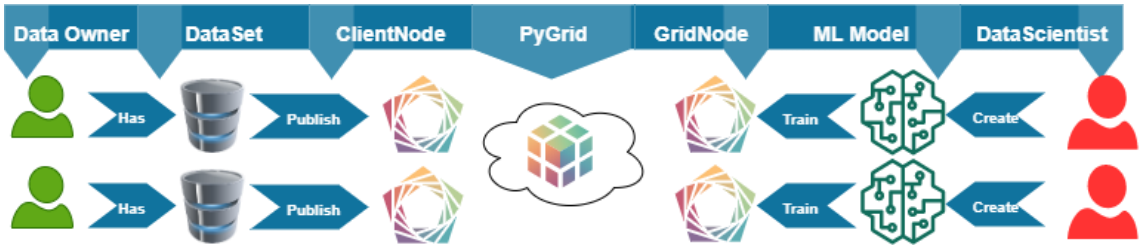


Ilustración 1 - Framework

1.1. Data Owner

Data Owner, es aquella entidad que posee un dataset para generación de modelos. Estos datasets tienen un valor asociado, ya sea por el costo de generación de estos, por procesamiento, el rubro de los datos, la cantidad de datos y calidad de información que aporta, etc.

Dado el valor que poseen estos datos, no es esperable que se brinden (por lo general) de forma gratuita. Y aunque fuera el caso, puede haber restricciones legales.

1.2. Data Scientist

El Data Scientist es aquella entidad que desea realizar modelos de ML pero no tiene los datasets necesarios para entrenar los modelos, ya sea por temas de financiamiento, infraestructura o capacidad de procesamiento.

1.3. Gobernanza

El framework a estudiar se presenta como una posible solución a la problemática de la gobernanza sobre los dataset generados a partir de datos sensibles de individuos u organizaciones.

Permite la gestión por parte del Data Owner sobre distintas limitaciones al acceso y utilización de sus datos encriptados. Lo cual le brinda la posibilidad de disponibilizar dichos datos de manera discrecional según su criterio personal dándole control sobre los mismos tanto lógica como físicamente.

1.4. Objetivo

Este proyecto pretende analizar y utilizar esta nueva tecnología para generar un modelo de clasificación de detección de ataques maliciosos, generando modelos para análisis de errores y comportamiento en logs de Apache Web Servers (AWS), los cuales tienen una estructura conocida pudiendo extraerse información de estos, pero que también contienen data sensible (IP, Usuario, Passwords, URI) que pueden ser usadas maliciosamente de no ser anonimizadas correctamente.

Se pretende estudiar la robustez y limitaciones de los algoritmos planteados por el framework en cuanto a cómo anonimizar la data y marcos de gobernanza mediante una prueba de concepto. Dicha prueba se realizará a partir de la implementación de la arquitectura requerida para analizar el caso real mencionada (logs de AWS), así como de la ingeniaría de atributos necesaria sobre los elementos a ser utilizados para la determinación de la clase selectora.

2. Estado del arte

A continuación, se pasa a describir brevemente parte del análisis bibliográfico realizado, definiciones y estado actual de los distintos conceptos fundamentales relevantes para el presente trabajo. Como son; Federated Learning (FD), Differential Privacy (DP), Homomorfic Encryption (HE), y las soluciones disponibles en el mercado similares a lo que busca resolver el framework de OpenMined.

Se toma especial énfasis en detallar el proceso por el cual se logra implementar la técnica de Federated Learning, ya que, junto con los algoritmos de encriptación, es la base para el framework de OpenMind a analizar. Esto en base al texto de referencia en la materia [1].

2.5.Federated Learning

Es una técnica de ejecución remota que facilita a los Data Scientist poder entrenar modelos en forma remota en los dominios de los Data Owner sin necesidad de tener acceso directo al dataset. En otras palabras, es la infraestructura que facilita a la generación de modelos en forma distribuida en los servidores de los Data Owner sin que los Data Scientist puedan inferir ni ver la data que se utiliza para el entrenamiento.

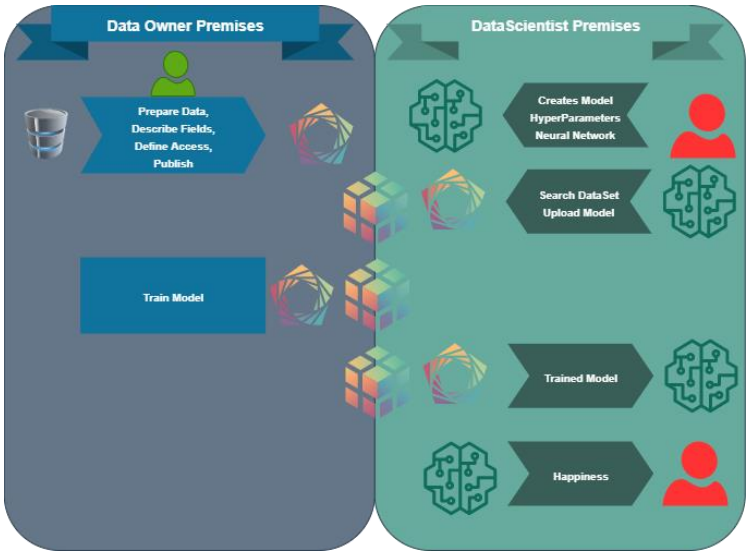


Ilustración 2 - Federated Learning, flujo alto nivel

Beneficios:

1) El Dataset permanece en todo momento en la red y bajo el control y supervisión del dueño de dichos datos (Data Owner).
2) La privacidad de los datos está dada y definida por el Data Owner que publica el Dataset.
3) Los Data Scientists no tienen responsabilidad sobre los datos ya que los modelos se que se obtienen son sobre datos anonimizados.
4) Disminuye la transferencia de grandes Dataset.

2.5.1. Antecedentes

Esta técnica de entrenamiento distribuido de modelos sobre un conjunto de datos descentralizados es abordada en papers presentados desde Google en trabajos orientados principalmente a Federated Optimization [2], y por ende que comienzan a trabajar sobre la base de Federated Learning [3].

Este avance en la investigación lleva al trabajo de referencia en la materia "Towards federated learning at scale: System design" [1], el cual es punto de partida para gran parte de las soluciones actuales. En donde se detalla una infraestructura para la aplicación de Federated Learning (FL) a gran escala utilizando dispositivos móviles basado en el uso de TensorFlow para entrenar una Deep Neural Network (DNN). Dicha infraestructura permite el entrenamiento de los modelos, acercando estos al dataset, en lugar de extraer los datos de los dispositivos, lo que permite mitigar los problemas asociados a la propiedad, privacidad y localización de dicha data.

Lo cual se logra a través de la aplicación de técnicas de Federated Averaging, Secure Agregation y Differential Privacy para combinar los pesos de la DNN entrenadas y mantener la privacidad de los datos. Para esto es necesaria cierta sincronización entre los lotes de dispositivos a utilizar para el entrenamiento, por lo cual el proceso de entrenamiento se realiza de forma síncrona en lotes.

2.5.2. Arquitectura

En la arquitectura propuesta en Towards federated learning at scale: System design, el servicio de FL se encuentra distribuido en la nube, los dispositivos se conectan a dicho servicio, y tanto estos como los datos que contienen presentan distintas características

que los hacen aptos para distintos tipos de tareas de entrenamiento. Por lo que ambos, tareas y recursos disponibles, son catalogados y clasificados.

De todos los recursos disponibles para la realización de FL, el servidor va seleccionando lotes más pequeños, de acuerdo con las características antes mencionadas y la disponibilidad de dichos dispositivos. Luego de confirmar que se encuentran aptos se procede a enviar el modelo a entrenar a dichos dispositivos, para luego agregar los resultados por medio de Federated Averaging y guardar estos como resultado del entrenamiento.

Para que se complete este proceso es necesaria que cierta proporción configurable de dispositivos del lote cumplan con todo el ciclo (completarlo sin interrupciones ni errores), a modo de tomar el resultado como válido. Este proceso transcurre en rondas en las cuales se llevan a cabo tres fases; selección, configuración y reporte.

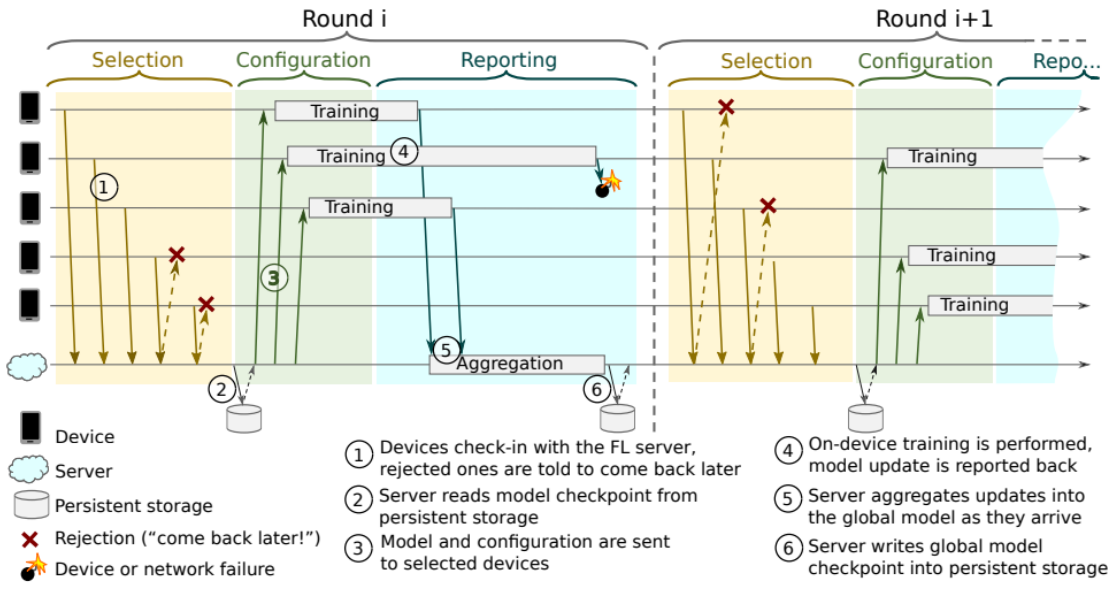


Figure 1: Federated Learning Protocol

Ilustración 3 - Protocólo de Federated Learning Página 2 [1]

2.5.3. Dispositivos

En el trabajo antes mencionado, y que sirve de referencia para los desarrollos que ocurrieron posteriormente, la interacción y gestión con el servidor de FL es a través de una API alojada en el dispositivo. Dado que es en dichos dispositivos donde se

encuentra almacenada la información, esta debe estar encriptada y seguir los protocolos de seguridad adecuados. Esta data se encuentra catalogada de acuerdo con la tarea de aprendizaje que se puede realizar sobre la misma, y es eliminada luego de cierto periodo.

El estado del dispositivo (acceso a una red WIFI, carga, disponibilidad de recursos) es tomado en cuenta para informar al servidor FL de la disponibilidad del dispositivo para realizar un entrenamiento de Machine Learning. Ya que se busca que dicho proceso no interfiera negativamente en la utilización del dispositivo por parte del usuario. A partir de esta información el servidor de FL decide si hay tareas disponibles para el tipo de información con la que cuenta el dispositivo, o si debe consultar nuevamente luego.

En caso de que se decida procesar una tarea en el dispositivo se le envía el modelo y el plan de entrenamiento y lo ejecuta sobre los datos recopilando los resultados. Estos son enviados al servidor y se borra toda la información temporal previamente utilizada.

A modo de confirmar la validez de dichos dispositivos para proteger los resultados del entrenamiento de un posible ataque se utiliza la solución proporcionada por SafetyNet. [4]

2.5.4. Servidor

En la solución presentada, el servidor de FL debe soportar la gestión de los estados de eventualmente millones de dispositivos, con distintos tipos de características para distintos tipos de tareas en distintas regiones y horarios para poder asignar tareas de ML a estos dispositivos. Al momento de asignar estas tareas debe soportar los niveles de tráfico procedentes de la interacción con los dispositivos, las señales de estado, el envío de los modelos y planes de ejecución, las métricas obtenidas, etc.

El diseño se basa en el Actor Model, con la participación en el sistema de principalmente cuatro actores; Coordinator, Master Agreggator and Selector, como se puede ver en la siguiente figura.

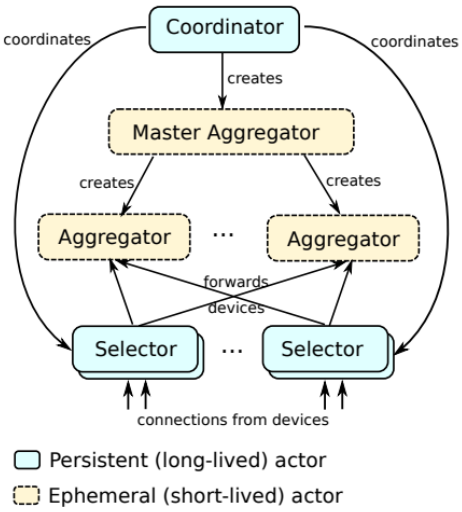


Figure 3: Actors in the FL Server Architecture

Ilustración 4 - Actores en la arquitectura de servidores de Federated Learning. Página 5. [1]

El Coordinador gestiona una determinada población de dispositivos y puede haber varios coordinadores, cada uno con su respectiva población las cuales responden a un solo Coordinador. Se encarga de la sincronización a alto nivel, y a partir de los estados de los dispositivos, recopilados por el Selector, les indica cuantos deben aceptar para realizar determinada tarea de ML y crea Agregadores Maestros para gestionar las rondas de entrenamiento para cada tarea de ML.

A su vez los Agregadores Maestros pueden crear uno o más Agregadores en los cuales delegan la gestión de las rondas. La información de estas rondas no se persiste hasta que es agregada por el Agregadores Maestros, sino que se trabaja en memoria lo que permite aumentar escalabilidad, disminuir latencia y proteger de eventuales ataques a los logs de las actualizaciones de las rondas. Dichos logs podrían tener información que permita identificar un individuo a partir de los datos resultante del entrenamiento, pero al no almacenarse se evita un posible punto de compromiso. Mientras están en memoria los datos se mantienen encriptados, a la vez que la actualización en los Agregadores se hace por medio de Secure Agregation.

Los Selectores son los que están directamente conectados con los dispositivos y que recopilan los estados en que se encuentran. A partir de la información recibida del Coordinador de cuántos dispositivos son necesarios para cumplir determinada tarea, los

Selectores aceptan la conexión de un subconjunto de estos y los envían a los Agregadores.

2.5.5. Modelos

Dentro de este esquema de Federated Learning tanto los modelos como los planes de ejecución de las tareas implementadas con TensorFlow son cargados en el servidor FL, y es éste el que los disponibiliza en los dispositivos a través de los mecanismos antes mencionados.

El primer paso es definir desde Python la tarea a aplicar en determinada población objetivo a través de funciones incorporadas en TensorFlow. Estas tareas se validan contra data de test, siendo esta data similar a la que se espera que haya en los dispositivos. También se definen distintos parámetros como número óptimo de dispositivos en una ronda, y demás hiper parámetros del modelo.

Estas tareas se asocian a un determinado plan, el cual consiste en una parte dirigida al dispositivo final y otra al servidor de FL. Para el cálculo en el dispositivo se indica el grafo de TensorFlow, el criterio de selección de la data para el entrenamiento que se encuentra almacenada en el dispositivo, cómo lotear esta data y en cuántos epochs se debe correr, entre otras indicaciones. La parte del plan dirigida al servidor de FL tiene principalmente la lógica para la agregación.

Antes de la carga de las tareas y el plan en el servidor se controla que el código haya sido revisado, validar que funciona correctamente en todas las versiones de TensorFlow asociadas a la tarea de FL, así como que los recursos a consumir de los dispositivos de la población objetivo no excedan ciertos umbrales. Se versionan los planes para cada tarea a modo de evitar incompatibilidades.

Luego del entrenamiento y el proceso de agregación, los resultados quedan almacenados en un lugar previamente definido por el Data Scientist. La solución propuesta ya presenta herramientas para el análisis de estos datos, así como formas estandarizadas para subir y por ende trabajar estos resultados desde Python.

2.5.6. Desafíos

En la revisión del trabajo tomado como punto de referencia para la construcción de un sistema a gran escala de Federated Learning, los autores admitían distintas posibilidades de mejora.

En primer lugar, se identifica el problema del sesgo en las poblaciones utilizadas para el entrenamiento. Esto a partir de que es requerimiento utilizar dispositivos cargados, con plan plano de datos o acceso irrestricto a Wifi al momento de tomarse como dispositivo apto. También es causa de sesgo las características mínimas requeridas por los dispositivos (hardware, software). Todo esto puede generar seso ya sea por regiones, poder adquisitivo, entre otros, que debe ser tomado en cuenta al momento de evaluar los resultados.

Se plantean posibilidades de mejora en la ejecución en paralelo para la aplicación de Federated Averaging. También se pueden mejorar los mecanismos de compresión del tráfico con los dispositivos para lograr un menor impacto en el ancho de banda utilizado, que para alguna de las tareas a aplicar FL pude ser bastante alto, incluso más alto que simplemente enviar la data (si no se tuviera en cuenta el componente de privacidad que aporta FL).

Y se plantea la oportunidad de utilizar como base esta arquitectura para no solo efectuar tareas de ML utilizando TensorFlow, sino que aprovechar esta técnica para hacer análisis a través de Federated Analytics, al aplicar métodos utilizados por los data scientist para analizar la información cruda almacenada localmente en los dispositivos. O incluso Federated Computation, donde se menciona la posibilidad de combinar la estructura con flujos en Map Reduce.

2.6. Differential Privacy

Esta técnica permite garantizar que la información públicamente visible no varíe mucho si un dato del Dataset varía. Esto a partir de la utilización de ruido aleatorio, lo que permite resguardar la privacidad de la data. [5]

En la práctica, es que se puedan eliminar los ataques por similitud de data, en donde se posee un Dataset conocido, con data en claro, y los resultados de hacer consultas sobre el mismo son utilizados para comparar los resultados sobre las consultas con el Dataset encriptado, pudiendo así determinar si un dato está presente o no. Por ejemplo, supongamos que tenemos un Dataset en claro de usuarios conectados a una plataforma X en un periodo, y un Dataset encriptado con las acciones que realizaron en la misma. Mediante el conocimiento de la data de usuario, fecha y hora se pueden hacer consultas sobre el Dataset encriptado que revelen si el usuario realizó o no cierta acción en el sitio. La idea principal se basa en que cualquier consulta (o ataque) de estudio de patrón de input-output sobre la data encriptada no debe de revelar si cierto dato está o no está presente en el Dataset.

Podemos decir en otras palabras, que la probabilidad de que el resultado de una consulta sobre un Dataset cambie es muy baja cuando un dato se agrega o quita del mismo. Dado dos Datasets de igual estructura, uno que tenga la información y otro que no, el resultado de la consulta X sobre los dos son probabilísticamente casi iguales.

Esta técnica tiene un trade-off entre privacidad y accuracy, cuanto más ruido se incorpore a los datos, menos accuracy vamos a tener. Esto generalmente se ajusta con un parámetro conocido como “perdida de privacidad”, cuanto más bajo este valor menos seguridad va a tener sobre la data, y cuanto mayor sea menos accuracy pudiendo afectar el resultado de los modelos de ML.

2.7. Homomorphic Encryption

Las técnicas de encriptación homomórficas son aquellas que nos permiten hacer operaciones sobre los datos encriptados sin necesidad de conocer la clave para desencriptarlos y aun así que los resultados sean los esperados y continúen encriptados.

El sistema de encriptación que usa se basa en el concepto de claves privada y públicas, en donde la función de “encriptación” es publica y conocida pero la función de “desencriptar” no lo es, de esta forma información solo es reversible por el que posee la función de encriptación privada.

Una de las propiedades que esta técnica de encriptación ofrece es la posibilidad de agregar data a la información original sin comprometerla. Dado un mensaje X, se le puede adicionar un contenido Y, con la misma encriptación en donde el receptor poseedor de la clave privada va a poder obtener X + Y por separado, preservando la integridad del mensaje original y obteniendo la nueva información.

La encriptación puede ser:

- Parcial, en donde operaciones de multiplicación o división son soportadas sobre la data, preservando la integridad del mensaje, pero no las dos.
- Total, en donde las operaciones no tienen limitaciones y el manejo de datos encriptados no difiere en su contraparte de datos en claro.

Sistema	características
RSA	Homomorfismo de multiplicación. Basado en 2 números primos grandes. No es randomica, puede llegar a ser quebrada.
ElGamal	Homomorfismo de multiplicación. Elige 2 números primos grandes basado en un generador de los mismos.
Paillier	Homomorfismo de sumatoria. Elige 2 números primos grandes y utiliza la función de Charmichel.

Tabla 1 - Homomorphic Encryption

Con estas técnicas los Data Owners no corren peligro de revelar información confidencial de su Dataset y los Data Scientists tienen garantizado que las operaciones sobre los datos van a ser las correctas aun sin tener acceso a la misma en claro.

2.8. Comparativa de soluciones de Mercado

En esta sección vamos a evaluar las principales opciones que se encuentran en el mercado destacando tanto en FL/DP como están implementados, lo que las diferencian, y los contras de estas.

El uso de FL/DP para la generación de modelos de Deep Learning es algo que es muy reciente ya algunas compañías aún no han sacado un framework completo, pero son key players en el mercado y en el desarrollo de nuevas tecnologías.

Las que se evaluaron son:

- PySyft de OpenMined
- Microsoft
- IBM
- FTE de Webbank's AI Department
- Google

2.8.1. PySyft

PySyft [6] [7] es un framework en Python propuesto por OpenMined para la generación de modelos en un ambiente federado en forma segura y anónima. Su fuerte es que es totalmente compatible con TensorFlow, PyTorch y Keras que son los frameworks más usados para Deep Learning.

Concepto	Implementación
FL	Implementa varias arquitecturas: • Model Delivery, los usuarios tienen datos, un Data Scientist crea un modelo, consigue un cliente con datos, le manda el modelo, recibe los gradientes, los average, acutaliza el modelo. • Safe Data Delivery. En los casos como análisis de NLP provee los mecanismos para hacer el pre procesamiento (tokenizacion, etc) en el data client y luego enviar eso encriptado de todos los clientes para poder generar el modelo en el server central. En esta forma de trabajo lo que se manda es el modelo de preprocesamiento.
DP/MPC	Utiliza MPC/HE dentro de Pytorch/Tensorflow para garantizar la seguridad y anonimidad de los datos. Soporta varios sabores de DP.
Pros	Utiliza frameworks ya conocidos haciéndolo muy fácil de poder usar y probar nuevos modelos. Para Data Scientists les da más libertad pudiendo controlar todos los detalles, hiperparametros, tipos de privacidad.
Cons	Está aún muy verde, no todo funciona tan bien como dicen

Ilustración 5 - Propiedades de PySyft

2.8.2. Google

Google implementó su librería Tensorflow Federated [8] exclusivamente para dispositivos tipo Android. Su estrategia esta basada en compartir modelos, los cuales son optimizados en un ambiente por los diferentes clientes, los cuales actualizan sus resultados con el concentrador.

Concepto	Implementación

FL	Nodo concentrador que tiene un modelo con pesos preestablecidos. Este nodo concentrador pasa su modelo a los diferentes dispositivos Android (clientes) los cuales van a actualizar los pesos del modelo original que recibieron con los datos en forma local. El modelo actualizado es enviado al nodo concentrador el cual utilizara para actualizar los pesos en el modelo core.
DP/MPC	La comunicación entre cliente y servidor es encriptada. Utiliza el concepto de "si usamos averages de cosas muy chiquitas nadie va a poder darse cuenta de que era el todo grande" (minibatch, average gradient, ruido gausiano, modelo, actualizar gradientes core) En el core server también hay mecanismos de actualización, se precisan XXX cantidad de resultados de clientes antes de actualizar el modelo core, el cual se hace haciendo otro average de todos los gradientes de las actualizaciones de los clientes. Solo el nodo concentrador conoce los resultados parciales de cada cliente. Los datos no salen nunca del cliente.
Pros	TensorFlow es una librería muy usada para Deep Learning. Muy fácil de usar, los datos están siempre en el cliente y la fuerza de procesamiento es chica y totalmente online. Las tasas de transferencia son chicas ya que la comunicación de las actualizaciones de grandientes son comprimidas y encriptadas a nivel de comunicación nada más.
Cons	Está pensado para Android y apps en Android. Está pensado para OnDevice Prediction.

Ilustración 6 - Propiedades TensorFlow Federated

2.8.3. Microsoft

Microsoft es uno de los frameworks que aún no ha salido en producción con su solución de FL/DP. Ha invertido mucho en research para potenciar su solución de AI Lab utilizando FL y DP con Homomorphic Encription [9].

Es considerado uno de los jugadores principales en el futuro cuando lance su plataforma de DP utilizando HE que está siendo desarrollada por el departamento de matemáticas de la universidad de Harvard.

Su implementación de FL [10] esta basada en no delegar la responsabilidad del entrenamiento a una sola de las partes sino compartir la tarea entre los dos, utilizando siempre técnicas de DP/HE.

2.8.4. IBM

IBM salió recientemente al mercado con su solución de FL/DP integrada claramente a su plataforma de AI Watson.

Implementa DP [11] básicamente con la adición de ruido y la generación de synthetic data garantizando la integridad de los datos. Para Deep Learning en AI también implementara HE [12] aunque aún no está disponible para ser usada.

Su plataforma de FL [13] permite el entrenamiento de modelos donde la data nunca deja las instalaciones del cliente, comparten pesos de los modelos los cuales son agregados en el servidor y redistribuidos.

Pros, al pareces tienen cero limitaciones algorítmicas, corren todos los algoritmos de Watson sin sudar una gotita, y que su NLP distribuido es la sal como el local.

Su plataforma es muy nuevita, sacada al mercado en Julio 2020, pero se diferencia en las otras en los algoritmos de Fusion [14] que manejan al momento de agregar los resultados de los clientes son bastante state of the art y específicos para cada tipo de modelos.

2.8.5. FATE de Webbank's AI Department

FATE (Federated AI Technology Enabler) [15] es un framework open-source, iniciado por el departamento de AI de Webank (Banco Digital de China) [16], el cual desarrollo toda su plataforma de servicios 100% online.

El sistema de Federated Learning implementado en su producto FATE-Flow brinda un pipeline completo para compartir datos, seleccionar features, implementar modelos y evaluar los mismos. Posteriormente estos modelos, son compartidos en forma segura sin comprometer información del origen de los datos y utilizados por terceros.

Al igual que Google utiliza técnicas de DP/MPC para asegurar la privacidad de los datos.

3. Dataset y problemática a resolver

Todos los días los servidores web son atacados por agentes maliciosos para poder explotar sus fallas de seguridad, de esa forma ganar acceso a recursos sin autorización y poder robar datos sensibles para su beneficio. "Apache Web Server" [17] es el servidor más usado junto con Nginx [18], como servidores de contenido web. Entre sus características, los Servidores Apache cuentan con un sistema de log (bitácora de registro) de la actividad que se realiza dentro del mismo por los usuarios externos.

El Dataset se origina desde los logs de servidores Web Apache. En el mismo, previamente clasificados, se encuentran

los diferentes ataques de intrusión que sufrieron en forma de URL. Estos ataques se presentan en forma de texto plano, interpretable por un humano a simple vista y puede contener información que comprometa la operación del sitio web de donde se obtuvo el mismo.

Los logs provistos por este tipo de servidores contienen una estructura conocida y analizable. Y las compañías no pueden compartir esta información sin comprometer información sensible acerca de su estructura interna, de los ataques exitosos, de URLs que den información clasificada.

Entre los diferentes tipos de ataques:

- Tipo de acceso POST/GET
- URI Exploit (Access Path, Domain Abuse)
- DoS
- SQL Injection
- XSS Attacks

Es por esto por lo que a la hora de utilizar la información sensible que brindan dichos logs, es necesario encontrar mecanismos para la protección de esta data. Es buscando resolver este punto que se utiliza Homomorphic Encryption para poder trabajar y aprovechar dichos datos, a la vez que se mantiene la seguridad y confidencialidad de los mismos.

4. Preparación del Ambiente

En esta sección se presenta la arquitectura física seleccionada sobre la cual desarrollar la generación de modelos por ambas entidades, el Data Owner y el Data Scientist.

Es también en esta fase donde se preparan los Dataset para que queden disponibles para ser utilizados en la generación de modelos para comparar el framework brindado por OpenMined contra un modelo de referencia. Este último modelo no está ni encriptado ni entrena de forma distribuida, lo cual permite realizar la comparación de las distintas performances con el de OpenMined y así poder formular conclusiones.

La arquitectura física y la elaboración de ambos modelos, el que fue implementado utilizando PySyft y el que se realiza como modelo de control, es detallada en los Anexos. Del Anexo 1 al 4 se encuentra la preparación del ambiente físico y en los Anexos 5 y 6 se encuentra el modelo de referencia y el que utiliza SyferText, respectivamente.

4.1. Arquitectura física

El objetivo de esta sección es explicar la arquitectura de los servidores instalados en la red y sus capacidades.

La red cuenta con 5 servidores en donde se buscó recrear una estructura de producción, en donde estén representadas las distintos agentes que participan en el flujo.

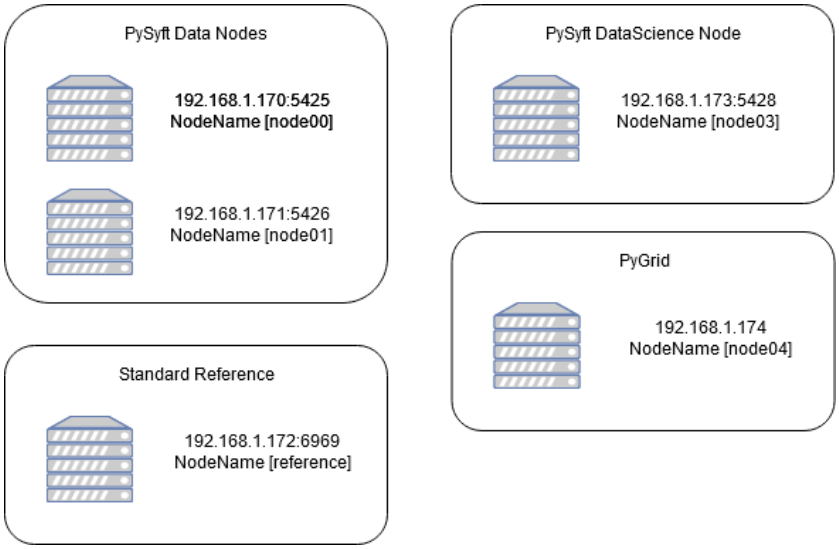


Ilustración 7 - Arquitectura física

- **PySyft Data Node.** Estos son los nodos donde están los Data Owner, que son los clientes poseedores de los datos, a los cuales se les debe garantizar la anonimidad de los datos. Estos tienen el alias de Bob y Alice para diferenciarlos.
- **PySyft Data Science.** Este es el nodo dedicado al Data Science, el cual elabora y publica el modelo a utilizar. Este nodo fue llamado Company.
- **Standard Reference.** Nodo donde se corre el modelo de referencia para poder compararlo con los resultados luego de aplicar el framework de análisis.
- **PyGrid.** Es el encargado de administrar usuarios y permisos, a la vez que ayuda a controlar la gobernanza de los datos.

4.2. Arquitectura lógica

En esta etapa se realiza el análisis del framework desde las dos ópticas descriptas, la del Data Owner y la del Data Scientist.

Data Owner: Se encarga de disponibilizar los datos de acuerdo con las restricciones que se establecen sobre los mismos.

Data Scientist: Se encarga de elaborar el modelo a entrenar sobre los datos del Data Owner a través del ambiente elaborado.

4.2.1. Tipo de modelo

Optamos por hacer un modelo NLP ya que el problema a resolver implica clasificar un conjunto de palabras asociadas a los logs de acceso Apache. Esto nos permitirá limpiar los datos, eliminar palabras que no son relevantes para el análisis, que no aportan información al momento del análisis.

4.2.2. Arquitectura del modelo

Se crea una RNN simple, muy simple, como para hacer las pruebas. Esta cuenta de una capa con 300 inputs y 2 salidas con activación de RELU, lo suficiente para prueba de concepto del framework que estamos analizando. La Cross Entropy no está implementada como tal, por lo que en este caso se utiliza la función de activación RELU en lugar de Softmax para simular Cross Entropy.

Esta red es utilizada en un modelo de referencia totalmente lineal con los datos en claro, y en el modelo de SyferText con los datos encriptados.

```
class Classifier(torch.nn.Module):
    def __init__(self, in_features, out_features):
        super(Classifier, self).__init__()
        self.fc = torch.nn.Linear(in_features, out_features)

    def forward(self, x):
        logits = self.fc(x)
        probs = F.relu(logits)
        return probs, logits
```

```
classifier = Classifier(in_features = 250, out_features = 2)
print(classifier)
```

```
Classifier(
  (fc): Linear(in_features=250, out_features=2, bias=True)
)
```

Ilustración 8 - Modelo para prueba de concepto del framework

Lo siguiente es un diagrama de la estructura de ejecución remota, donde se reflejan los distintos roles y cómo van a interactuar:

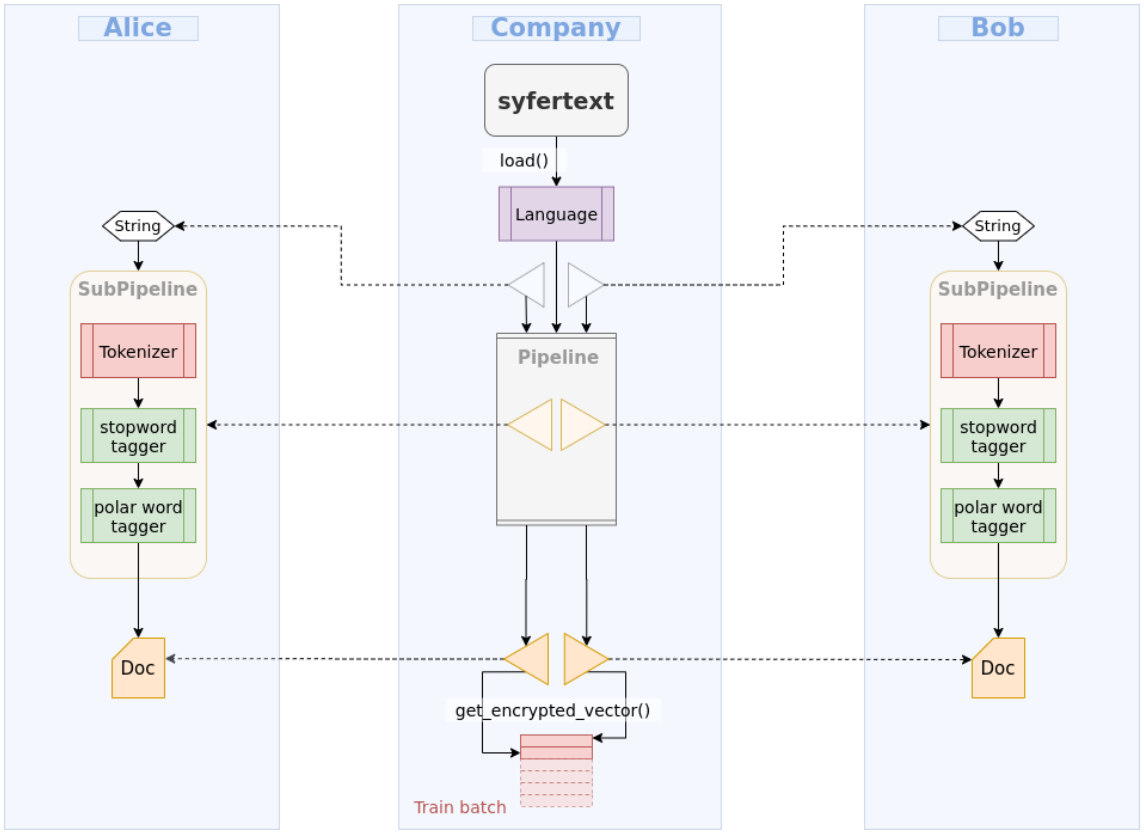


Ilustración 9 - Extraído del GitHub de SyferText [19]

En esta interacción se ve como Company coordina la recepción de los datos encriptados tanto desde de Bob como de Alice, para poder aplicar el modelo.

La anonimidad remota y la consistencia de datos entre los distintos Data Owners se preserva mediante el envío por parte de Company de un objeto con las instrucciones de preprocesamiento para que todos los dataset hagan el mismo preprocesamiento de datos. Al mismo tiempo les envía un encriptor común para todos, para que los datos que lleguen encriptados estén estandarizados.

Pasos del flujo:

1) Bob y Alice disponibilizan sus datasets con los accesos que consideran pertinentes.
2) Company quiere entrenar un modelo de NLP con los datos de Bob y Alice.
3) Company define un conjunto de secuencias para preprocesamiento de los datos (SubPipeline), para que queden utilizables para su modelo.

4) Company define un objeto de encriptación, responsable de la encriptación. Esto es importante porque el objetivo es que la misma palabra quede estandarizada en su valor encriptado en los distintos datasets.
5) Company envía los objetos definidos en los puntos 3 y 4 a Bob y Alice.
6) Bob y Alice, por separado, toman los objetos que recibieron, hacen el preprocesamiento de sus respectivos datases y los encriptan, y devuelven los datasets encriptados a Company.
7) Company entrena el modelo con los datos de Bob y Alice en conjunto haciendo un merge de ambos datasets y lo utiliza como un gran dataset.

Cabe la pena recalcar que tanto la arquitectura de la red como la definición del algoritmo de train, la función de perdida y el optimizador es el mismo que el Data Science utilizaría así no usara el framework de PySyft.

El método de encriptación utilizado es el proporcionado por Encriptación Homomorfica, que garantiza la anonimidad de los datos, y que no permite identificar el dato original a partir del encriptado.

Al momento de implementar el caso de uso no estaban disponibles los remote workers, por lo que fue realizado en forma local.

4.2.3. Resultados del Train

Estas graficas muestran la evolución del Accuracy y Loss al momento de realizar un entrenamiento de la red.

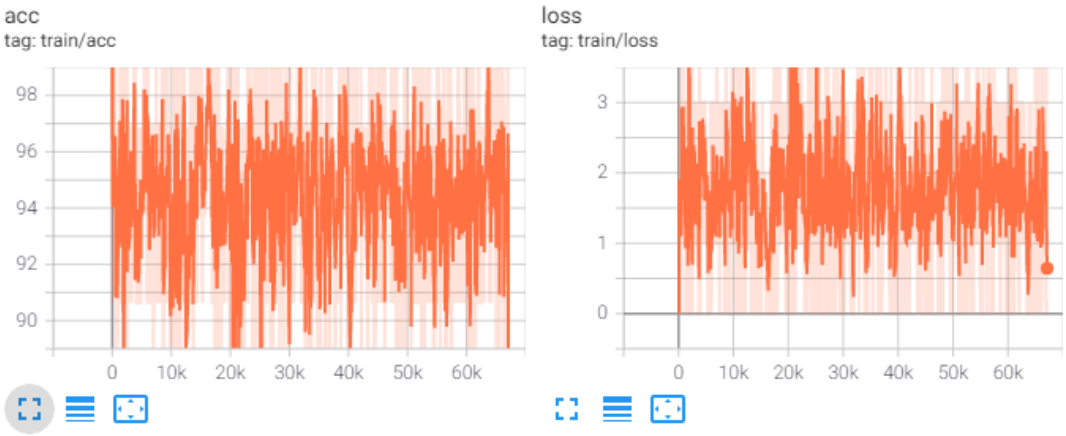


Ilustración 10 - Modelo de referencia

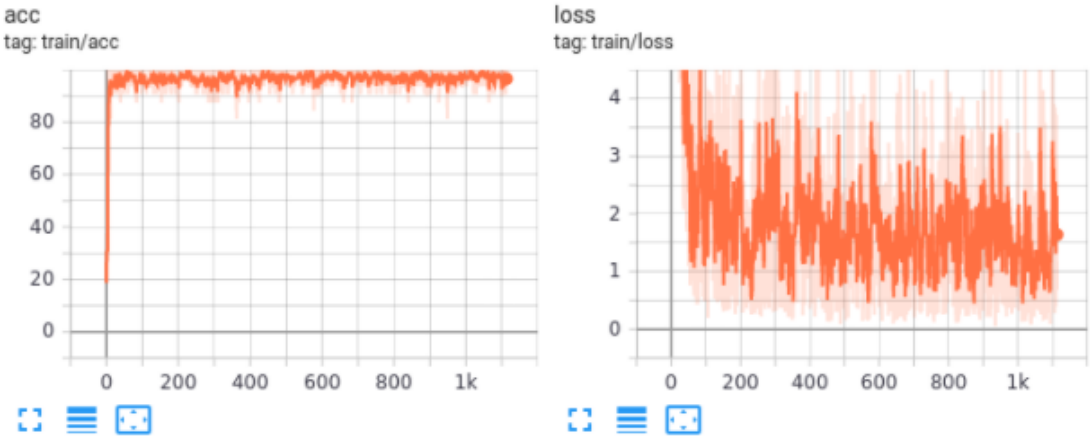


Ilustración 11 - Modelo de SyferText

5. Conclusiones

El objetivo de este trabajo es la evaluación del framework de OpenMined en los siguientes puntos:

Garantizar la anonimidad de los datos. Los datos no deben ser reconstruibles a partir de la información utilizada para el entrenamiento del modelo, y que permitan identificar registros específicos.

Seguridad de la información. Sumado al punto anterior el framework debe proveer mecanismos de gobernanza sobre el uso de los datos por parte de los usuarios que le permitan a estos brindar distintos accesos a los mismos de acuerdo con sus preferencias.

Transparencia al momento de generar modelos. Que los datos estén encriptados no deben agregar complejidad extra, ni poner limitaciones, al momento de definir y entrenar modelos.

Performance de los modelos. La utilización de los datos encriptados no debería implicar una perdida en la calidad de los resultados obtenidos por el modelo.

5.1. Evaluación del framework

El framework garantiza que no se pueden identificar registros específicos a partir de las técnicas usadas de encriptación como la homomorfica. Luego de encriptados los datos originales no son accesibles ni reconstrubiles por parte del Data Scientist u otro usuario que no sea el Data Owner.

Está planificado en un corto plazo proveer la funcionalidad de creación de usuarios y un sistema de permisos sobre los datos publicados que permita contar un mecanismo para la gobernanza de datos robusto.

En la implementación que realizamos, el framework utiliza la misma definición de red, el mismo algoritmo de entrenamiento, el mismo optimizador, la misma función de perdida y validación tanto en el modelo de referencia con los datos en claro como en el modelo encriptado. Por lo que cumple el cometido de no agregar una capa de complejidad extra al Data Scientist al momento de crear y entrenar sus modelos.

Tanto en el modelo de referencia como en el encriptado que se utilizaron para medir las distintas performances, se lograron valores de accuracy cercanos al 98%. No se detectaron perdidas en calidad de los resultados de los modelos entrenados por medio del framework analizado.

6. Referencias bibliográficas

[1] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konecny, S. Mazzocchi, B. McMahan, T. Van Overveldt, D. Petrou, D. Ramage y J. Roselander, «Towards federated learning at scale: System design.,» Marzo 2019. [En línea]. Available: arXiv:1902.01046. [Último acceso: 26 03 2021].

[2] J. Konecny, B. McMahan y D. Ramage, «Federated Optimization: Distributed Optimization Beyond the Datacenter.,» Noviembre 2015. [En línea]. Available: arXiv:1511.03575. [Último acceso: 26 03 2021].

[3] J. Konecny, B. McMahan, F. Yu, A. T. Suresh, D. Bacon y P. Richtarik, «Federated Learning: Strategies for improving communication efficiency.,» Octubre 2017. [En línea]. Available: arXiv:1610.05492. [Último acceso: 26 03 2021].

[4] SafetyNet, «SafetyNet Attestation API,» [En línea]. Available: https://developer.android.com/training/safetynet/attestation. [Último acceso: 03 2021].

[5] Z. Ji, Z. Lipton y C. Elkan, «Differential Privacy and Machine Learning: a Survey and Review.,» Diciembre 2014. [En línea]. Available: arXiv:1412.7584. [Último acceso: 26 03 2021].

[6] OpenMined, «PySyft,» [En línea]. Available: https://github.com/OpenMined/PySyft. [Último acceso: 26 03 2021].

[7] OpenMined, «OpenMined,» [En línea]. Available: https://www.openmined.org/. [Último acceso: 26 03 2021].

[8] Google, «Tensorflow Federated,» [En línea]. Available:

https://www.tensorflow.org/federated. [Último acceso: 26 03 2021].

[9] Microsoft, «Differential Privacy Home,» [En línea]. Available: https://www.microsoft.com/en-us/ai/ai-lab-differential-privacy. [Último acceso: 26 03 2021].

[10] A. Agarwal, J. Langford y C. Y. Wei, «Federated Residual Learning,» Marzo 2020. [En línea]. Available: arXiv:2003.12880. [Último acceso: 26 03 2021].

[11] N. Holohan, «IBM Differential Privacy Library: The single line of code that can protect your data,» IBM Research Europe, [En línea]. Available: https://www.ibm.com/blogs/research/2020/06/ibm-differential-privacy-library-the-single-line-of-code-that-can-protect-your-data/. [Último acceso: 26 03 2021].

[12] IBM, «IBM InfoSphere Optim Test Data Fabrication,» [En línea]. Available: https://www.ibm.com/products/infosphere-optim-test-data-fabrication. [Último acceso: 26 03 2021].

[13] K. Tran, «Towardsdatascience,» Julio 2020. [En línea]. Available: https://towardsdatascience.com/introduction-to-ibm-federated-learning-a-collaborative-approach-to-train-ml-models-on-private-data-2b4221c3839. [Último acceso: 26 03 2021].

[14] IBM, «IBM Federated Learning,» [En línea]. Available: https://github.com/IBM/federated-learning-lib. [Último acceso: 26 03 2021].

[15] fate.fedai.org, «fate.fedai.org,» [En línea]. Available: https://fate.fedai.org/. [Último acceso: 26 03 2021].

[16] WebBank, «https://www.webbank.com/,» [En línea]. Available: https://www.webbank.com/. [Último acceso: 26 03 2021].

[17] Apache, «Apache.org,» [En línea]. Available: https://httpd.apache.org/. [Último acceso: 26 03 2021].

[18] «https://w3techs.com/,» [En línea]. Available: https://w3techs.com/technologies/overview/web_server. [Último acceso: 26 03 2021].

[19] OpenMined, «SyfeterText Home,» [En línea]. Available: https://github.com/OpenMined/SyferText. [Último acceso: 01 11 2020].

[20] Ubuntu, «askubuntu.com,» [En línea]. Available: https://askubuntu.com/questions/1241397/ubuntu-20-04-service-autostart. [Último acceso: 01 11 2020].

[21] OpenMined, «PySyft Install Guide,» [En línea]. Available: https://pysyft.readthedocs.io/en/latest/installing.html. [Último acceso: 01 11 2020].

ANEXO 1 – Descripción de la Arquitectura

La red cuenta con 5 servidores:

Cantidad	Nombre	IP	Uso
1	server04	192.168.1.174	Grid Server
2	server00/server01	192.168.1.170/192.168.1.171	Data Node
1	server03	192.168.1.173	Data Scientist
1	server02	192.168.1.172	Reference Environment

Tabla 2 - Descripción Servers

Característica	Descripción
OS	Linux
Version	Ubuntu Server 20.04.1 Focal Fossa
Servername	server00
Private IP	192.168.1.170
OS Username	julito
OS Password	appleauto943
Notebook Port	5425
Public Access	http://julitosa.ddns.net:5425
Notebook Pass	tolaspi943
Grid Name	node00

Tabla 3 - server00 - PySyft Enviroment - DataNode

Característica	Descripción
OS	Linux
Version	Ubuntu Server 20.04.1 Focal Fossa
Servername	server00
Private IP	192.168.1.171
OS Username	julito
OS Password	appleauto943
Notebook Port	5426
Public Access	http://julitosa.ddns.net:5426
Notebook Pass	tolaspi943
Grid Name	node01

Tabla 4 - server01 - PySyft Enviroment - DataNode

Característica	Descripción
OS	Linux
Version	Ubuntu Server 20.04.1 Focal Fossa
Servername	server00
Private IP	192.168.1.173
OS Username	julito
OS Password	appleauto943
Notebook Port	5428
Public Access	http://julitosa.ddns.net:5428

Notebook Pass	tolaspi943
Grid Name	node03

Tabla 5 - server03 - PySyft Enviroment – Data Scientist

Característica	Descripción
OS	Linux
Version	Ubuntu Server 20.04.1 Focal Fossa
Servername	server02
Private IP	192.168.1.172
OS Username	julito
OS Password	appleauto943
Notebook Port	6969
Public Access	http://julitosa.ddns.net:6969
Notebook Pass	tolaspi943
Name	reference

Tabla 6 - server02 - Reference Enviroment

Característica	Descripción
OS	Linux
Version	Ubuntu Server 20.04.1 Focal Fossa
Servername	server04
Private IP	192.168.1.174
OS Username	julito
OS Password	appleauto943
Name	node04

Tabla 7 - server04 - PyGrid - Grid Server

Característica	Descripción
Drive	https://drive.google.com
Username	tesisjulipa
Password	Paratesis1!
Github	https://github.com/tesisjulipa/

Tabla 8 - Repository

ANEXO 2 – Instalación del Ambiente de Referencia

Especificaciones de Software

Característica	Descripción
OS	Linux
Python	3.8.3
Pytorch	1.7.0

Tabla 9 - Características software

Especificaciones de Hardware

Característica	Descripción
CPU	Intel(R) Core(TM) i5-2400S CPU @ 2.50GHz (4 Nucleos)
RAM	12GB
HD	WDC WD2500AAKX-07U6AA0

Tabla 10 - Características hardware

Instalación Imagen Linux

Instalar la siguiente imagen Linux. Se recomienda fijar un swap file tan grande como la memoria física que se tiene. En el caso que el disco sea un SSD duplicar el swap file.

Al momento de crear se va a pedir un usuario inicial, en nuestro caso el usuario que se configuró fue “julito”, y toda referencia a este usuario debe de ser reemplazada a futuro por el usuario que se genera.

Característica	Descripción
Home	https://ubuntu.com/download/server/
Image URL	https://releases.ubuntu.com/20.04.1/ubuntu-20.04.1-live-server-amd64.iso
OS	Linux
Versión	20.04.1 Server

Tabla 11 - Imagen de instalación

Configuración ambiente producción

De ahora en más se asume que el usuario que ejecuta los comandos es “julito” que es el usuario que se generó al momento de instalar el SO.

Característica	Descripción
python3-dev	Headers necesarios para poder compilar librerías en el sistema.
build-essential	Paquete de utilidades para poder compilar en C librerías y nuevos componentes.

Tabla 12 - Dependencias

```
$ sudo apt install python3-dev
$ sudo apt install build-essential
```

Instalar Anaconda

Característica	Descripción
Anaconda	Interprete de notebooks y ambiente para dataScience. Versión con Python 3.8

Tabla 13 - Anaconda

```
$ curl https://repo.anaconda.com/archive/Anaconda3-2020.07-Linux-x86_64.sh --output anaconda.sh
$ bash ./anaconda.sh
$ source ~/.bashrc
```

Instalar PyTorch

Se instala la versión estable al momento de generar este documento anexo (01-11-2020)

```
$ conda install pytorch==1.7.0 torchvision=0.8.1 torchaudio cpuonly -c pytorch
```

Configurar Jupyter Notebook

La versión de Jupyter Notebook que queda instalada es la que se instala en las dependencias de PySyft.

```
$ cd ~
$ mkdir ~/ort
$ jupyter notebook --generate-config
```

Luego hay que generar un SHA1 con una password ingresada por consola.

```
$ python
Python 3.8.3 (default, Jul  2 2020, 16:21:59)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>from notebook.auth import passwd; passwd()
Enter password:
Verify password:
'sha1:790d7d736482:64981edfae406ecf8ee5b4e962b09946b8e99e1e'
>>>quit()
```

Se edita el archivo de configuración previamente generado.

```
$ vi /home/julito/.jupyter/jupyter_notebook_config.py
```

Se editan las siguientes entradas.

```
c.NotebookApp.password                                                    =
'sha1:0b1c69469a08:7f3bc8efb001b944309486d9f7dc2c984c3d13d0'
c.NotebookApp.port = 6969
c.ConnectionFileMixin.ip = '0.0.0.0'
c.NotebookApp.ip = '0.0.0.0'
```

Configurar Startup Service

En el ejemplo de configuración el usuario se llama “julito” y debe reemplazarse por el que se configuró cuando se instala el SO. El archivo de configuración fue tomado de referencia de la página de Ubuntu. [20]

```
$ sudo vi /etc/systemd/system/jupyterlab.service
```

```
[Unit]
Description=Jupyter Lab Server

[Service]
User=julito
Group=julito
Type=simple
WorkingDirectory=/home/julito/ort
ExecStart=/home/julito/anaconda3/bin/jupyter-notebook                    --
config=/home/julito/.jupyter/jupyter_notebook_config.py
StandardOutput=null
Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target
```

Configurar Firewall

Se deben abrir los puertos del firewall por los que se va a acceder.

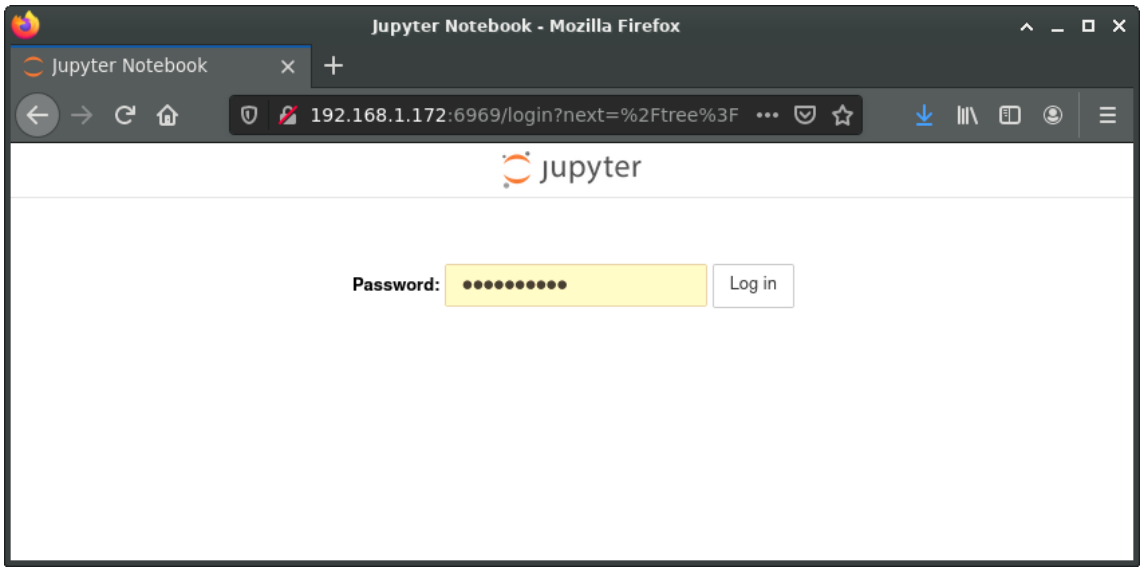
```
$ sudo ufw allow 6969
$ sudo ufw allow OpenSSH
```

Configurar Git

```
$ git config --global user.name "tesisjulipa"
$ git config --global user.email "tesisjulipa@gmail.com"
$ git config --global -list
user.name=tesisjulipa
user.email=tesisjulipa@gmail.com
```

```
$ cd ~/ort
$ git clone https://github.com/tesisjulipa/references
$ git clone https://github.com/tesisjulipa/syft
```

Probar Acceso



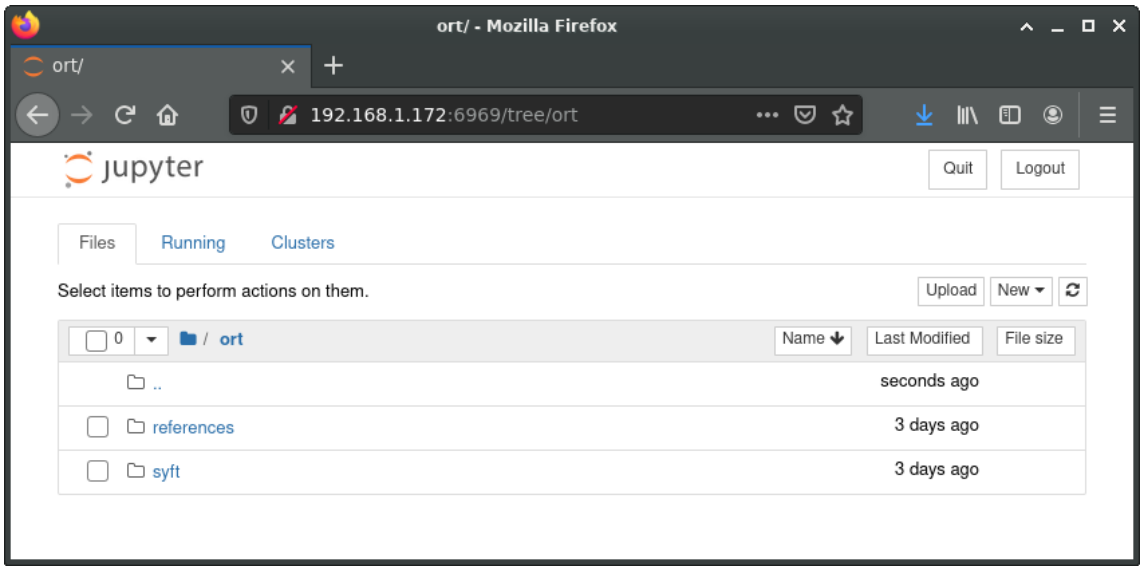


Tabla 14 - Test de acceso exitoso

ANEXO 3 – Instalación del Ambiente de PySyft

Instalación Imagen Linux

Instalar la siguiente imagen de Linux. Se recomienda fijar un swap file tan grande como memoria física se tiene. En el caso que el disco sea un SSD duplicar el swap file.

Al momento de crear se va a pedir configurar un usuario inicial, en nuestro caso el usuario que se configuró fue "julito", y toda referencia a este usuario debe ser reemplazada a futuro por el usuario que se genera.

Característica	Descripción
Home	https://ubuntu.com/download/server/
Image URL	https://releases.ubuntu.com/20.04.1/ubuntu-20.04.1-live-server-amd64.iso
OS	Linux
Versión	20.04.1 Server

Tabla 15 - Imagen de instalación

Configuración ambiente de producción

De ahora en más se asume que el usuario que ejecuta los comandos es "julito" que es el usuario que se generó al momento de instalar el SO.

Característica	Descripción
python3-dev	Headers necesarios para poder compilar librerías en el sistema.
build-essential	Paquete de utilidades para poder compilar en C librerías y nuevos componentes.

Tabla 16 - Dependencias

```
$ sudo apt install python3-dev
$ sudo apt install build-essential
```

Instalar Anaconda

Característica	Descripción
Anaconda	Interprete de notebooks y ambiente para dataScience. Versión con Python 3.8

Tabla 17 - Anaconda

```
$ curl https://repo.anaconda.com/archive/Anaconda3-2020.07-Linux-x86_64.sh --output anaconda.sh
$ bash ./anaconda.sh
$ source ~/.bashrc
```

Se instala la versión estable al momento de generar este documento anexo (01-11-2020).

```
$ conda install pytorch==1.7.0 torchvision=0.8.1 torchaudio cpuonly -c pytorch
```

Instalar PySyft

Basado en la guía brindada por OpenMined [21]

```
$ cd ~
$ git clone https://github.com/OpenMined/PySyft.git
$ cd PySyft
$ pip install -r pip-dep/requirements.txt
$ python setup.py install
```

Instalar SyferText

Basado en el proceso de configuración github [19]

```
$ cd ~
$ curl -s https://packagecloud.io/install/repositories/github/git-lfs/script.deb.sh | sudo bash
$ sudo apt-get install git-lfs
$ git lfs install

$ pip install git+git://github.com/Nilanshrajput/syfertext_en_core_web_lg@master
$ git clone https://github.com/OpenMined/SyferText.git
$ cd SyferText
$ python setup.py install
```

Configurar Jupyter Notebook

La versión de Jupyter Notebook que queda instalada es la que se instala en las dependencias de PySyft.

```
$ cd ~
$ mkdir ~/ort
$ jupyter notebook --generate-config
```

Luego hay que generar un SHA1 con una password ingresada por consola.

```
$ python
Python 3.8.3 (default, Jul  2 2020, 16:21:59)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>from notebook.auth import passwd; passwd()
Enter password:
Verify password:
'sha1:790d7d736482:64981edfae406ecf8ee5b4e962b09946b8e99e1e'
>>>quit()
```

Se edita el archivo de configuración previamente generado.

```
$ vi /home/julito/.jupyter/jupyter_notebook_config.py
```

Se editan las siguientes entradas.

```
c.NotebookApp.password                                                    =
'sha1:0b1c69469a08:7f3bc8efb001b944309486d9f7dc2c984c3d13d0'
c.NotebookApp.port = 5425
c.ConnectionFileMixin.ip = '0.0.0.0'
```

En este ejemplo se configura la primera máquina del cluster que va a quedar en el puerto 5425.

Configurar Startup Service

En el ejemplo de configuración el usuario se llama “julito” y debe reemplazarse por el que se configuró cuando se instala el SO. El archivo de configuración fue tomado de referencia de la página de Ubuntu. [20]

```
$ sudo vi /etc/systemd/system/jupyterlab.service
```

```
[Unit]
Description=Jupyter Lab Server

[Service]
User=julito
Group=julito
Type=simple
WorkingDirectory=/home/julito/ort
ExecStart=/home/julito/anaconda3/bin/jupyter-notebook                    --
config=/home/julito/.jupyter/jupyter_notebook_config.py
StandardOutput=null
Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target
```

```
$ sudo systemctl start jupyterlab
$ sudo systemctl enable jupyterlab
```

Configurar Firewall

Se deben abrir los puertos del firewall por los que se va a acceder.

```
$ sudo ufw allow 5425
$ sudo ufw allow OpenSSH
```

Configurar Git

```
$ git config --global user.name "tesisjulipa"
$ git config --global user.email "tesisjulipa@gmail.com"
$ git config --global –list
user.name=tesisjulipa
user.email=tesisjulipa@gmail.com
```

```
$ cd ~/ort
$ git clone https://github.com/tesisjulipa/references
$ git clone https://github.com/tesisjulipa/syft
```

Probar Acceso

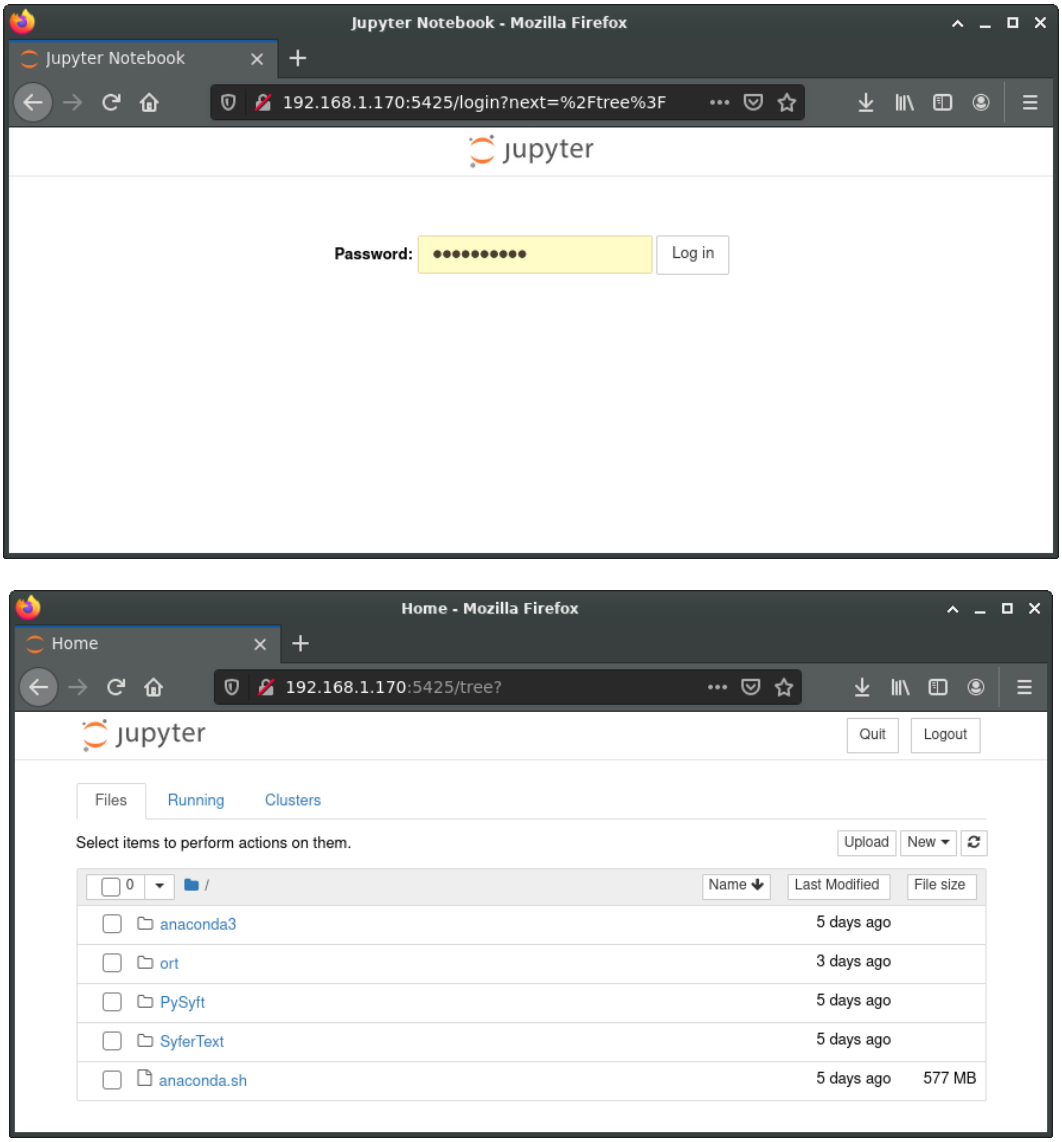


Ilustración 12 - Test de acceso exitoso

ANEXO 4 – Instalación de la Red PyGrid

El objetivo de esta guía es la instalación de la red PyGrid en los ambientes donde se tenga configurado PySyft para poder comunicarse e interactuar entre ellos, actuando como nodos y workers. En esta guía se va a instalar y configurar uno de los nodos, pero la configuración va a ser completa para la red que ya se tiene, para evitar confusiones.

Definición del Ambiente

La red PyGrid precisa de un servidor concentrador, el cual va a administrar los diferentes clientes y hacer el nexo entre ellos. Si bien es posible habilitar protocolos de seguridad de usr/pwd con JVT, para nuestro caso decidimos quitar esta capa de complejidad.

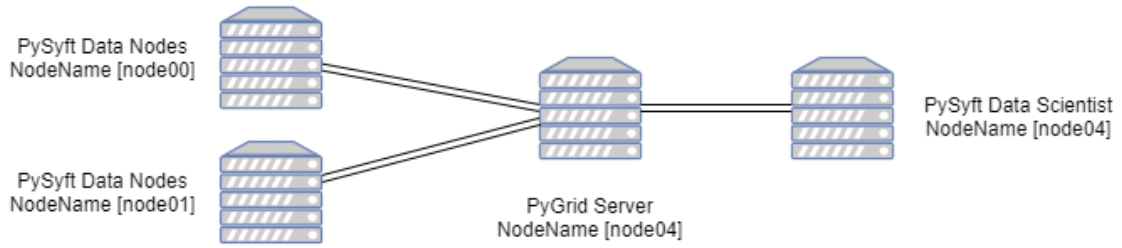


Ilustración 13 - Ambiente de Producción

Instalación Imagen Linux

Instalar la siguiente imagen de Linux. Se recomienda fijar un swap file tan grande como la memoria física que se tiene. En el caso que el disco sea un SSD duplicar el swap file. Al momento de crear se va a pedir configurar un usuario inicial, en nuestro caso el usuario que se configuró fue "julito", y toda referencia a este usuario debe de ser reemplazada a futuro por el usuario que se genera.

Característica	Descripción
Home	https://ubuntu.com/download/server/
Image URL	https://releases.ubuntu.com/20.04.1/ubuntu-20.04.1-live-server-amd64.iso
OS	Linux
Versión	20.04.1 Server

Tabla 18 - Imagen de instalación

Configuración Servidor PyGrid

De ahora en más se asume que el usuario que ejecuta los comandos es “julito” que es el usuario que se generó al momento de instalar el SO.

El servidor de PyGrid es el encargado de conectar y administrar los distintos servicios de la red que los data scientist y los data owner van a poder acceder.

Instalación MySQL

Al momento de poder comunicarse y persistir las diferentes configuraciones hay que instalar una BD donde el server pueda acceder.

```
$ sudo apt install mysql-client
$ sudo apt install mysql-server
$ sudo mysql_secure_installation
```

```
$ sudo vi /etc/mysql/mysql.conf.d/mysqld.cnf
```

Cambiamos la entrada de bind-address para que sea accesible por toda la red.

```
bind-address            = 0.0.0.0
```

Configuramos un usuario local en MySQL que sea accesible por toda la red. La password de root la pidió al momento de configurar el server por primera vez.

```
$ sudo mysql -u root -p
mysql> create user 'pygrid'@'%' idenfitied by 'pygrid';
mysql> grant all privileges on pygrid.* to 'pygrid'@'%';
```

```
$ sudo ufw allow 3306
$ sudo systemctl start mysql
```

Configurar los host de la red

Para que la red se pueda comunicar entre sí, o bien se cuenta con DNS interno para resolver los nombres correctamente, o se modifica el archivo hosts donde se informan la IP con el nombre del server.

```
$ sudo vi /etc/hosts
```

Agregar las líneas al final:

```
192.168.1.170 server00 node00
192.168.1.171 server01 node01
192.168.1.172 server02 rerefence
192.168.1.173 server03 node03
192.168.1.174 server04 grid00
```

Instalar PyGrid

```
$ git clone https://github.com/OpenMined/PyGrid
$ cd PyGrid
$ python ./setup.py install
```

Se instala poetry para poder instalar todas las dependencias de PyGrid.

```
$ pip install poetry
```

```
$ cd apps/network
$ poetry install
```

```
$ sudo apt-get install libmysqlclient-dev
$ poetry add mysqlclient
```

Configurar PyGrid

```
$ vi /home/julito/PyGrid/apps/network/service.sh
```

```
#!/bin/bash
. ~julito/.bashrc

cd ~julito/PyGrid/apps/network

export DATABASE_URL='mysql://pygrid:pygrid@192.168.1.174/pygrid'

/home/julito/anaconda3/bin/poetry run python -m src "$@"
```

```
$ chmod 777 /home/julito/PyGrid/apps/network/service.sh
```

```
$ sudo vi /etc/systemd/system/pygrid.service
```

```
[Unit]
Description=PyGrid Network

[Service]
User=julito
Group=julito
Type=simple
WorkingDirectory=/home/julito/
ExecStart=/home/julito/PyGrid/apps/network/service.sh --port 7000
StandardOutput=syslog
Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target
```

```
$ sudo ufw allow 7000
$ sudo systemctl enable pygrid
$ sudo systemctl start pygrid
```

Configuración Nodo PyGrid

De ahora en más se asume que el usuario que ejecuta los comandos es “julito” que es el usuario que se generó al momento de instalar el SO.

Configurar los host de la red

Para que la red se pueda conectar entre sí, al igual que lo comentado en el server, o bien se cuenta con DNS interno para resolver los nombres correctamente, o se modifica el archivo hosts donde se informan la IP con el nombre del server.

```
$ sudo vi /etc/hosts
```

Agregar las líneas al final:

```
192.168.1.170 server00 node00
192.168.1.171 server01 node01
192.168.1.172 server02 rerefence
192.168.1.173 server03 node03
192.168.1.174 server04 grid00
```

Instalar PyGrid

```
$ git clone https://github.com/OpenMined/PyGrid
$ cd PyGrid
$ python ./setup.py install
```

Se instala poetry para poder instalar las dependencias de PyGrid.

```
$ pip install poetry
```

```
$ cd apps/ node
$ poetry install
```

```
$ vi ./pyproject.toml
```

La versión que preinstala de Syft es 0.2.9 y en los servers la versión estable instalada es 0.2.8 por lo que hay que hacer un downgrade.

```
syft = "0.2.8"
```

```
$ poetry install
```

Configurar PyGrid

```
$ vi /home/julito/PyGrid/apps/node/service.sh
```

```
#!/bin/bash
. ~julito/.bashrc

cd ~julito/PyGrid/apps/node

export DATABASE_URL='mysql://pygrid:pygrid@192.168.1.174/pygrid'

/home/julito/anaconda3/bin/poetry run python -m src "$@"
```

```
$ chmod 777 /home/julito/PyGrid/apps/node/service.sh
```

```
$ sudo vi /etc/systemd/system/pygrid.service
```

Lo marcado con resaltador amarillo es el nombre de la red, es un nombre lógico que se debe de corresponder con el nombre del host configurado al inicio. Y en los ejemplos es “bob”, “alice”, etc.

Lo marcado con resaltador verde es el server concentrador y el puerto donde quedó configurado PyGrid.

```
[Unit]
Description=PyGrid Network

[Service]
User=julito
Group=julito
Type=simple
WorkingDirectory=/home/julito/
ExecStart=/home/julito/PyGrid/apps/node/service.sh    --id    node00    --host
192.168.1.170 --port 5000 --network='http://192.168.1.174:7000'
StandardOutput=null
Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target
```

```
$ sudo ufw allow 5000
$ sudo systemctl enable pygrid
$ sudo systemctl start pygrid
```

ANEXO 5 – Modelo de Referencia

El dataset está compuesto por un conjunto de URL previamente clasificadas como "buenas" y "malas". El objetivo es entrenar una red (RNN) para la correcta determinación de la clase. El enfoque que se va a tomar es hacer una red para determinar un análisis del sentimiento para poder determinar, en base a las palabras usadas, la clase de estas.

	# Lines	% Lines
Buenas	1294531	96%
Malas	48126	4%

Tabla 19 - Distribución de la clase

Paso 1 – Importar los archivos y crear 2 listas

```
with open('./raw/badqueries.unix', 'r', encoding='utf-8') as dataset_file:
    bads = dataset_file.readlines()
with open('./raw/goodqueries.unix', 'r', encoding='utf-8') as dataset_file:
    goods = dataset_file.readlines()


reviews = bads + goods
labels  = ([1] * len(bads)) + ([0] * len(goods))
```

Se obtienen 2 listas, review y labels con la misma cantidad de registros.

Paso 2 – Tokenizar todas las posibles palabras del dataset

```
all_reviews=list()
for text in reviews:
  text = text.lower()
  for a in punctuation:
      text = text.replace(a, ' ')
  all_reviews.append(text)
all_text = " ".join(all_reviews)
all_words = all_text.split()
```

Paso 3 – Crear el diccionario de palabras

```
from collections import Counter

# Vamos a contar todas las palabras que hay contando el metodo counter
count_words  = Counter(all_words)      # Array con el distinct de palabras
total_words  = len(all_words)          # Numero total de palabras en el datase
sorted_words = count_words.most_common(total_words) # Lista ordenadas por cantidad de ocurrencias de palabras unicas

# Al final obtenemos un dict en donde el key es la palabra y el valor es numerador de palabra
# Esta lista la vamos a usar luego para reemplazar las palabras por su numero dentro de las
# posibles palabras y usar los numeros en el tensor
vocab_to_int={w:i+1 for i,(w,c) in enumerate(sorted_words)}
```

Paso 4 – Transformar la lista de palabras a una lista de array de números

```
# El objetivo es pasar la lista inicial de palabras del dataset y reemplazar la palabra
# por su valor numerico de palabra del diccionario de transformacion
# Al final encoded_reviews va a tener una lista con las lineas del dataset encoded
# en formato numerico

encoded_reviews=list()
for review in all_reviews:
  encoded_review=list()
  for word in review.split():
    if word not in vocab_to_int.keys():
      encoded_review.append(0) # Si la palabra no existe poner 0 para que no de error
    else:
      encoded_review.append(vocab_to_int[word])
  encoded_reviews.append(encoded_review)
```

Paso 5 – Crear una lista con el tamaño del tensor para la RNN

```
# Ahora vamos a crear un vector, con la cantidad constante de numeros para todas las lineas
# Esto porque este va a ser la entrada a la red y tiene que ser un numero constante

sequence_length=250
features=np.zeros((len(encoded_reviews), sequence_length), dtype=int)
for i, review in enumerate(encoded_reviews):
  review_len=len(review)
  if (review_len<=sequence_length):
    zeros=list(np.zeros(sequence_length-review_len))
    new=zeros+review
  else:
    new=review[:sequence_length]
  features[i,:]=np.array(new)
```

Paso 6 – Split Dataset Train/Test

```
# Split los dataset, 80% para training y 20% para test
from sklearn.model_selection import train_test_split

train_x, valid_x, train_y, valid_y = train_test_split(features, labels, test_size=0.2, random_state=666)

# La distribucion de clases se tiene que mantener (96% Clase 0)
print("Numero de Elementos [{}] Numero de Clase 0 [{}] Perc [{}]".format(len(train_y), train_y.count(0), train_y.count(0)/ler
print("Numero de Elementos [{}] Numero de Clase 0 [{}] Perc [{}]".format(len(valid_y), valid_y.count(0), valid_y.count(0)/ler
```

```
Numero de Elementos [1074125] Numero de Clase 0 [1035675] Perc [0.9642034213895031]
Numero de Elementos [268532] Numero de Clase 0 [258856] Perc [0.963967050481879 3]
```

Paso 7 – Crear los Tensores y Dataset para PyTorch

```
import torch
from torch.utils.data import DataLoader, TensorDataset

#create Tensor Dataset
train_data=TensorDataset(torch.FloatTensor(train_x), torch.FloatTensor(train_y))
valid_data=TensorDataset(torch.FloatTensor(valid_x), torch.FloatTensor(valid_y))

#dataloader
batch_size=32
trainloader=DataLoader(train_data, batch_size=batch_size, shuffle=True)
validloader=DataLoader(valid_data, batch_size=batch_size, shuffle=True)
```

Paso 8 – Definición de RNN

```
class Classifier(torch.nn.Module):
    def __init__(self, in_features, out_features):
        super(Classifier, self).__init__()
        self.fc = torch.nn.Linear(in_features, out_features)

    def forward(self, x):
        logits = self.fc(x)
        probs = F.relu(logits)
        return probs, logits
```

```
classifier = Classifier(in_features = 250, out_features = 2)
print(classifier)
```

```
Classifier(
  (fc): Linear(in_features=250, out_features=2, bias=True)
)
```

Paso 9 – Optimizador

```
import torch.optim as optim
optim = optim.SGD(params = classifier.parameters(), lr = 0.001)
```

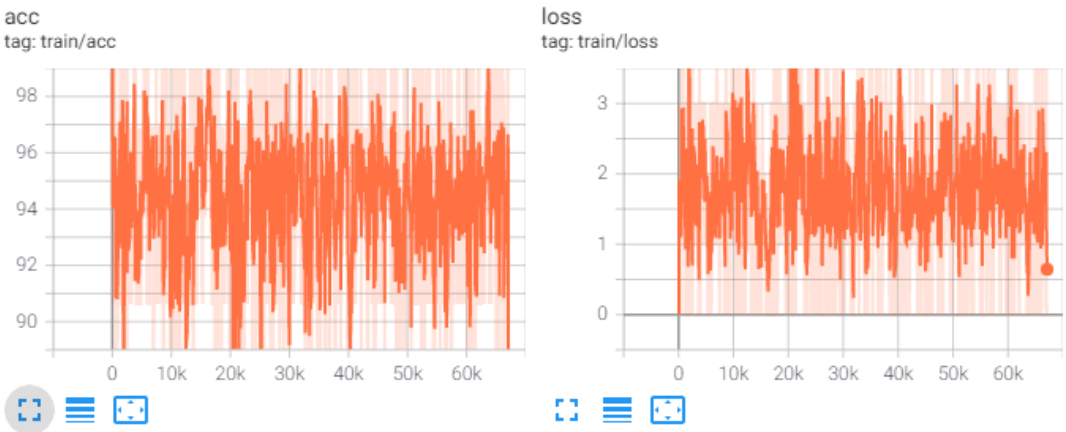
Paso 10 – Entrenar el Modelo

```
import torch.nn.functional as F
from tqdm import tqdm

writer = SummaryWriter()

iter     = 0
epochs   = 2

acc_list  = list()
loss_list = list()

for epoch in range(epochs):

    print("Doing Epoch [{}]".format(epoch))
    for inputs, labels in tqdm(trainloader):
        iter += 1

        classifier.train()
        optim.zero_grad()

        # Predict sentiment probabilities
        probs, logits = classifier(inputs)

        target  = torch.tensor(labels, requires_grad=True)
        loss    = (( probs.argmax(dim=1) -  target)**2).sum()

        preds   = probs.argmax(dim=1)
        targets = target

        accuracy = (preds == targets).sum().float()
        accuracy = 100 * (accuracy / batch_size)

        loss.backward()
        optim.step()

        writer.add_scalar('train/loss', loss,     epoch * iter )
        writer.add_scalar('train/acc',  accuracy, epoch * iter )
```

Paso 11 – Validar el Modelo con los datos

```
val_iter = 0
classifier.eval()
for inputs, labels in tqdm(validloader):
    val_iter =+ 1

    probs, logits = classifier(inputs)
    loss = ((probs.argmax(dim=1) -  labels)**2).sum().float()

    preds   = probs.argmax(dim=1)
    targets = labels

    accuracy = preds.eq(targets).sum().float()
    accuracy = 100 * (accuracy / batch_size)

    # Log to tensorboard
    writer.add_scalar('val/loss', loss,     val_iter)
    writer.add_scalar('val/acc',  accuracy, val_iter)
```

Resultados del train



ANEXO 6 – Modelo de SyferText

Paso 1 – Importar los archivos y crear 2 listas

```
with open('./raw/badqueries.unix', 'r', encoding='utf-8') as dataset_file:
    bads = dataset_file.readlines()
with open('./raw/goodqueries.unix', 'r', encoding='utf-8') as dataset_file:
    goods = dataset_file.readlines()


reviews = bads + goods
labels  = ([1] * len(bads)) + ([0] * len(goods))
```

Se obtienen 2 listas, review y labels con la misma cantidad de registros.

Paso 2 – Tokenizar todas las posibles palabras del dataset

```
all_reviews=list()
for text in reviews:
  text = text.lower()
  for a in punctuation:
      text = text.replace(a, ' ')
  all_reviews.append(text)
all_text = " ".join(all_reviews)
all_words = all_text.split()
```

Paso 3 – Crear el diccionario de palabras

```
from collections import Counter

# Vamos a contar todas las palabras que hay contando el metodo counter
count_words  = Counter(all_words)      # Array con el distinct de palabras
total_words  = len(all_words)          # Numero total de palabras en el datase
sorted_words = count_words.most_common(total_words) # Lista ordenadas por cantidad de ocurrencias de palabras unicas

# Al final obtenemos un dict en donde el key es la palabra y el valor es numerador de palabra
# Esta lista la vamos a usar luego para reemplazar las palabras por su numero dentro de las
# posibles palabras y usar los numeros en el tensor
vocab_to_int={w:i+1 for i,(w,c) in enumerate(sorted_words)}
```

Paso 4 – Crear los Workers

```
# Create a torch hook for PySyft
hook = sy.TorchHook(torch)

# Create some PySyft workers
me = hook.local_worker # This is the worker representing the deep learning company
bob = sy.VirtualWorker(hook, id = 'bob') # Bob owns the first dataset
alice = sy.VirtualWorker(hook, id = 'alice') # Alice owns the second dataset

crypto_provider = sy.VirtualWorker(hook, id = 'crypto_provider') # provides encryption primitive for SMPC

# Create a summary writer for logging performance with Tensorboard
writer = SummaryWriter()
```

Se crea workers para Alice, Bob, Company, y un worker extra para el proveedor de encriptación.

Paso 5 – Creación Dict

```
# Crear un dataset_local que es un dict con 2 entradas, review y label
# Key['review'] -> El texto entero
# Key['label']  -> La label Codificada 0/1

dataset_local = []

for i in range(len(all_reviews)):
    example = dict(review = String(all_reviews[i]), label = labels[i])
    dataset_local.append(example)
```

Se crea un dataset, de tipo dict, con 2 campos review y label Review contiene la lista de palabras y label es 0 para positivo, 1 para negativo.

Paso 6 – Separación de datos Bob vs Alice

```
# Create two datasets, one for Bob, and the other for Alice

a, b = train_test_split(dataset_local, train_size = 0.10)
dataset_bob, dataset_alice = train_test_split(a, train_size = 0.5)

# dataset_bob, dataset_alice = train_test_split(dataset_local, train_size = 0.5)

# Now create a validation set for Bob, and another for Alice
train_bob,   val_bob   = train_test_split(dataset_bob, train_size = 0.7)
train_alice, val_alice = train_test_split(dataset_alice, train_size = 0.7)
```

Se separa el dataset grande en 2 partes distribuidas equitativamente para Bob y para Alice, así los 2 trabajan con diferentes datos.

Paso 7 – Encriptar los Datasets

```
# A function that sends the content of each split to a remote worker
def make_remote_dataset(dataset, worker):

    # Got through each example in the dataset
    for example in tqdm(dataset):

        # Send each review text
        example['review'] = example['review'].send(worker)

        # Send each label as a one-hot-enceded vector
        one_hot_label = torch.zeros(2).scatter(0, torch.Tensor([example['label']]).long(), 1)

        # Send the review label
        example['label'] = one_hot_label.send(worker)
```

```
# Bob's remote dataset
make_remote_dataset(train_bob, bob)
make_remote_dataset(val_bob,   bob)

# Alice's remote dataset
make_remote_dataset(train_alice, alice)
make_remote_dataset(val_alice,   alice)
```

```
100%|██████████| 46992/46992 [00:27<00:00, 1678.98it/s]
100%|██████████| 20140/20140 [00:13<00:00, 1519.72it/s]
100%|██████████| 46993/46993 [00:27<00:00, 1693.41it/s]
100%|██████████| 20140/20140 [00:11<00:00, 1696.70it/s]
```

El objetivo de este paso es encriptar los datasets de Bob y Alice.

Paso 8 – Objeto NLP

```
# Create a Language object with SyferText
nlp = syfertext.load('en_core_web_lg', owner = me)
```

```
nlp.pipeline_template
```

```
[{'remote': True, 'name': 'tokenizer'}]
```

```
use_stop_tagger = False
use_polarity_tagger = False

# Tokens with these custom tags
# will be excluded from creating
# the Doc vector
excluded_tokens = {}
```

Se crea el objeto nlp, que es el objeto que el data scientist va a pasarle a Bob y Alice para el preprocesamiento de los datos. Para nuestro ejemplo no se usa la eliminación de stoppers, ni palabras polarizadas.

Paso 9 – Definición de Dataset en Company

```
# Instantiate a training Dataset object
trainset = DatasetIMDB(sets = [train_bob,
                               train_alice],
                       share_workers = [bob, alice],
                       crypto_provider = crypto_provider,
                       nlp = nlp
                      )

# Instantiate a validation Dataset object
valset = DatasetIMDB(sets = [val_bob,
                             val_alice],
                     share_workers = [bob, alice],
                     crypto_provider = crypto_provider,
                     nlp = nlp
                    )
```

Acá es donde el data scientist genera un dataset mezclando los datos de Bob y Alice en un único dataset para entrar.

Paso 10 - Train

El train del modelo es idéntico al modelo de referencia.