Máster en Seguridad de las TIC

Machine Learning en Ciberseguridad

Trabajo Fin de Máster

Curso 2018-2019

Alejandro Fernández Khatiboun
Máster Universitario en la Seguridad de las TIC
Análisis de Datos

Profesor responsable de la asignatura
Enric Hernández Jiménez

4 de junio de 2019

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Machine learning en la ciberseguridad*
Nombre del autor:	*Alejandro Fernández Khatiboun*
Nombre del consultor/a:	-
Nombre del PRA:	*Enric Hernández Jiménez*
Fecha de entrega	06/2019
Titulación:	*Máster universitario en la Seguridad de las TIC*
Área del Trabajo Final:	*Análisis de Datos*
Idioma del trabajo:	*Castellano*
Palabras clave	*Machine Learning, Detección de fraude, Ciberseguridad*

Resumen del Trabajo:

El presente trabajo está centrado en la evaluación y análisis de algoritmos de machine learning capaces de detectar un uso fraudulento en las transacciones realizadas con tarjetas bancarias.

Se realizará un estudio de diferentes algoritmos de machine learning, de tal forma que, tras seleccionar uno de los algoritmos de Machine learning, se consiga determinar qué tipo de transacciones son fraudulentas. De la misma manera se compararán los diferentes resultados obtenidos por cada uno de los algoritmos.

Para ello, se utilizará un dataset en el que se registran 284.807 transacciones realizadas en dos días, de las cuales 492 son fraudulentas, con el fin entrenar a los modelos.

Abstract:

This paper explores the analysis and evaluation of machine larning algorithms that are able to recognize fraudulent credit card transactions.

A study will be conducted about the different machine learning algorithms to determine the fraudulent transactions. Similarly, the results of the different algorithms will be compared to obtain a conclusion.

To realice the study, a Credit Card Fraud dataset is used. This dataset contains 284,807 different transactions realized along two days. 492 of these transactions are fraudulent.

Índice

1. Introducción 1
1.1 Contexto y justificación del Trabajo 1
1.2 Objetivos del Trabajo 2
1.3 Enfoque y método seguido 2
1.4 Planificación del Trabajo (Añadir tabla con fechas) 1
1.5 Breve descripción de los otros capítulos de la memoria 1
2. Estado del arte 2
2.1 Introducción. 2
2.2 ¿Qué es Machine Learning? 3
2.3 Tipos de sistemas de Machine Learning 4
2.4 Aplicaciones en la ciberseguridad y en el uso fraudulento de tarjetas bancarias 7
3. Tecnologías y productos utilizados 8
3.1 Lenguaje de programación 8
3.2 Librerías 9
3.3 Aplicaciones 10
3.4 Datos 11
4. Análisis del sistema 12
4.1 Fases del sistema 12
4.2 Algoritmo del sistema 15
5. Desarrollo del sistema. 17
5.1 Configuración del entorno. 18
5.2 Obtención de los datos. 21
5.3 Preprocesado de los datos. 22
5.4 Estudio de los datos. 26
5.5 Estudio del modelo y realización de pruebas 28
5.5.1 Random Forest 28
5.5.2 Isolation Forest 34
5.6 Análisis del resultado. 37
6. Conclusiones 39
7. Bibliografía 40

Listado de Figuras

Figura 1 Planificación del trabajo 1
Figura 2 Sistema básico de aprendizaje automático. 3
Figura 3 Esquema de modelo de aprendizaje supervisado. 5
Figura 4 Esquema modelo de aprendizaje no supervisado 6
Figura 5 Esquema modelo de aprendizaje por refuerzo 6
Figura 6 Logo Python 8
Figura 7 Porcentaje de uso de lenguajes de programación en ML 9
Figura 8 Logo Pandas 9
Figura 9 Logo Sckit Learn 9
Figura 10 Logo Numpy 10
Figura 11 Logo Matplotlib 10
Figura 12 Logo Jupyter 10
Figura 13 Logo Anaconda 11
Figura 14 Fases de un sistema de Machine Learning 12
Figura 15 Selección de algoritmos propuesto por Sckit-Learn 15
Figura 16 Ejemplo de representación IF 16
Figura 17 Distribución Anaconda 18
Figura 18 Anaconda Navigator 19
Figura 19 Ejemplo de Jupyter Notebook 19
Figura 20 Ejemplo importación de librerías Python 20
Figura 21 Tabla Datos del Dataset 21
Figura 22 Código Python. Fichero I. Preparación 22
Figura 23 Ficheros datos entrenamiento y test 25
Figura 24 Histogramas de variables 26
Figura 25 Datos estadísticos del Dataset 27
Figura 26 Matriz de correlación de variables v1 a v28 27
Figura 27 Ejemplo gráfico de Random Forest 29
Figura 28 Ejemplo algoritmo Random Forest 30
Figura 29 Fórmula impureza de Gini 30
Figura 30 Ejemplo de propiedades de Random Forest 31
Figura 31 Código Python: Fichero II. Solución Random Forest 32
Figura 32 Ejemplo gráfico Isolation Forest 34
Figura 33 Código Python: Fichero III. Solución Isolation Forest 35
Figura 34 Resultados de tiempo y precisión de Random Forest 37
Figura 35 Resultados de tiempo y precisión de Isolation Forest 38

1. Introducción

1.1 Contexto y justificación del Trabajo

En los últimos años, la inteligencia artificial ha sido uno de los temas más hablados, sin embargo, lleva con nosotros desde finales de los años 50. Ha estado latente en los laboratorios de investigación desde que unos científicos de la informática se reunieron alrededor del término en las Conferencias de Dartmouth en 1956 y dieron a luz el campo de la Inteligencia artificial.

Tras los avances conseguidos en los últimos años, la tecnología está lista para dar este salto y llevarla a sectores como el automovilístico, energía, industria, bancario, sanidad, defensa y ciberseguridad.

El Machine Learning en su forma más básica es la práctica de usar algoritmos para analizar datos, aprender de ellos y luego hacer una determinación o predicción sobre algo en el mundo.

La industria de la ciberseguridad no ha tardado en investigar, desarrollar y emplear el uso de machine learning como respuesta a los retos actuales Estas prácticas ayudan a analizar mejor las amenazas y ser más efectivas a la hora de detener los incidentes de seguridad. Ejemplos de aplicaciones de IA en la ciberseguridad son: detección de intrusos, clasificación de malware, detección de fraude de tarjetas bancarias, detección de ataques DDoS, etc.

1.2 Objetivos del Trabajo

- Realizar un estudio previo de Machine Learning, detallando los diferentes tipos de sistemas existentes, y su inclusión en la ciberseguridad.
- Realizar y estudiar una selección algoritmos de Machine Learning útiles para el caso en concreto.
- Realizar y explicar cada una de las fases del proceso de Machine Learning para el sistema propuesto.
- Realizar un entrenamiento del modelo seleccionado mediante los algoritmos Isolation Forest y Random Forest.
- Valoración de los resultados obtenidos

1.3 Enfoque y método seguido

Para la realización de este trabajo es necesario realizar un estudio previo de los algoritmos de machine learning para la predicción del uso fraudulento de tarjetas bancarias.

Será necesario también localizar una base de datos o, mejor dicho, un dataset que contenga información acerca de registros del uso fraudulento y no de tarjetas bancarias.

Una vez se haya realizado un estudio de viabilidad de los algoritmos a utilizar y hallado un dataset, se procederá a realizar un entrenamiento del modelo. Para ello, en primer lugar, se deberán de preparar los datos, en caso de que no lo estén, como eliminar atributos que no sean necesarios, tener los datos en el formato adecuado (eliminar ruido), normalizar los datos, etc. De esta forma, se optimizará el proceso de entrenamiento del modelo, logrando mejores resultados.

A continuación, dividiremos el data-set en dos subconjuntos de datos para trabajar. Uno para entrenar al modelo y otro para evaluarlo. Para entrenar los modelos utilizaremos el lenguaje de programación Python y librerías como, pandas, numpy, scikit-learn…orientadas al cálculo matemático y al machine learning. Gracias a estas librerías podremos analizar los resultados obtenidos y seleccionar el algoritmo más acorde a la solución que se plantea en este trabajo.

1.4 Planificación del Trabajo

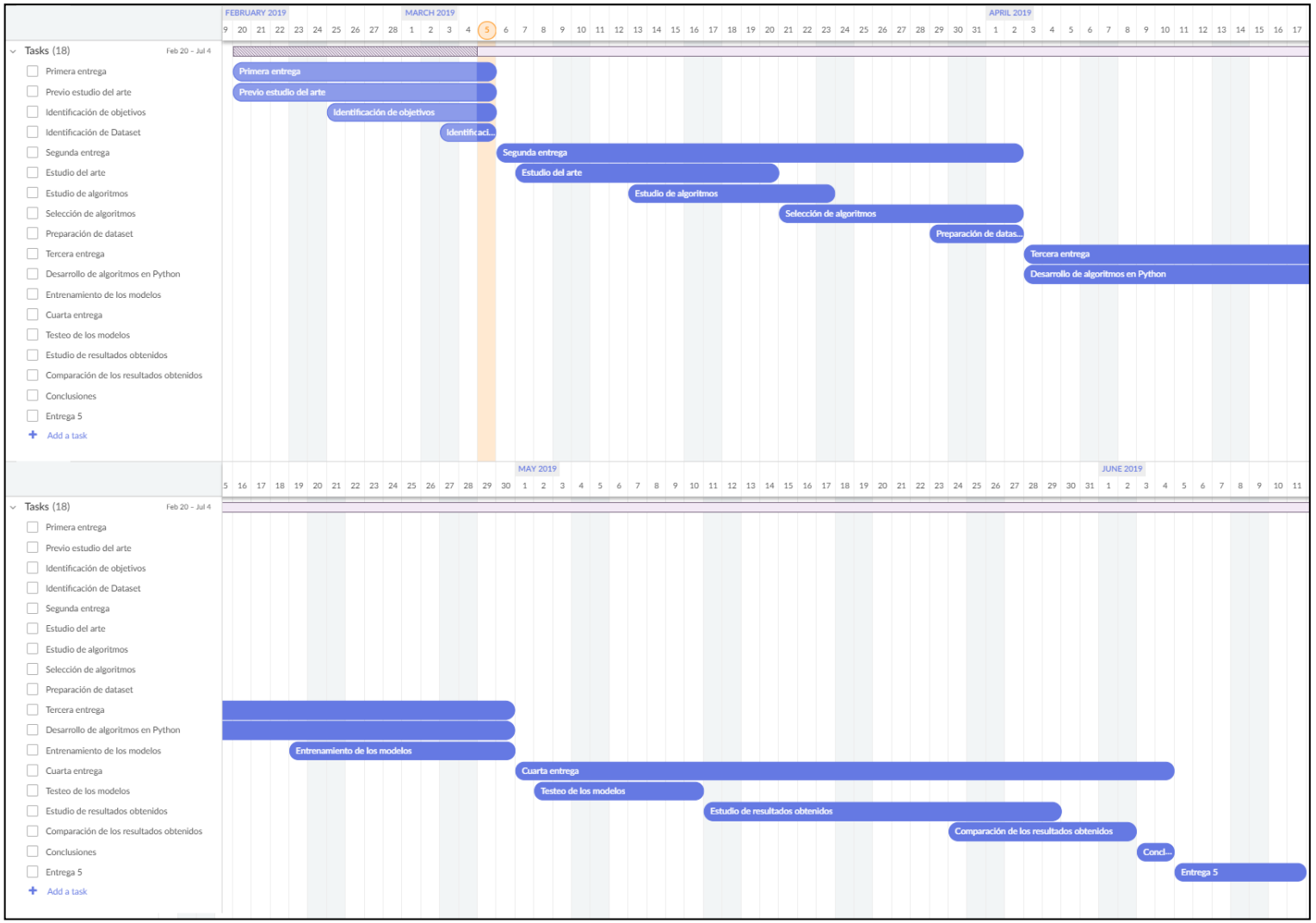

Figura 1 Planificación del trabajo

1.5 Breve descripción de los otros capítulos de la memoria

A continuación, se detallan los capítulos de la presente memoria:

1. **Introducción**. Una introducción del presente Trabajo de Fin de Máster. Detallando los objetivos, la justificación del trabajo, así como una planificación a seguir.
2. **Estado del arte**. Dónde además de introducir al lector en las técnicas de Machine Learning, se realizará un análisis de su uso dentro del ámbito de la ciberseguridad.
3. **Tecnologías y productos utilizados**. Se detallarán los lenguajes, paquetes, aplicaciones y el conjunto de datos, o dataset, que requerirá el sistema.
4. **Análisis del sistema**. Se detallarán las fases del sistema que requerirá el sistema a desarrollar, del proceso de Machine Learning y se explicará el algoritmo que implementará nuestro sistema.
5. **Desarrollo del sistema**. Se pondrá en práctica cada una de las fases descritas anteriormente utilizando el software descrito.
6. **Conclusiones**. Elaboración de las conclusiones obtenidas a lo largo del trabajo.
7. **Bibliografía**. Referencias a material de información consultado durante el presente trabajo.

2. Estado del arte

2.1 Introducción.

Los inicios de Machine Learning (ML) se encuentran en los años 50. Cuando Arthur Samuel escribió el primer programa informático basado en dotar de “inteligencia” a una máquina para el juego de “damas”. Fue posible al estudiar qué movimientos podían hacerse con la victoria.

En los años 60, la creación del algoritmo conocido como “nearest neighbor” permitió a las computadoras utilizar un reconocimiento de patrones, logrando trazar un mapa de una ruta para vendedores ambulantes.

En los años 90, el ML sufrió un gran cambio gracias a la intersección de la informática y la estadística que dio lugar a enfoques probabilísticos en la IA. En este periodo que se comenzó a utilizar esta tecnología en áreas comerciales para la minería de datos, software adaptable y aplicaciones web, aprendizaje de texto y aprendizaje de idiomas.

En los últimos años, las grandes empresas tecnológicas han dejado su huello en este ámbito.

IBM creó un sistema informático de IA capaz de leer y entender el lenguaje natural, con su riqueza de matices y giros lingüísticos, de responder a las preguntas formuladas en pocos segundos y, además, de aprender con cada experiencia, lo que le volvía cada vez más inteligente.

Google ha desarrollado el proyecto GoogleBrain, en el que se desarrolla una red neuronal profunda que puede aprender a descubrir y categorizar objetos.

Facebook consigue crear DeepFace, un algoritmo de software que puede reconocer o verificar individuos en fotos al mismo nivel que los humanos.

El ML se ha convertido en el más popular y exitoso de los sub-campos de la IA, una tendencia guiada por la disponibilidad de mejor hardware y gigantes conjuntos de datos. Se encuentra fuertemente relacionado a la estadística matemática, lidiando con grandes y complejos conjuntos de datos.

2.2 ¿Qué es Machine Learning?

Son varias las definiciones que podemos encontrar sobre Machine Learning (ML). Arthur Samuel, pionero en el campo de la inteligencia artificial, define esta técnica en 1959 como *"el campo de estudio que proporciona a las máquinas la capacidad de aprender sin ser programadas de forma explícita"*. Una definición más detallada sería la de Dan Fagella, CEO de la empresa Emerj: "*Machine Learning es la ciencia que permite que las computadoras aprendan y actúen como lo hacen los humanos, mejorando su aprendizaje a lo largo del tiempo de una forma autónoma, alimentándolas con datos e información en forma de observaciones e interacciones con el mundo real".*

Un sistema de ML es "*entrenado"* en vez de ser explícitamente "*programado"*. En base a una serie de ejemplos o datos relevantes para la tarea en cuestión, encuentra la **estructura estadística o patrones** en dichos ejemplos que eventualmente permiten al sistema aprender las reglas para automatizar dicha tarea.

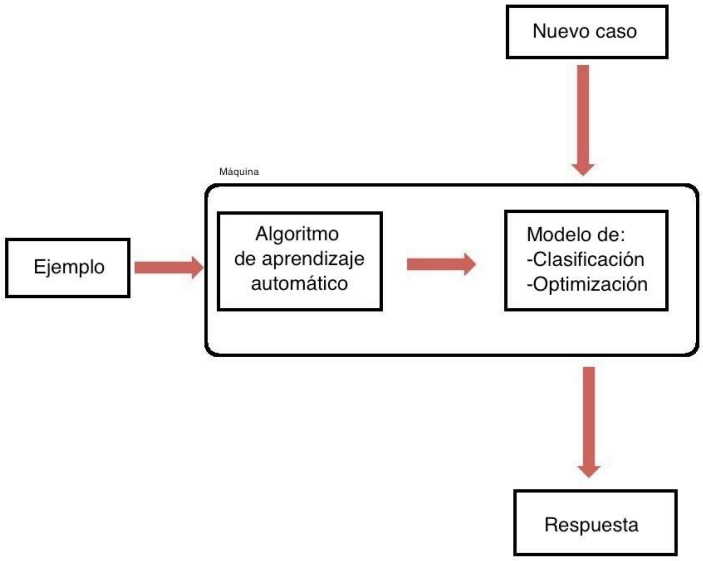

Figura 2 Sistema básico de aprendizaje automático.

Dado que el *ML* un sistema basado en el procesamiento y análisis de datos que son traducidos a hallazgos, se puede aplicar a cualquier campo que cuente con bases de datos lo suficientemente grandes.

Por el momento, algunos de sus usos más populares y desarrollados son:

- Clasificación de secuencias de DNA
- Predicciones económicas y fluctuaciones en el mercado bursátil
- Mapeos y modelados 3D
- Detección de fraudes
- Diagnósticos médicos
- Buscadores en Internet
- Sistemas de reconocimiento de voz.

2.3 Tipos de sistemas de Machine Learning

Existen varias formas de categorizar a los diferentes tipos de sistemas de ML. Podríamos categorizarlos en función de si requieren o no que los modelos sean entrenados por supervisión humana, en función de si pueden o no aprender de forma incremental a medida que el sistema obtenga nuevos datos (batch vs online) o incluso de si se tratan de un aprendizaje basado en instancias o en modelos.
Para clasificarlos, vamos a escoger la primera forma de las descritas anteriormente, es decir, en función de que requieran o no una supervisión humana. Algunos tipos de algoritmos son:

- ***Aprendizaje supervisado***: su objetivo es el de crear una función capaz de predecir el valor correspondiente a unos datos de entrada teniendo un conjunto de datos previamente etiquetado y clasificado. La salida de la función puede ser o bien un valor numérico o una etiqueta de clase.

Algunos de los métodos y algoritmos más utilizados son: Regresión logística, regresión lineal, K vecinos más próximos, redes neuronales y máquinas de vectores de soporte (SVM).

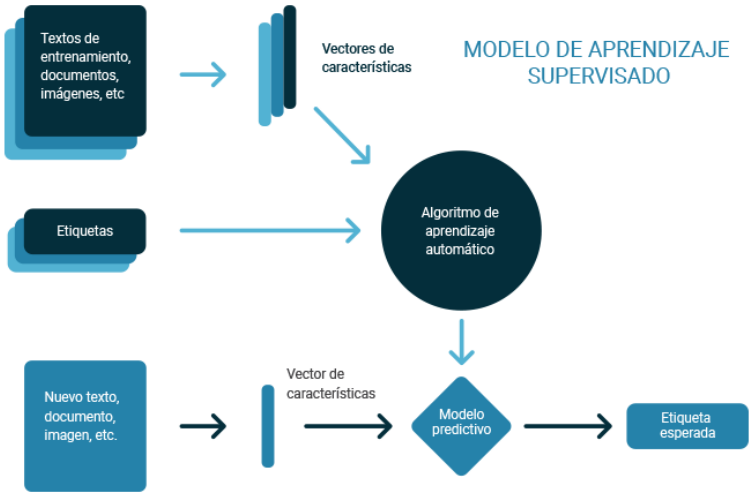

Figura 3 Esquema de modelo de aprendizaje supervisado.

- ***Aprendizaje no supervisado:*** el sistema recibe datos de entrenamiento sin etiquetar e intenta aprender sin un profesor. Los algoritmos que utilizan este tipo de aprendizaje hacen uso de la técnica clustering, que se basa en reunir los datos en grupos que posean características similares entre sí.

  Se utilizan a menudo en sistemas donde se desee detectar anomalías. De forma que se utilicen datos para entrenar al modelo se consideran "normales" y en cuanto se presenta un dato diferente a lo normal, se detecta la anomalía y el sistema alerta de ello.

  Algunos de los métodos y algoritmos más utilizados son: K-medias (K-means), mezcla gaussiana, agrupamiento jerárquico y mapas auto-organizados.

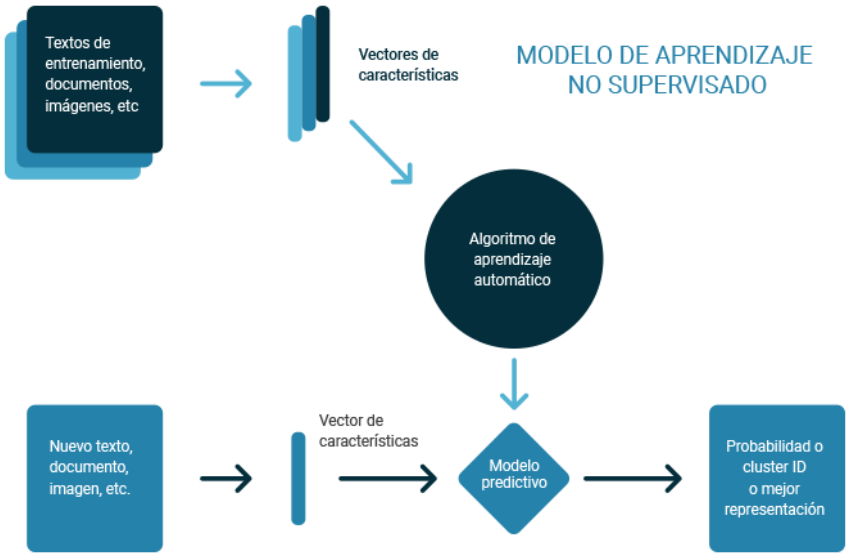

Figura 4 Esquema modelo de aprendizaje no supervisado

- ***Aprendizaje por refuerzo***: el sistema, llamado en este caso *agente*, puede observar el entorno y, mediante la selección de acciones posibles, tomará de decisiones que le harán recibir recompensas positivas o negativas (penalizaciones). El agente debe de aprender cual es la estrategia a seguir en la que exista una mayor recompensa.

  Entre los algoritmos más utilizados podemos nombrar: programación dinámica, Q-learning y SARSA.

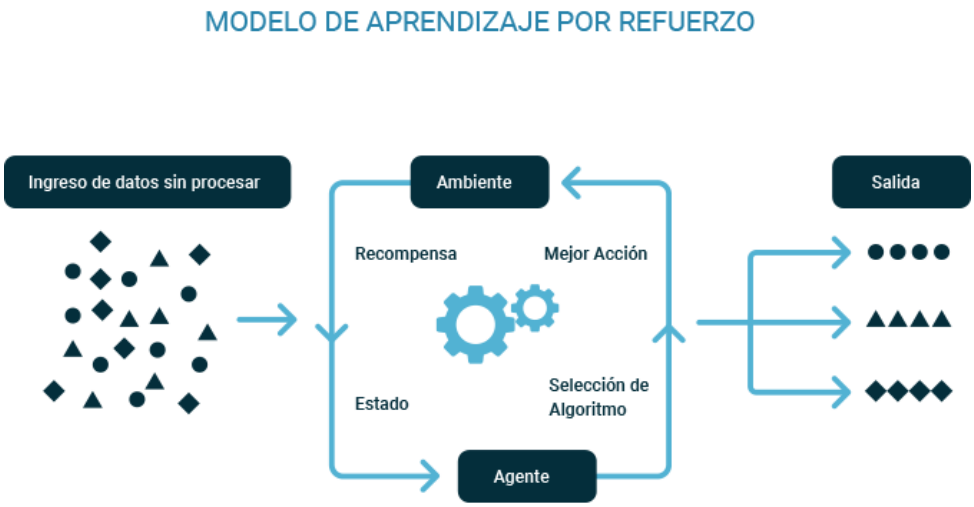

Figura 5 Esquema modelo de aprendizaje por refuerzo

2.4 Aplicaciones en la ciberseguridad y en el uso fraudulento de tarjetas bancarias

La ciberseguridad es uno de los sectores que más está adaptando su tecnología a la inteligencia artificial. Las prácticas de *machine learning* pueden ayudar a las empresas a analizar mejor las amenazas y ser más efectivas a la hora de detener los incidentes de seguridad.

Existen varias aplicaciones de la IA dentro del ámbito, como la detección y la clasificación del malware, en sistemas de detección de intrusos (IDS), dónde se analizan los datos para detectar signos de posibles amenazas

Además, estas tecnologías destacan por permitir automatizar determinadas tareas para las que hasta ahora era necesario desplegar equipos de seguridad altamente capacitados.

Un estudio de Webroot [9], revela que un 71% de empresas planean usar más inteligencia artificial y aprendizaje automático en sus herramientas de ciberseguridad este año, aunque más de la mitad (58%) no están seguras de lo que realmente hace esa tecnología.

Dentro del ámbito financiero, la IA se podría aplicar en la detección de fraudes. Compañías como Visa han estado usando y mejorando su tecnología para la detección del fraude durante los últimos años, enfocándose en el aprendizaje profundo (Deep Learning) y modelos de aprendizaje automático escalables, pudiendo variar el rango de datos utilizados para determinar una conclusión basada en diferentes contextos. También se centran en soluciones de análisis predictivo en tiempo real.

3. Tecnologías y productos utilizados

A continuación, se detallan todas las herramientas, tales como lenguajes, librerías, aplicaciones y datos, que han sido utilizados a lo largo del desarrollo del producto:

3.1 Lenguaje de programación

El lenguaje utilizado durante el desarrollo del producto ha sido *Python*. Es un lenguaje de programación interpretado y de alto nivel. Soporta orientación a objetos, programación imperativa y programación funcional.

Python es utilizado comúnmente por científicos de datos y desarrolladores para la construcción y análisis de modelos.

Cuenta con multitud de librerías y de frameworks para realizar operaciones que podríamos requerir en un proceso de ML, como abrir archivos, serializar datos, realizar cálculos matemáticos, analizar datos, gráficas, etc.

Su código es conciso y legible, lo que facilita su uso y comprensión. Es una ventaja a destacar ya que, tanto el ML como Deep Learning se basa en algoritmos extremadamente complejos y flujos de trabajo de múltiples etapas, por lo que cuanto menos tenga el desarrollador que centrarse en la comprensión del código, mejor.

Además, es un lenguaje de programación de código abierto y está respaldado por una gran cantidad de recursos y documentación de alta calidad.

Figura 6 Logo Python

Un estudio con datos del portal de empleo Indeed [13], reveló que Python es el lenguaje de programación para ML más utilizado.

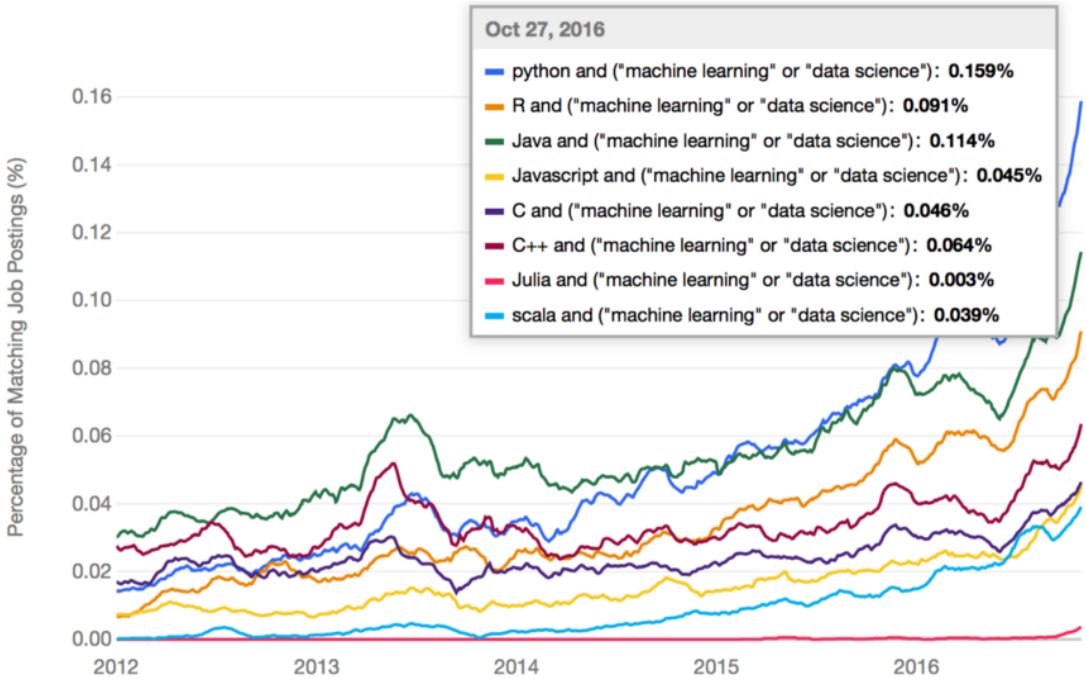

Figura 7 Porcentaje de uso de lenguajes de programación en ML

3.2 Librerías

A continuación, se detallan las librerías que han sido utilizadas:

- **Pandas**: es una librería destinada al análisis de datos, que proporciona unas estructuras de datos flexibles y que permiten trabajar con ellos de forma muy eficiente.

Figura 8 Logo Pandas

- **Sckit-learn**: es una librería que incluye varios algoritmos de clasificación, regresión y análisis de grupos. Está destinada para interoperar con las bibliotecas numéricas y científicas NumPy y SciPy.

Figura 9 Logo Sckit Learn

- **Numpy**: incluye una serie de funciones matemáticas de alto nivel para operar con vectores o matrices.

Figura 10 Logo Numpy

- **Matplotlib**: es una librería utilizada para la generación de gráficos a partir de datos contenidos en listas o arrays.

Figura 11 Logo Matplotlib

3.3 Aplicaciones

- **Jupyter Notebook**: es una aplicación web de código abierto, desarrollada utilizando lenguaje HTML agnóstico que permite crear, compartir y editar documentos en los que se puede ejecutar código Python.

Figura 12 Logo Jupyter

- **Anaconda**: es una Suite de código abierto que abarca una serie de aplicaciones, librerías y conceptos diseñados para el desarrollo de la Ciencia de datos con Python.

  En líneas generales es una distribución de Python que funciona como un gestor de entorno, un gestor de paquetes y que posee una colección de más de 720 paquetes de código abierto.

Figura 13 Logo Anaconda

3.4 Datos

El conjunto de datos, en adelante dataset, con el que trabajaremos consistirá en una recopilación de transacciones realizadas con tarjetas de crédito en septiembre de 2013. Recoge las transacciones que se realizaron durante dos meses, en las qué de 284.807 transacciones, 492 fueron fraudulentas.

Únicamente contiene variables de entrada numéricas, que son el resultado de una transformación mediante la técnica del análisis de componentes principales (ACP). Haciendo uso de esta técnica se reduce el número de variables, resaltando las de mayor información.

Debido a la confidencialidad de los datos, no se proporciona información acerca de algunas de las variables. En su lugar, vemos como se nombra a estas variables *V1*, *V2*, …, *V28*, a excepción de las variables *time*, *amount* y *class.*

La variable *time* contiene los segundos transcurridos de la transacción entre la transacción en concreto y la primera. La variable *amount* se corresponde con la cantidad en Euros de la transacción en concreto. Y, por último, la variable *class* toma el valor 1 en caso de fraude y 0 en caso contrario.

4. Análisis del sistema

En este capítulo se analizará el sistema a desarrollar, haciendo hincapié en las fases del sistema o de un proceso de ML, los documentos Python en los que se dividirá la solución y la selección del algoritmo que utilizará el sistema.

4.1 Fases del sistema

Las diferentes fases del proceso de ML que se realizarán son:

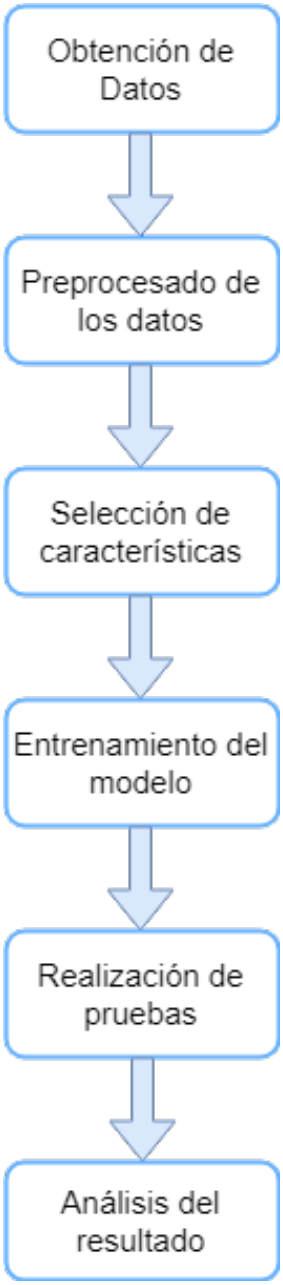

Figura 14 Fases de un sistema de Machine Learning

I. **Obtención de datos.** Es el primero de los procesos. El dataset detallado en el apartado *3.4 Datos* podemos obtenerlo en *Kaggle*, comunidad enfocada al aprendizaje a la ciencia de datos. Podemos descargar el dataset en el siguiente enlace: *https://www.kaggle.com/mlg-ulb/creditcardfraud/*.

II. **Preprocesado de los datos.** Durante esta etapa se deben de preparar los datos con el fin de que el proceso sea más efectivo y menos costoso en recursos y tiempo.

    Normalmente, los datos los obtenemos en formatos no óptimos, incluso imposibles de tratar. Estudiaremos si podría ser viable utilizar técnicas como el *Data cleaning* o limpieza de datos, eliminando datos con ruido o incorrectos; transformar los datos, mediante normalización o el uso de ACP o requerir a la técnica denominada como *Data reduction*, que consiste en reducir el tamaño de los datos.

III. **Selección de características**. Es un proceso que normalmente se aplica para la construcción de conjuntos de datos de entrenamiento para tareas de modelado predictivo.

    El objetivo es seleccionar un subconjunto de las características a partir del conjunto de datos original que reduce sus dimensiones al usar un conjunto de características mínimo para que represente la cantidad máxima de varianza en los datos. Este subconjunto de características se utiliza para entrenar el modelo.

    La selección de características aumenta aumenta la precisión de la clasificación a través de la eliminación de características irrelevantes, redundantes o altamente correlacionadas.

Además, al disminuir la cantidad de características, hace que el proceso de entrenamiento del modelo sea más eficiente. Esto resulta especialmente importante cuando se trata de sistemas de aprendizaje que son costosos de entrenar.

IV. **Entrenamiento del modelo.** Es el proceso en el que se detectan los patrones de un conjunto de datos. Una vez identificados los patrones, se pueden hacer predicciones con nuevos datos que se incorporen al sistema.

V. **Realización de pruebas.** Después de entrenar al modelo, debemos de realizar pruebas utilizando transacciones fraudulentas y no fraudulentas. De forma que verifiquemos que efectivamente, el modelo es capaz de detectar la anomalía, es decir, el fraude.

VI. **Análisis del resultado.** Se realizará una descripción y evaluación de los resultados obtenidos. Analizando los tiempos de lectura y tratado de los datos, además de la precisión de acierto obtenida. Si es lo suficientemente elevada y se consigue en tiempos óptimos, habremos dado con un buen modelo.

Realmente, estas tres últimas fases (Entrenamiento del modelo, realización de pruebas y análisis del resultado), se realizan de forma cíclica. Si obtenemos unos resultados no deseados, debemos de entrenar de nuevo el modelo, ajustando sus características para obtener resultados diferentes. Es realmente complicado obtener unos buenos resultados al primer intento, por lo que se vuelve indispensable analizar los resultados y volver a ajustar el modelo.

4.2 Algoritmo del sistema

A la hora de seleccionar un algoritmo se debe de tener en cuenta ciertos factores, como pueden ser el tamaño y el tipo de los datos con los que se trabaja, la información que se desea obtener de los datos y cómo se empleará dicha información.

El proyecto sckit propuso una forma de seleccionar un algoritmo basado en las características anteriormente mencionadas:

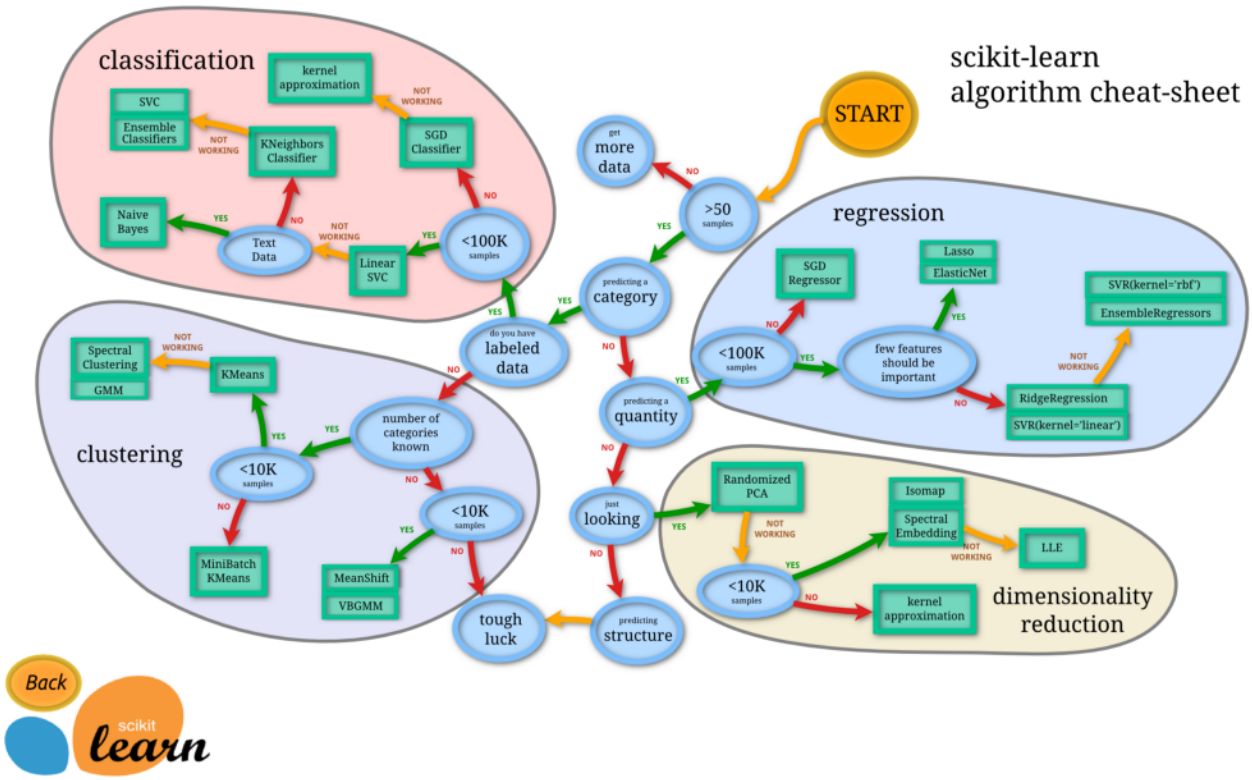

Figura 15 Selección de algoritmos propuesto por Sckit-Learn

Sin embargo, realmente no existe ningún método mejor que otro ni ninguno que se pueda aplicar de forma universal. Es por ello por lo que se utiliza el ensayo y error probando diferentes algoritmos hasta dar con el más acorde a lo que busquemos.

Finalmente, el algoritmo que utilizará el sistema será Isolation Forest (IF). En casos en los que no se disponga de un conocimiento de las variables, debido a la confidencialidad de los datos, son útiles los algoritmos no supervisados. Este algoritmo es comúnmente utilizado para la detección de anomalías, por lo que nos es útil para nuestro caso.

El proceso de aislar instancias anómalas en este algoritmo se realiza mediante un conjunto de Árboles de Decisión Binarios. Se seleccionan características de modo aleatorio y eligiendo también aleatoriamente valores de las variables (entre el valor mínimo de la variable y el máximo) para particionar el espacio de búsqueda y generar ramas del árbol de decisión.

De este modo, se categorizan como anomalías aquellas instancias que tienen menor profundidad en el árbol, es decir, aquellas cuya longitud de la hoja a la raíz sea la mínima, puesto que se asume que las anomalías son más fáciles de aislar mediante el particionamiento recursivo del dataset.

A continuación, se muestra un ejemplo de la representación de este algoritmo:

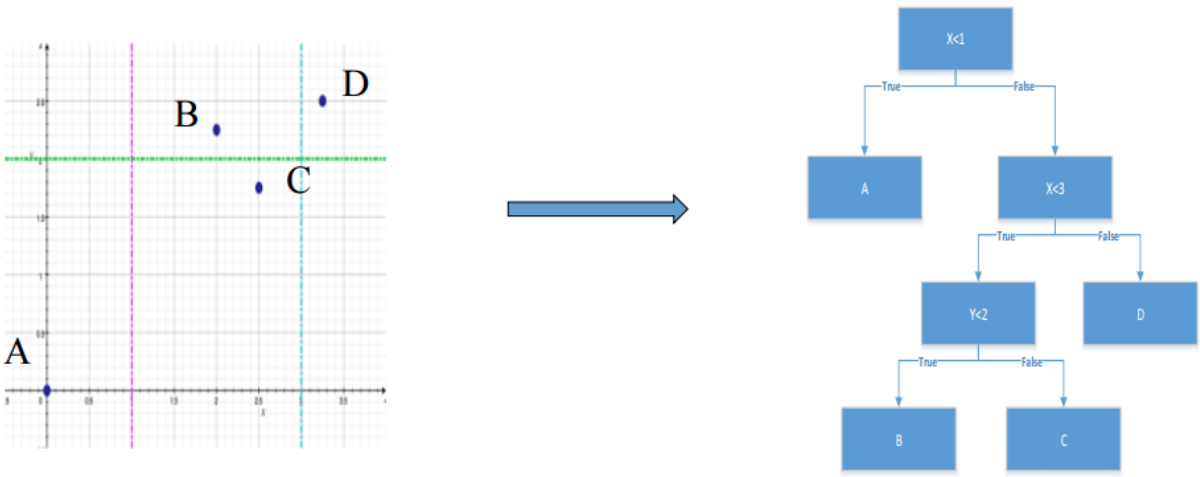

Figura 16 Ejemplo de representación IF

5. Desarrollo del sistema.

En este apartado se explicará el desarrollo de la solución obtenida. En primer lugar, se detallará la configuración del entorno, más adelante se procederá a explicar la solución para cada una de las fases de un proceso de Machine Learning, descritas anteriormente en el apartado *4.1 Fases del sistema.*

Las fases anteriormente citadas se encontrarán desarrolladas en diferentes Notebooks de Jupyter.

Así, en esta solución tendremos un fichero en donde leeremos los datos de un .csv (formato del Dataset), los estudiaremos, los preprocesaremos y guardaremos una serie de datos del dataset, necesarios para entrenar y predecir, en formato de matrices.

En un segundo fichero recuperaremos estos datos en formato de matrices, entrenaremos al modelo, que sigue un algoritmo de IsolationForest y predeciremos.

El tercer fichero será parecido al anterior, con la diferencia de que el algoritmo que implementará nuestro modelo será RandomForest.

De esta forma, tendremos la solución más “limpia”, reducimos código y nos abstraemos de varias funcionalidades, como la obtención y preprocesamiento de los datos, cada vez que queramos entrenar y probar modelos.

5.1 Configuración del entorno.

En primer lugar, vamos a proceder a instalar Python, además de todas las librerías necesarias y anteriormente citadas en el apartado *3.2 Librerías.* Para ello, se ha descargado e instalado Anaconda.

Consiste en una distribución libre y abierta utilizada en ciencias de datos y machine Learning. Hará de gestor de entorno y gestor de paquetes, posee una colección de más de 720 paquetes de código abierto. Podemos descargar esta distribución de la página oficial https://www.anaconda.com/distribution/. Se ha seleccionado la versión de 3.7 de Python.

Anaconda se agrupa en 4 sectores o soluciones tecnológicas, Anaconda Navigator, Anaconda Project, Las librerías de Ciencia de datos y Conda. Todas estás se instalan de manera automática y en un procedimiento muy sencillo.

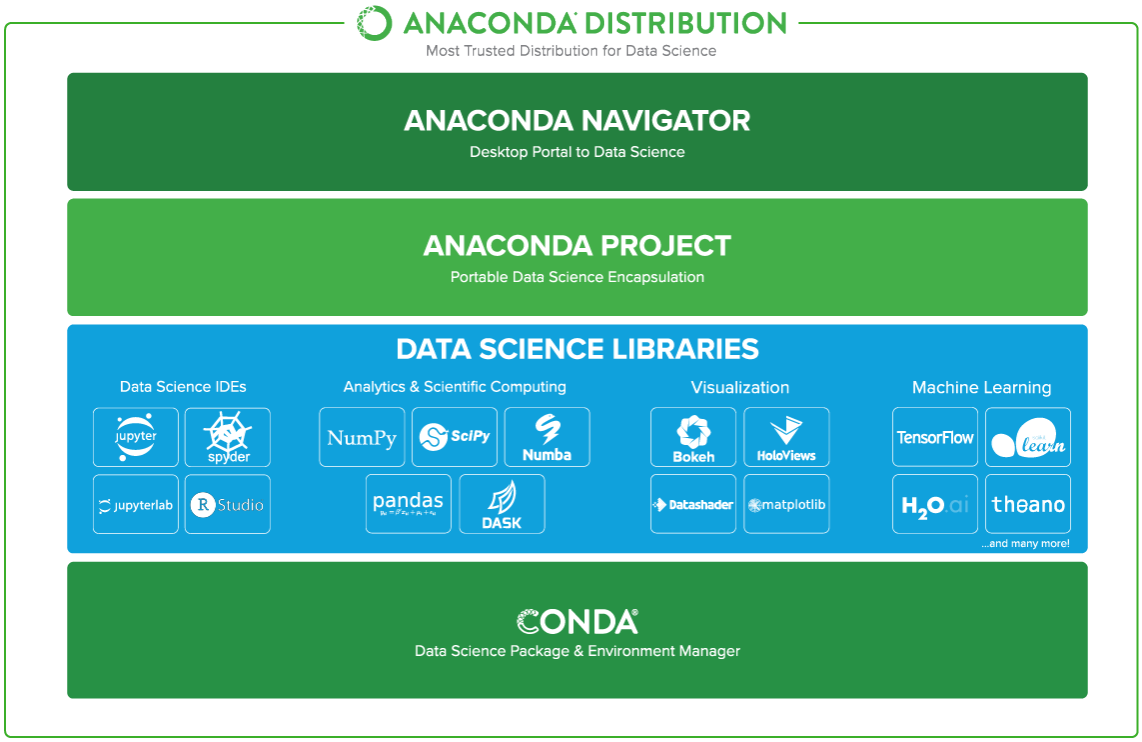

Figura 17 Distribución Anaconda

Cuando instalamos Anaconda, tendremos disponibles todas estas herramientas ya configuradas, podemos gestionarla mediante la interfaz gráfica de usuario Navigator o podemos utilizar Conda para la administración mediante la consola. Además, se podrá instalar, eliminar o actualizar cualquier paquete Anaconda en la interfaz gráfica Navigator o bien, mediante un solo comando de Conda.

A continuación, se muestra la interfaz gráfica de Anaconda, Anaconda Navigator.

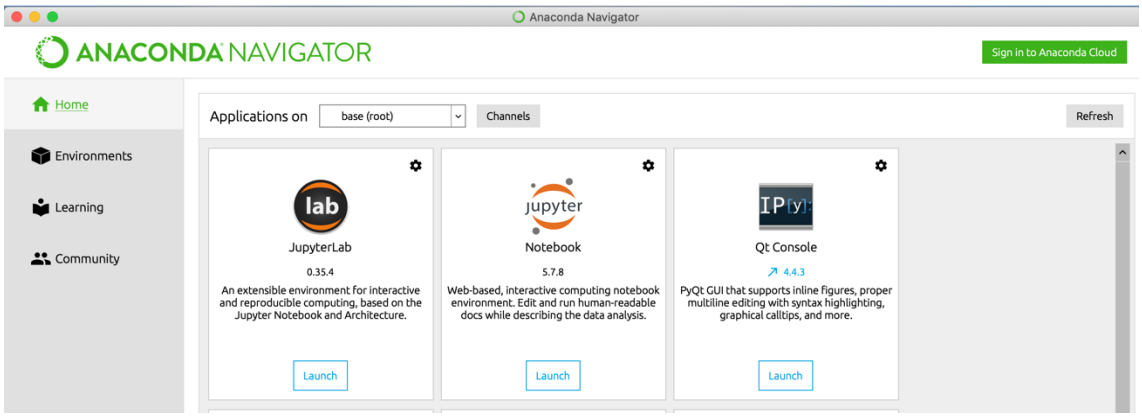

Figura 18 Anaconda Navigator

A través de Anaconda Navigator, se lanzará un Jupyter Notebook. Consiste en un entorno de trabajo que permite desarrollar código en Python de manera dinámica.

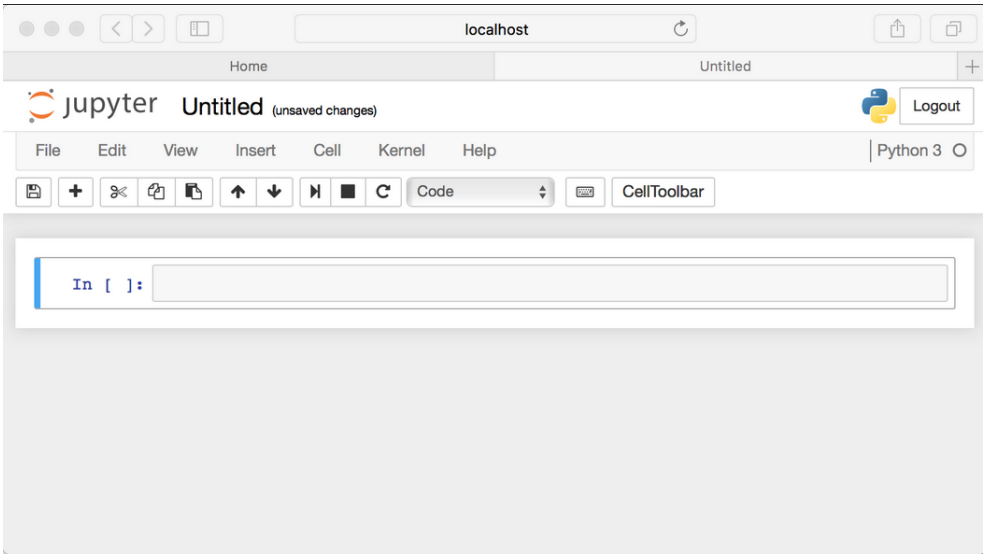

Figura 19 Ejemplo de Jupyter Notebook

Por último, debemos de importar en el notebook creado, los módulos y las clases a utilizar. Podemos hacerlo mediante las siguientes instrucciones:

import <*nombre módulo*> **as** <*nombre al que nos referiremos en el código*>
from <*nombre módulo*> **import** <*nombre de la clase*>

Así, un ejemplo de las importaciones del fichero de preprocesamiento de los datos sería el siguiente:

```
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.preprocessing import normalize
import pandas as pd
import numpy as np
```

Figura 20 Ejemplo importación de librerías Python

5.2 Obtención de los datos.

El Dataset consiste en un conjunto de transacciones realizadas con tarjetas de crédito en septiembre de 2013. Recoge las transacciones que se realizaron durante dos meses, en las qué de 284.807 transacciones, 492 fueron fraudulentas.

Fue recogido por el grupo *Worline and the Machine Learning* de la Universidad Libre de Bruselas con el objetivo de realizar estudios en la minería de datos en Big Data y en la detección de fraude.

Podemos obtenerlo en *Kaggle*, comunidad enfocada al aprendizaje a la ciencia de datos. Podemos descargar el dataset en el siguiente enlace: *https://www.kaggle.com/mlg-ulb/creditcardfraud/*.

A continuación, se detalla la información de los datos que aparecen recopilados en el dataset:

Nº columna	Nombre campo	Explicación
1	Time	Segundos transcurridos de la transacción entre la transacción en concreto y la primera.
2-29	V1", "", …, "V28"	Debido a la confidencialidad de los datos, no se proporciona información acerca de estas variables.
30	Amount	Cantidad en Euros de la transacción en concreto.
31	Class	Toma el valor 1 en caso de fraude y 0 en caso contrario.

Figura 21 Tabla Datos del Dataset

5.3 Preprocesado de los datos.

Como se explicó en el inicio del presente apartado, el primero de los ficheros Python consiste en obtener los datos del fichero, preprocesarlos y almacenar los datos en formato de matrices en ficheros que, posteriormente, serán leídos cuando queramos entrenar al modelo y predecir.

A continuación, se muestra el código de este fichero:

```
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.preprocessing import normalize
import pandas as pd
import numpy as np

#[1] Importamos los datos del csv
data = pd.read_csv("Datasets/creditcard.csv")

#[2] Convertimos los datos en formato Numpy
data_matrix = data.values
X = data_matrix[:, 1:29]
y = data_matrix[:, 30]

#[3] Normalizamos
X = normalize(X, norm="l1")

#[4] Dividimos los datos en conjuntos de entrenamiento y de predicción
stratSplit = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=32)
for train_index, test_index in stratSplit.split(X, y):
    X_train = X[train_index]
    X_test = X[test_index]
    y_train = y[train_index]
    y_test = y[test_index]

#[5] Guardamos las matrices
np.save("Datasets/creditcard.X_train", X_train)
np.save("Datasets/creditcard.X_test",  X_test)

#[6] Guardamos los resultados almacenados en el dataset.
np.save("Datasets/creditcard.y_train", y_train)
np.save("Datasets/creditcard.y_test", y_test)
```

Figura 22 Código Python. Fichero I. Preparación

Al principio del documento, importamos los módulos necesarios. Para hacer a nuestro código más “limpio”, siempre que utilicemos Pandas y Numpy lo haremos escribiendo *pd* y *np*, respectivamente.

En [1], leemos el Dataset, que se encuentra en formato .csv, y almacenamos los datos leídos en la variable *data.* Lo hacemos mediante el método *read_csv*

que nos proporciona Pandas. Es un método sobrecargado al que se le puede pasar multitud de parámetros, pero únicamente le pasaremos como parámetro la ruta del fichero en donde se encuentra ubicado el Dataset. Nos devolverá una estructura de datos bidimensional.

En [2], se crea una variable denominada *data_matrix*. En ella almacenaremos la estructura bidimensional retornada por el método *read_csv*, podemos recuperar los datos mediante la propiedad *values.* En la variable X guardamos los datos que se correspondan con todas las filas y de la columna 2 a 30. En la variable Y guardamos los datos que se correspondan con todas las filas y de la columna 31. De esta forma, tendremos en la variable X los datos time, *v1,v2,…v28* y *amount*, mientras que en la variable Y tendremos los datos de la columna *class*, es decir, el "resultado".

Antes de dividir el conjunto de datos, para poder entrenar al modelo y predecir, debemos de procesar los datos. Como se mencionó en el punto *5.2 Obtención de los datos*, el Dataset se realizó con el objetivo de realizar estudios en la minería de datos en Big Data y en la detección de fraude. Son datos que previamente han sido procesados. No existen columnas o filas vacías, los datos no se encuentran en formatos que no pueden ser tratados, no existe ruido. Además, como se indica en la página donde se obtuvo el Dataset, se ha realizado un Análisis de Componentes Principales, en adelante ACP (PCA en inglés).

ACP es una técnica comúnmente utilizada para reducir las dimensiones de un conjunto de datos, ya que un número elevado e innecesario de columnas de un Dataset puede llegar a ser un problema. Busca la estructuración de un conjunto de datos multivariado mediante la reducción del número de variables originales a un conjunto más pequeño, las cuales son combinaciones lineales de aquellas que recogen la mayor parte de la variabilidad del conjunto inicial de variables. Para que se pueda realizar el PCA, es necesario que estén muy correlacionadas entre sí.

Además de aplicar ACP, en [3] vamos a normalizar los datos. La normalización de los datos consiste en comprimir o extender los valores de la variable para que estén en un rango definido. Podemos hacerlo con el método *normalize* (param1, param2), en donde el primer parámetro será la variable a normalizar y la segunda la técnica de normalización a utilizar. L1 hace referencia a la técnica llamada Mínimas desviaciones absolutas (LAD por sus siglas en inglés).

La técnica intenta encontrar una función que se aproxima mucho a un conjunto de datos. El método minimiza la suma de errores absolutos (SAE), la suma de los valores absolutos de los "residuos" verticales entre puntos generados por la función y los puntos correspondientes en los datos.

A continuación, dividiremos los datos en [4]. Este proceso consistirá en dividir los datos ya procesados en dos conjuntos, uno de entrenamiento (*train*) y otro para realizar predicciones o pruebas (test).

El de entrenamiento contendrá un 80% de los datos totales y el de pruebas contendrá un 20%. Es importante que no se prueben los resultados con unos datos que sean los mismo que los de entrenamiento ya que, aunque nos proporcione una precisión del 100% de acierto en las pruebas, no tendríamos la certeza de que, efectivamente, el modelo es lo suficientemente bueno. Podríamos tener nuevos datos para realizar pruebas y que esta vez nos de un resultado mucho menor. Es por ello por lo que se realiza esta división en dos partes.

Esta división podemos hacerla mediante el método *StratifiedShuffleSplit*, al que le pasaremos tres parámetros. El primero de ellos*, n_splits*, es el número de “cortes” que sufrirá el conjunto de datos. Lo igualamos a uno para poder tener dos partes. El segundo, *test_size*, es un valor comprendido entre el 0.0 y el 1, e indica la proporción del tamaño del conjunto de pruebas, en este caso será 0.2, es decir, el tamaño del conjunto de pruebas será de un 20%. Por lo que el de entrenamiento será de un 80% (este valor se añade automáticamente al definir la proporción del conjunto de pruebas). Y el tercer parámetro, *Random_state*.

Lo podemos igualar a un valor entero cualquiera para que los datos se generan de forma aleatoria. El valor igualado a este parámetro actúa de semilla para generar el orden aleatorio.

Una vez definido el método de división de los datos, que será guardado en la variable *stratSplit*, vamos a utilizarlo mediante el método *Split* y pasándole como parámetros los datos X y los resultados Y (acorde a cada dato de X). De forma que, mediante un bucle for, se almacenen los datos en las variables x_*train*, x_*test*, y_*train*, e y_*test*.

A continuación, en [5] y en [6], vamos a guardar cada uno de estos datos en ficheros con formato numpy. Podemos hacerlo mediante el método *save* de Numpy, contendrá dos parámetros. El primero de ellos será la ubicación y el nombre del fichero, y el segundo la variable a guardar en ese fichero, que tendrá que ser una estructura de datos en formato numpy.

Como podemos ver en la siguiente imagen, se crean los siguientes ficheros. El tamaño que ocupa cada uno de ellos es acorde a la división que teníamos pensado realizar:

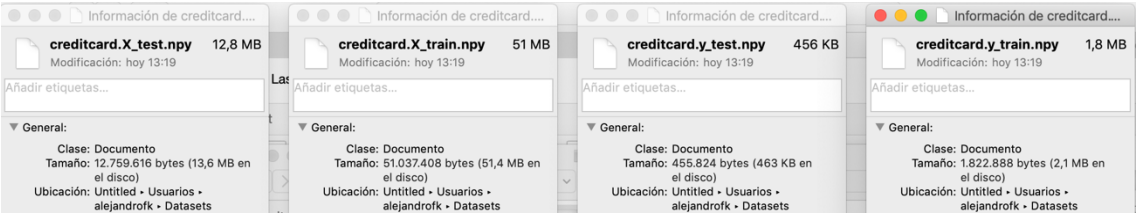

Figura 23 Ficheros datos entrenamiento y test

Ahora ya podremos utilizar estos datos para realizar el estudio del modelo y pruebas, en vez del Dataset directamente.

5.4 Estudio de los datos.

Para realizar el estudio de los datos, se realizará un histograma para cada una de las variables, una matriz de correlación entre todas las variables y la obtención de una serie de datos como la media, cuartiles, varianza, valores mínimos y máximos, etc.

Un histograma nos puede resultar de bastante utilidad para comprender la frecuencia con la que se obtiene un valor de una determinada variable. Nos puede resultar útil para comprobar la veracidad de los datos, por ejemplo, no tendría sentido que para la variable *Class* se obtuviesen valore s diferentes a 0 y 1, o que la variable Time no tuviese ningún valor comprendidos entre su mínimo y máximo valor.

A continuación, se muestran los histogramas para cada una de las variables:

```
import matplotlib.pyplot as plt
data.hist(bins=50, figsize=(20,15))
plt.show
```

```
<function matplotlib.pyplot.show(*args, **kw)>
```

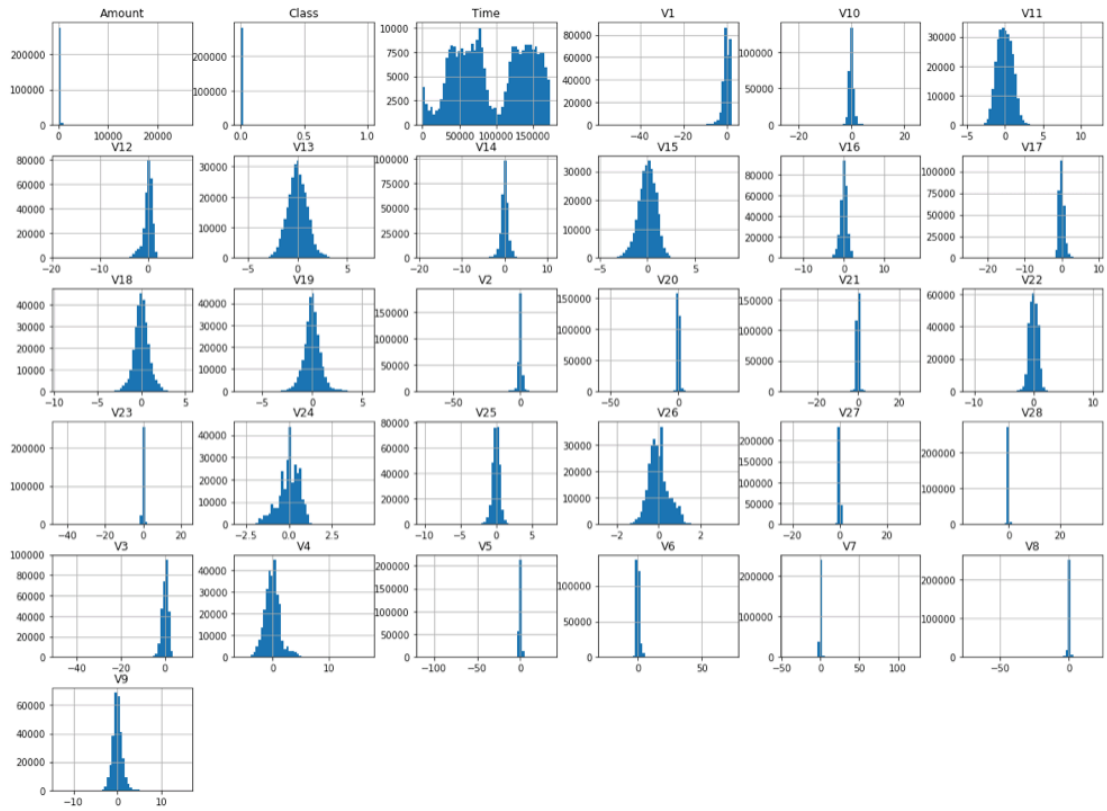

Figura 24 Histogramas de variables

Podemos obtener también una serie de datos como la media, la varianza, mínimos, máximos, etc. Y estudiar si tiene sentido los valores que obtenemos. Por ejemplo, no tendría sentido obtener un máximo superior a 1 de la variable *Class* o un valor negativo para Time.

```
data.describe()
```

	Time	V1	V2	V3	V4	V5	V28	Amount	Class
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	284807.000000	284807.000000
mean	94813.859575	3.919560e-15	5.688174e-16	-8.769071e-15	2.782312e-15	-1.552563e-15	-1.206049e-16	88.349619	0.001727
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	3.300833e-01	250.120109	0.041527
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-1.543008e+01	0.000000	0.000000
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-5.295979e-02	5.600000	0.000000
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	1.124383e-02	22.000000	0.000000
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	7.827995e-02	77.165000	0.000000
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	3.384781e+01	25691.160000	1.000000

Figura 25 Datos estadísticos del Dataset

También podemos realizar una matriz de correlación. De forma que podamos analizar si existe una relación entre dos variables. Nos puede resultar útil especialmente para el caso de las variables *v1* a *v28*, ya que no sabemos realmente que tipo de información almacena.

```
import seaborn as sns

correlation_matrix = data.corr()
fig = plt.figure(figsize=(9,7))
sns.heatmap(correlation_matrix,vmax=0.8)
plt.show()
```

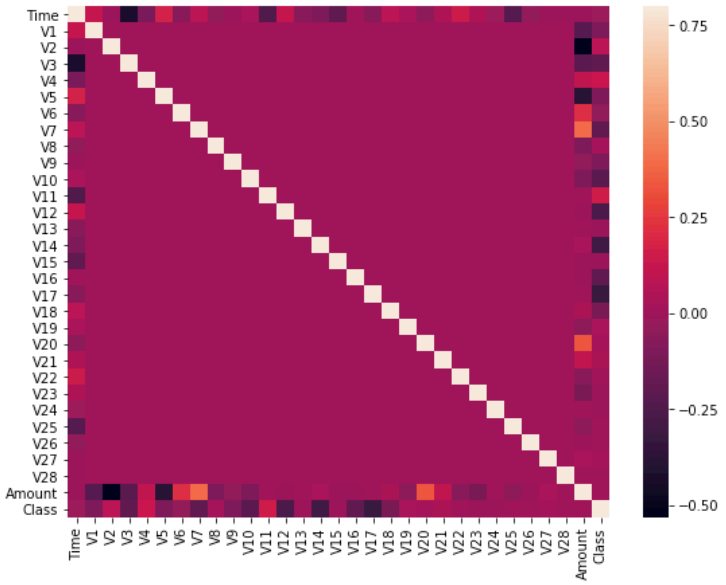

Figura 26 Matriz de correlación de variables v1 a v28

Vemos que no existe ninguna relación entre ellas.

5.5 Estudio del modelo y realización de pruebas

El estudio del modelo y la realización de pruebas se hace conjuntamente debido a que al realizar pruebas y no obtener los resultados deseados debemos de estudiar el modelo de nuevo. Y repetir este proceso hasta obtener los resultados deseados.

Antes de realizar el modelo, se hará un estudio previo de los algoritmos que se utilizan, Random Forest (RF) e Isolation Forest (IF). RF hace uso de árboles de decisión, mientras que IF hace uso del algoritmo isolation tree.

Este apartado si encuentra dividido en dos subapartados, en cada uno de ellos se explican estos algoritmos, además de realizar el modelo en Python para cada uno de ellos.

5.5.1 Random Forest

Este algoritmo hace uso de la técnica Agregación Bootstrap, también conocida como Bagging. Consiste en combinar modelos a partir de una familia inicial, provocando una disminución de la varianza y evitando el sobreajuste.

De forma más precisa, esta técnica consiste en dado un conjunto de entrenamiento D, de tamaño n, se generan m nuevos conjuntos de entrenamiento, Di, de tamaño n', tomando al azar elementos de D de manera uniforme y con reemplazo. Por tanto, algunos elementos del conjunto original pueden aparecer repetidos en los nuevos conjuntos generados. A partir de estos m nuevos conjuntos de entrenamiento se construyen m nuevos modelos de aprendizaje, y la respuesta final de la combinación se consigue por medio de votación de las m respuestas (en caso de buscar una clasificación) o por la media de ellas (en caso de buscar una regresión).

Los modelos de los que se hacen uso en esta técnica son árboles de decisiones. Podríamos decir, por tanto, que RF consiste en un conjunto de árboles de decisiones, y que, mediante la suma de “votos” o decisiones

tomadas por cada uno de los árboles llegamos a una conclusión, al resultado que nos proporciona RF.

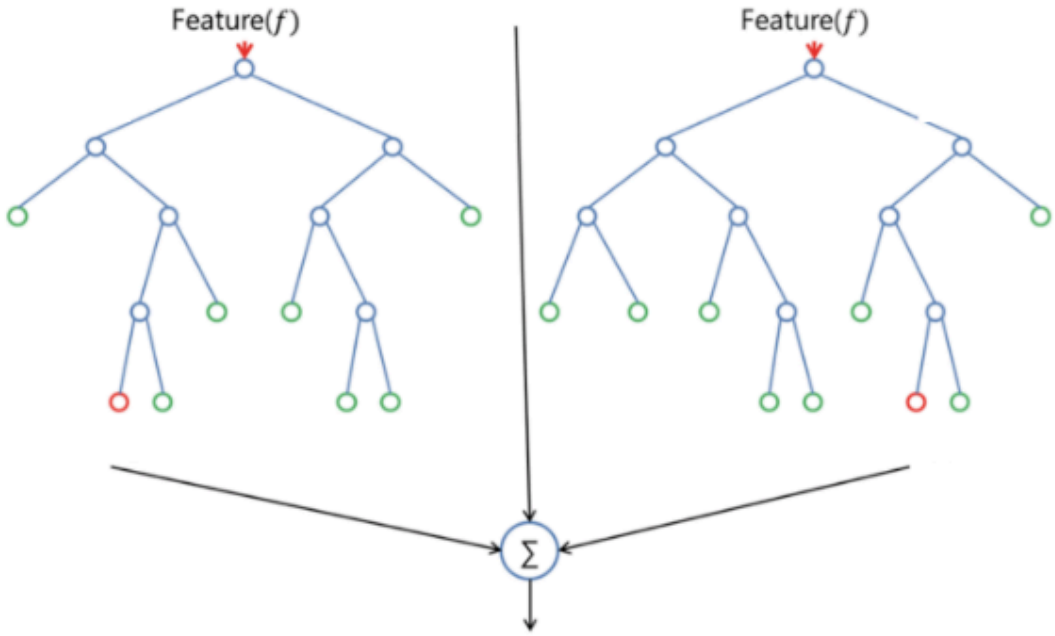

Figura 27 Ejemplo gráfico de Random Forest

Los árboles de decisión proporcionan un conjunto de reglas que se van aplicando sobre los nuevos datos para decidir qué clasificación es la más adecuada a sus atributos.

Está formado por un conjunto de nodos de decisión y de hojas, nodo ubicado en el nivel más profundo del árbol.

Un nodo de decisión está asociado a uno de los atributos y tiene 2 o más ramas que salen de él, cada una de ellas representando los posibles valores que puede tomar el atributo asociado. En el nodo de decisión se realiza una cuestión con dos posibles respuestas, y dependiendo de la respuesta que se obtenga, el flujo tomará una de las dos ramas salientes.

Una hoja está asociada a la clasificación que se quiere proporcionar, y devuelve la decisión del árbol con respecto a los datos de entrada.

A continuación, se muestra un ejemplo sencillo de la estructura de un árbol de decisión (derecha) y un gráfico en donde viene representadas las decisiones tomadas para según qué valores.

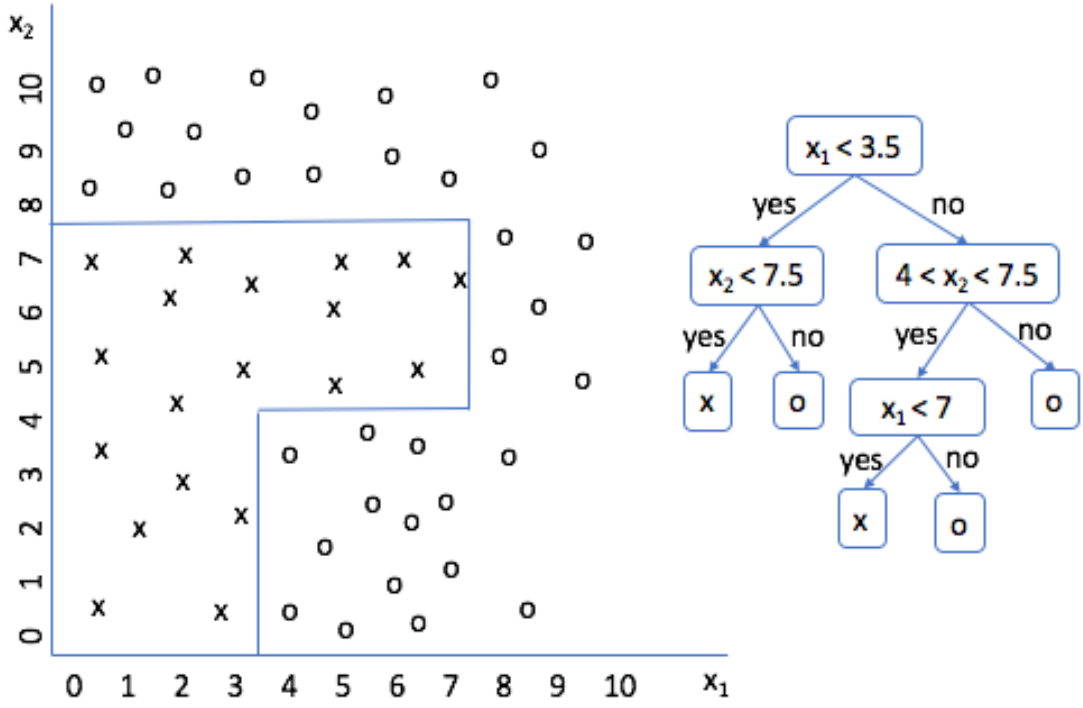

Figura 28 Ejemplo algoritmo Random Forest

Aunque el algoritmo sea capaz de tomar una decisión, esta decisión cuenta con un porcentaje de error, llamado impureza de Gini. Este valor nos mide cuán a menudo un elemento elegido aleatoriamente del conjunto sería etiquetado incorrectamente. La impureza de Gini se puede calcular sumando la probabilidad de cada elemento siendo elegido multiplicado por la probabilidad de un error en la categorización de ese elemento.

$$Gini(t) = 1 - \sum_{c \in C} p_{t,c}^2$$

Figura 29 Fórmula impureza de Gini

Donde C es el conjunto de clases y P t,c es la probabilidad de ocurrencia de la clase c en el noto t.

Con la librería scikit-learn, podemos visualizar el árbol de decisión previamente entrenado utilizando el método export_graphviz(). Por cada nodo, nos indicará la condición o pregunta en caso de que sea un nodo de decisión, el grado de impureza, un array llamado Value, donde cada elemento se corresponde con el número de muestras u objetos de una case y, finalmente, la de la decisión tomada (clase).

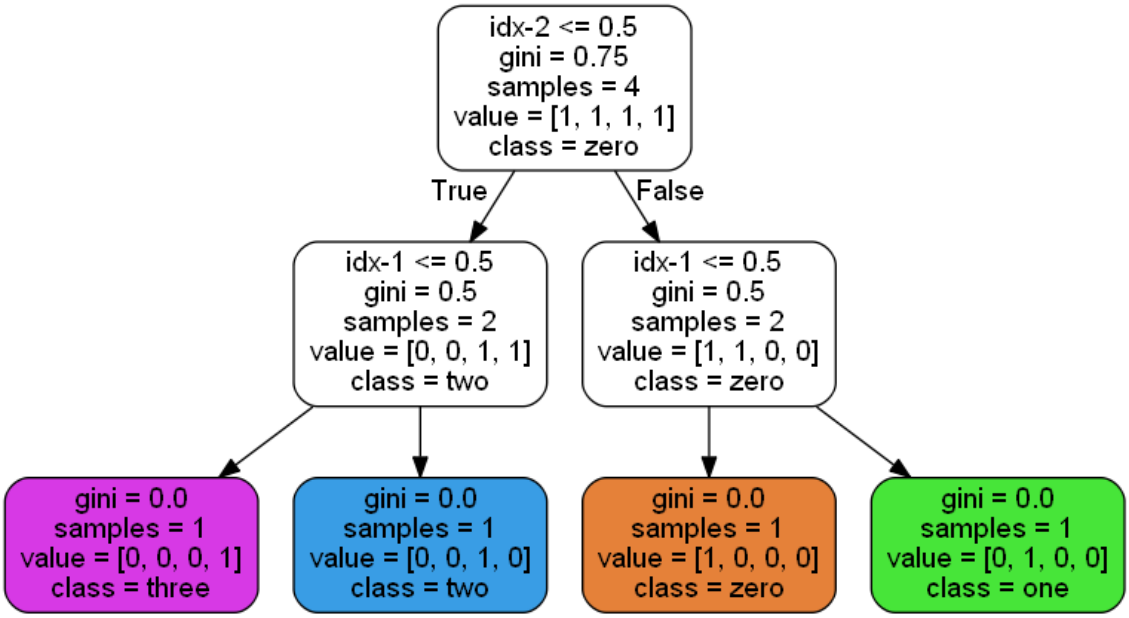

Figura 30 Ejemplo de propiedades de Random Forest

Este algoritmo cuenta con una serie de ventajas:

- Son capaces de adaptarse a datasets complejos.
- Es versátil, puede trabajar tanto con tareas de clasificación como de regresión.
- Son la base de otros algoritmos de ML más potentes.
- Son intuitivos, pueden ser entendidos por personas que cuenten con poca experiencia.

A continuación, se muestra la solución propuesta de RF en Python:

```
import pandas as pd
import numpy as np
import time
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import matthews_corrcoef
from sklearn.ensemble import RandomForestClassifier
```

```
#[1]
t0 = time.time()

X_train = np.load("Datasets/creditcard.X_train.npy")
X_test  = np.load("Datasets/creditcard.X_test.npy")
y_train = np.load("Datasets/creditcard.y_train.npy")
y_test  = np.load("Datasets/creditcard.y_test.npy")

print("Time (s): {0:.2f}".format(time.time()-t0))
```

```
#[2]
t0 = time.time()
pipeline_rf = Pipeline([
    ('model', RandomForestClassifier(n_jobs=2, random_state=1))
])
param_grid_rf = {'model__n_estimators': [100]}
grid_rf = GridSearchCV(estimator=pipeline_rf, param_grid=param_grid_rf, cv=5)
grid_rf.fit(X_train, y_train)

print("Time (s): {0:.2f}".format(time.time()-t0))
```

```
#[3]
t0 = time.time()
y_pred = grid_rf.predict(X_test)
print("Time (s): {0:.2f}".format(time.time()-t0))
print('MCC', matthews_corrcoef(y_test, y_pred))
```

Figura 31 Código Python: Fichero II. Solución Random Forest

Después de importar los módulos necesarios (ver primera celda del notebook), en [1] cargamos los ficheros con estructura Numpy que previamente habíamos creado (ver apartado *5.2 Preprocesado de los datos)* con el método load del módulo Numpy, al que le pasaremos como parámetro la ruta del fichero. A continuación, se muestra el tiempo en segundos que ha tardado en realizarse este proceso. Previamente a realizar la carga de ficheros, se inicializa un timer, (variable t0). La inicializaremos cada vez que deseemos consultar el tiempo que dura un proceso.

En [2], se ha utilizado un Pipeline al que añadiremos únicamente un paso. Un modelo *RandomForestClassifier*, al que le añadimos dos parámetros: n*_jobs*, número de hilos concurrentes utilizados que se utilizan para procesos en paralelo y R*andom_state*, que actúa de semilla aleatoria para hacer que el algoritmo sea determinista, es decir, que al ejecutarlo varias veces salga el mismo resultado.

Utilizando *GridSearchCV* podemos entrenar (con el método fit) y predecir (con método *predict*) al modelo utilizando un estimador, que será modelo anteriormente creado, le añadimos como segundo parámetro el número de estimadores, es decir, el número de árboles que tendrá RF, en este caso, será 100.

Mediante el método *fit* podemos entrenar al *gridSeachCV*, creado anteriormente. Añadiremos como parámetros los datos de entrenamiento de los valores de “X” y de “Y”. A la hora de predecir, con *predict* lo hacemos utilizando los datos del conjunto de pruebas de X.

Una vez realizada la predicción, debemos de comparar lo resultados obtenidos, almacenados en la variable y_pred, con lo que nos debería de haber dado, estos valores se encuentran almacenados en la variable y_test.

Podemos hacer esta comprobación mediante el coeficiente de correlación de Matthews (también llamado función de correlación de Phi) que determinará si existe una relación lineal entre dos variables y que esta relación no sea debida al azar, es decir, que la relación sea estadísticamente significativa. Se emplea cuando nos encontramos ante dos variables dicotómicas, es decir, variables para las cuales sólo es posible considerar dos modalidades.

5.5.2 Isolation Forest

Al igual que RF hace uso de árboles de decisión, Isolation Forest lo hará de árboles de asolación (en inglés isolation trees). Isolation trees es un algoritmo pensado para la detección de anomalías. Es por ello por lo que resulta interesante recurrir a él en nuestro caso, la detección de uso fraudulento de tarjetas bancarías. Este uso fraudulento no deja ser una anomalía, un uso no común de una tarjeta de crédito. Podríamos decir que las anomalías son ocurren poco y son diferentes al resto, y, por lo tanto, son más susceptibles al aislamiento.

En Isolation trees, la partición de las instancias se repite recursivamente hasta que todas las instancias están aisladas. Esta partición aleatoria produce rutas notoriamente más cortas para las anomalías. Las pocas instancias de anomalías dan como resultado un número menor de particiones, es decir, rutas más cortas en una estructura de un árbol. Por lo tanto, cuando Isolation Forest, un conjunto de Isolation trees, produce colectivamente longitudes de camino más cortas para algunos puntos particulares, será muy probable que sean anomalías.

A continuación, se muestra una comparación de un punto “común” (Xi) y una anomalía (Xo). Como podemos ver en la figura, la anomalía cuenta con una longitud de camino más corta que el punto común.

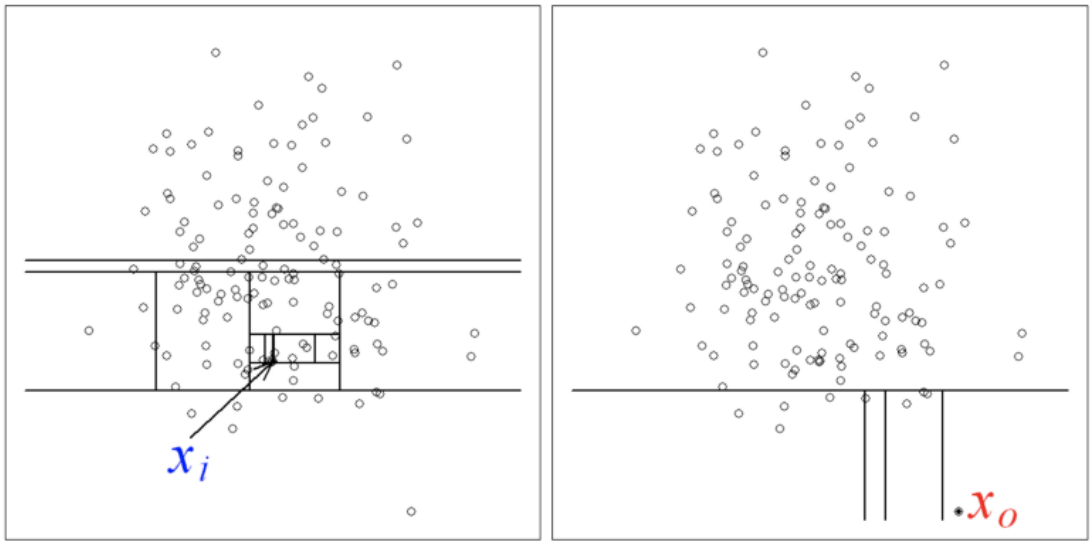

Figura 32 Ejemplo gráfico Isolation Forest

La detección de anomalías utilizando Isolation Forest consiste en un proceso de dos etapas. En la primera etapa se construye un conjunto de Isolation trees utilizando submuestras del conjunto de entrenamiento.

En la segunda etapa, utilizando el conjunto de pruebas, se pasa las instancias de prueba a través de Isolation trees para obtener una puntuación de anomalía para cada instancia. Al igual que Random Forest, se recurre a un sistema de votos, dónde la decisión que tome el mayor numero de votos será la elegida, debido a que es un algoritmo de clasificación y no de regresión, que se recurría a la media.

A continuación, se muestra la solución propuesta de IF en Python:

```
import pandas as pd
import numpy as np
import time
from sklearn.ensemble import IsolationForest
```

```
#[1]
t0 = time.time()

X_train = np.load("Datasets/creditcard.X_train.npy")
X_test  = np.load("Datasets/creditcard.X_test.npy")
y_train = np.load("Datasets/creditcard.y_train.npy")
y_test  = np.load("Datasets/creditcard.y_test.npy")

print("Time (s): {0:.2f}".format(time.time()-t0))
```

```
Time (s): 0.09
```

```
#[2]
t0 = time.time()

clf = IsolationForest(max_samples=200, behaviour='new', random_state=3)
clf.fit(X_train)

print("Time (s): {0:.2f}".format(time.time()-t0))
```

```
#[3]
t0 = time.time()

y_pred_test = clf.predict(X_test)
print("Time (s): {0:.2f}".format(time.time()-t0))
print("Accuracy:", list(y_pred_test).count(1)/y_pred_test.shape[0])
```

Figura 33 Código Python: Fichero III. Solución Isolation Forest

Después de importar los módulos y clases necesarias, en [1] cargamos los ficheros con estructura Numpy que previamente habíamos creado (ver apartado *5.2 Preprocesado de los datos),* en donde se almacenan los datos de entrenamiento de pruebas, con el método load del módulo Numpy, al que le pasaremos como parámetro la ruta del fichero. A continuación, se muestra el tiempo en segundos que ha tardado en realizarse este proceso. Previamente a realizar la carga de ficheros, se inicializa un *timer*, (variable t0). La inicializaremos cada vez que deseemos consultar el tiempo que dura un proceso.

En [2] vamos a utilizar el modelo de *IsolationForest* que contendrá tres parámetros: *Max_sample*, que es el número de muestras aleatorias que seleccionará del conjunto de datos original para crear árboles de aislamiento. *Behaviour*, al que le asignamos “new”, para que la función de decisión cambie para coincidir con otra API de algoritmo de detección de anomalías. Y *Random_state*, que actúa de semilla aleatoria para hacer que el algoritmo sea determinista, es decir, que al ejecutarlo varias veces salga el mismo resultado. Si no se lo indicamos, por defecto este algoritmo está compuesto de 100 Isolation trees.

Una vez definido el modelo, procedemos a entrenarlo con los datos de entrenamiento mediante el método *fit*. A diferencia de RF que utilizaba *y_train*, para poder clasificar (además de *x_train*), IF no lo hace, y sólo utiliza el conjunto de datos *x_train*. Esto es así debido a que IF no es un algoritmo de clasificación, se encarga de detectar anomalías, por lo tanto, no requiere de unos datos que clasifiquen los resultados. Le basta únicamente *x_train*.

A continuación, debemos de realizar pruebas [3]. Podemos hacerlo mediante el método *predict* del módelo previamente definido, al que le pasaremos como parámetro el conjunto de datos de pruebas, *x_test*. El resultado lo guardaremos en una variable llamada *y_pred_test*. Esta variable será de tipo array (vector undimensional), que contendrá un 1 si se trata de un resultado normal(inlier) y

un -1 en caso contrario(outlier). Se calcula la precisión para el caso de transacciones normales y se muestra en pantalla mediante la función *print*.

5.6 Análisis del resultado.

A continuación, se muestran los tiempos que ha tardado cada proceso y la precisión obtenida utilizando cada modelo. Podemos ver que IF, un algoritmo no supervisado utilizado comúnmente en la detección de anomalías tiene más precisión de acierto (0.90) que RF de clasificación (0.8576), un algoritmo supervisado. Además, los tiempos de entrenamiento son mucho menores en IF que en RF, aún utilizando el mismo número de árboles (100).

Resultados de tiempo y precisión de Random Forest:

```
import pandas as pd
import numpy as np
import time
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import matthews_corrcoef
from sklearn.ensemble import RandomForestClassifier
```

```
#[1]
t0 = time.time()

X_train = np.load("Datasets/creditcard.X_train.npy")
X_test  = np.load("Datasets/creditcard.X_test.npy")
y_train = np.load("Datasets/creditcard.y_train.npy")
y_test  = np.load("Datasets/creditcard.y_test.npy")

print("Time (s): {0:.2f}".format(time.time()-t0))
```

```
Time (s): 0.09
```

```
#[2]
t0 = time.time()
pipeline_rf = Pipeline([
    ('model', RandomForestClassifier(n_jobs=2, random_state=1))
])
param_grid_rf = {'model__n_estimators': [100]}
grid_rf = GridSearchCV(estimator=pipeline_rf, param_grid=param_grid_rf, cv=5)
grid_rf.fit(X_train, y_train)

print("Time (s): {0:.2f}".format(time.time()-t0))
```

```
Time (s): 454.85
```

```
#[3]
t0 = time.time()
y_pred = grid_rf.predict(X_test)
print("Time (s): {0:.2f}".format(time.time()-t0))
print('MCC', matthews_corrcoef(y_test, y_pred))
```

```
Time (s): 0.42
MCC 0.8576194535617819
```

Figura 34 Resultados de tiempo y precisión de Random Forest

Resultados de tiempo y precisión de Isolation Forest.

```
import pandas as pd
import numpy as np
import time
from sklearn.ensemble import IsolationForest
```

```
#[1]
t0 = time.time()

X_train = np.load("Datasets/creditcard.X_train.npy")
X_test  = np.load("Datasets/creditcard.X_test.npy")
y_train = np.load("Datasets/creditcard.y_train.npy")
y_test  = np.load("Datasets/creditcard.y_test.npy")

print("Time (s): {0:.2f}".format(time.time()-t0))
```

```
Time (s): 0.09
```

```
#[2]
t0 = time.time()

clf = IsolationForest(max_samples=200, behaviour='new', random_state=3)
clf.fit(X_train)

print("Time (s): {0:.2f}".format(time.time()-t0))
```

```
/Users/alejandrofk/anaconda3/lib/python3.7/site-packages/sklearn/ensembl
amination parameter 0.1 will change in version 0.22 to "auto". This will
  FutureWarning)
```

```
Time (s): 18.72
```

```
#[3]
t0 = time.time()

y_pred_test = clf.predict(X_test)
print("Time (s): {0:.2f}".format(time.time()-t0))
print("Accuracy:", list(y_pred_test).count(1)/y_pred_test.shape[0])
```

```
Time (s): 2.34
Accuracy: 0.9000561778027457
```

Figura 35 Resultados de tiempo y precisión de Isolation Forest

6. Conclusiones

A lo largo del presente trabajo, hemos visto como las prácticas de machine learning pueden ayudar a analizar mejor las amenazas y ser más efectivas a la hora de detener los incidentes de seguridad.

Dentro del sistema bancario, las aplicaciones para evitar el fraude se han caracterizado por el uso de técnicas más simples basadas en reglas estáticas, capaces de analizar un menor número de datos y rentabilizar en menor medida las variables que se tienen. En la actualidad, las técnicas de Machine Learning han permitido avanzar dentro de este campo, ya que son capaces de trabajar con grandes volúmenes de datos y variables.

Gracias a los algoritmos de Machine learning, tales como Random Forest e Isolation Forest, se ha conseguido detectar transacciones fraudulentas, que bien podrían tratarse de datos anómalos, de un gran volumen de transacciones en pocos segundos. Es aquí donde hemos comprobado que trabaja especialmente bien Isolation Forest frente a Random Forest, pese a tener un conjunto de datos clasificado o etiquetado.

Después de haber realizado este trabajo, he llegado a comprender el potencial de la inteligencia artificial, un campo que, pese a encontrarse en pleno crecimiento, ya ha demostrado logros sobresalientes en diferentes ámbitos tales como ciberseguridad, sanidad, automovilismo, marketing. Si bien es cierto que la IA ha sufrido de optimismo a lo largo de su historia, quizás estemos muy cerca de un verdadero auge, logrando cambiar la manera en la que trabajamos, pensamos y vivimos.

7. Bibliografía

1. Aurelien Geron. Hands-On Machine Learning with Scikit-Learn and TensorFlow. Publicación de 24 de marzo de 2017. Editorial O'Reilly Media.

2. Andreas C. Müller, Sarah Guido. Introduction to Machine Learning with Python: A Guide for Data Scientists 1st Edition. Publicación de 26 de septiembre de 2016. Editorial O'Reilly Media.

3. Wes McKinney. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython 2nd Edition. Publicación de 25 de septiembre de 2017. Editorial 0'Reilly Media.

4. Python Software Foundation. The Python Tutorial. Última actualización 31 de mayo de 2019. Disponible en: https://docs.python.org/3/tutorial/

5. Fei Tony Liu, Kai Ming Ting y Zhi-Hua Zhou. Isolation Forest. 15 de diciembre de 2008. Disponible en: *https://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/icdm08b.pdf*

6. Wikipedia. Artículo Inteligencia Artificial. Última actualización: 24 de abril de 2019. Disponible en: https://es.wikipedia.org/wiki/Inteligencia_artificial.

7. Niccolo Mejia. Artificial Intelligence at Visa – Current Use Cases and Services. Artículo de Emerj. 21 de marzo de 2019. Disponible en: https://emerj.com/ai-sector-overviews/ai-at-visa-current-projects-and-services/

8. MyCloudDoor. Presente y futuro del Machine Learning. 26 de abril de 2018. Disponible en: https://myclouddoor.com/es/presente-y-futuro-del-machine-learning/

9. Webroot Smarter Cybersecurity. Knowledge gaps: AI and Machine Learning in cybersecurity. 5 de diciembre de 2018. Disponible en: https://www-cdn.webroot.com/6015/4999/4566/Webroot_AI_ML_Survey_US-2019.pdf

10. Shubman Gupta. Tutorial Decision tree. Disponible en: https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/ml-decision-tree/tutorial/

11. Reclu IT. Historia y evolución del Machine Learning. 3 de agosto de 2018. Disponible en: https://www.recluit.com/historia-y-evolucion-del-machine-learning/#.XOnO6MgzaMo

12. Keith D. Foote. A brief History of Machine Learning. 26 de marzo de 2019. Disponible en: https://www.dataversity.net/a-brief-history-of-machine-learning/#

13. Jean-François Puget. The Most Popular Language For Machine Learning Is …Publicado el 19 de diciembre de 2016. Disponible en: https://www.ibm.com/developerworks/community/blogs/jfp/entry/What_Language_Is_Best_For_Machine_Learning_And_Data_Science?lang=en