**UNIVERSIDAD DEL VALLE DE GUATEMALA**
CC3094 - Security Data Science
Sección 10
Jorge Andres Yass Coy


# Proyecto SIMARGL

Paula Camila González Ortega, 18398
Diana Ximena de León Figueroa, 18607
Maria Inés Vásquez Figueroa, 18250

**GUATEMALA, 21 de febrero de 2022**

# Abstract

As for many other fields, data science and its methods have been very useful for detection, analysis and management, and for this paper these have been used for detection of malware, usage of machine learning models for intrusion detection, usage and explanation of metrics to determine the best model with the best results for the given dataset and all these to foment the usage of data science for cyber security. The given dataset is of the project SIMARGL (Secure Intelligent Methods of Advanced Recognition of malware and stegomalware). The objective of this project is to provide effective methods for detection of cyber threats, attacks and malware. They have given a dataset that showcases different attacks to study them and learns from their behavior. The models selected for these dataset are: Gradient Boosting Classifier, Random Forest Classifier and Adaboost Classifier. All of these were implemented in Python and followed with metrics like accuracy score, confusion matrix, cross validation and k-folds to select the best model based on these evaluations. The model with best accuracy was Random Forest and Gradient Boosting, with an equal average accuracy of 0.99. The worst accuracy was given by the model Adaboost, with a random accuracy of 0.71, which is very low.

# Introducción

Este proyecto consiste en la exploración y preparación de un dataset de características de una intrusión de malware con el objetivo de implementar modelos de machine learning que se basen en el tráfico de red para determinar posibles intrusiones. Se explican a grandes rasgos los conceptos relacionados a security data science que se aplicaron y los modelos de machine learning implementados: Gradient Boosting Classifier, Random Forest Classifier and Adaboost Classifier. Cuyas métricas resultantes fueron sometidas a evaluación para encontrar al mejor modelo, por tanto se encontró que Gradient Boosting Classification y Random Forest Classification fueron los más acertados.

# Marco Teórico

## Malware

Describe cualquier programa o código malicioso que es dañino para los sistemas. El malware intrusivo intenta invadir, dañar o deshabilitar ordenadores, sistemas informáticos, redes, tabletas y dispositivos móviles, asumiendo el control parcial de las operaciones de los dispositivos.

La intención del malware es sacarle de dinero al usuario de forma ilícita, aunque no puede dañar el hardware de los sistemas o el equipo de red, si se puede robar, cifrar o borrar sus datos, alterar o secuestrar funciones básicas del ordenador y espiar su actividad en el ordenador sin su conocimiento o permiso.

Las maneras más comunes en las que el malware obtiene acceso al sistema es mediante el internet y el correo electrónico, es decir básicamente todo el tiempo que está conectado a internet. Las aplicaciones maliciosas pueden ocultarse en aplicaciones aparentemente legítimas, especialmente cuando se descargan a través de sitios web o mensajes y no desde una App Store segura.

## Stegomalware

Es un tipo de malware que utiliza esteganografía, es la práctica de ocultar un archivo o mensaje dentro de otro archivo, para dificultar la detección. Opera construyendo en un sistema esteganográfico para ocultar datos maliciosos dentro de sus recursos, luego los extrae y ejecuta dinámicamente.

## SMOTE

Trabajar con conjuntos de datos desequilibrados tendrán un rendimiento deficiente en la clase minoritaria de las técnicas de aprendizaje automático. Un enfoque para abordar conjuntos desequilibrados es sobre muestrear la clase minoritaria. El enfoque más simple consiste en duplicar ejemplos de la clase minoritaria pero estos ejemplos no agrega información nueva al modelo, en su lugar se pueden sintetizar ejemplos a partir de los ejemplos existentes, un ejemplo de estos es Synthetic Minority Oversampling Technique (SMOTE).

SMOTE primero selecciona una instancia de clase minoritaria al azar y se encuentra sus k vecinos de clase minoritaria más cercanos. Luego, la instancia sintética se crea eligiendo uno de los k vecinos b más cercanos al azar y conectando a y b para formar un segmento de línea en el espacio de características. Las instancias sintéticas se generan como una combinación convexa de las dos instancias elegidas a y b.

## Gradient Boosting Classifier

Gradient boosting es una técnica de aprendizaje automático utilizado para análisis de regresión y problemas de clasificación. Está formado por un conjunto de árboles de decisión individuales entrenados de forma secuencial, de forma que cada nuevo árbol trata de mejorar los errores de los árboles anteriores. La predicción de una nueva observación se obtiene agregando las predicciones de todos los árboles individuales que forman el modelo.

## Random Forest Classifier

Random Forest es un modelo de aprendizaje automático, en el caso de este proyecto, para clasificar. Su método se basa en un conjunto de árboles de decisión combinados para formar un modelo poderoso. Diferentes árboles ven distintas porciones de los datos por lo que al entrenar cada árbol con diferente muestra de los datos para un mismo problema, los errores son compensados y se obtiene una mejor predicción.

De acuerdo con González (2018) para clasificar un nuevo objeto basado en atributos, cada árbol de decisión produce una clasificación y finalmente la decisión con mayor “votos” es la predicción del algoritmo. Para este proyecto se utilizó la librería de sklearn para la implementación del modelo  Random Forest Classifier porque ofrece hiper-parámetros útiles como el número de árboles que va a tener el bosque aleatorio, el número de cores que se pueden usar para entrenar los árboles, la profundidad máxima del árbol, entre otros.

### Adaboost Classifier

Adaboost es una abreviación de”Adaptive Boosting” y es algoritmos de clasificación estadística implementada por Yoav Freund y Robert Schapire. Este se puede combinar con otros para resultados más óptimos, como árbol de decisiones. Es una técnica de aprendizaje automático donde el algoritmo construye un modelo y otorga pesos iguales a todos los data points, luego le asigna más pesos a data points mal clasificados. De esta forma, en cada iteración los pesos más grandes tendrán más importancia y se seguirá entrenando hasta recibir un error mínimo.

# Metodología

## Exploración

Los dos datasets fueron cargados y se exploró cada uno de forma individual para mantener la integridad de la información, así como encontrar las diferencias y similitudes de estructura entre los datasets. Se encontró que ambos archivos contaban con 50 columnas, sin embargo la primera parte contaba con 3,570,666 observaciones y la segunda 8,637,207.

El 86% de los datos en ambos datasets eran de tipo numérico y únicamente el 14% de tipo categórico. Sin embargo, algunas de las columnas numéricas realmente eran categóricas ya que los únicos valores eran 0 y 1 como la de “DIRECTION”.

Para las gráficas se optó por visualizar aquellas variables de importancia en la predicción de la columna “LABEL” (imagen 1) y la matriz de correlación entre variables. Es importante mencionar que la mayoría de las correlaciones eran nulas. Otras gráficas elaboradas fueron la representación de los variables numéricas y las variables categórica utilizando la librería *quickda* y *seaborn* para ciertas gráficas. Además de generar reportes HTML con ProfileReport().

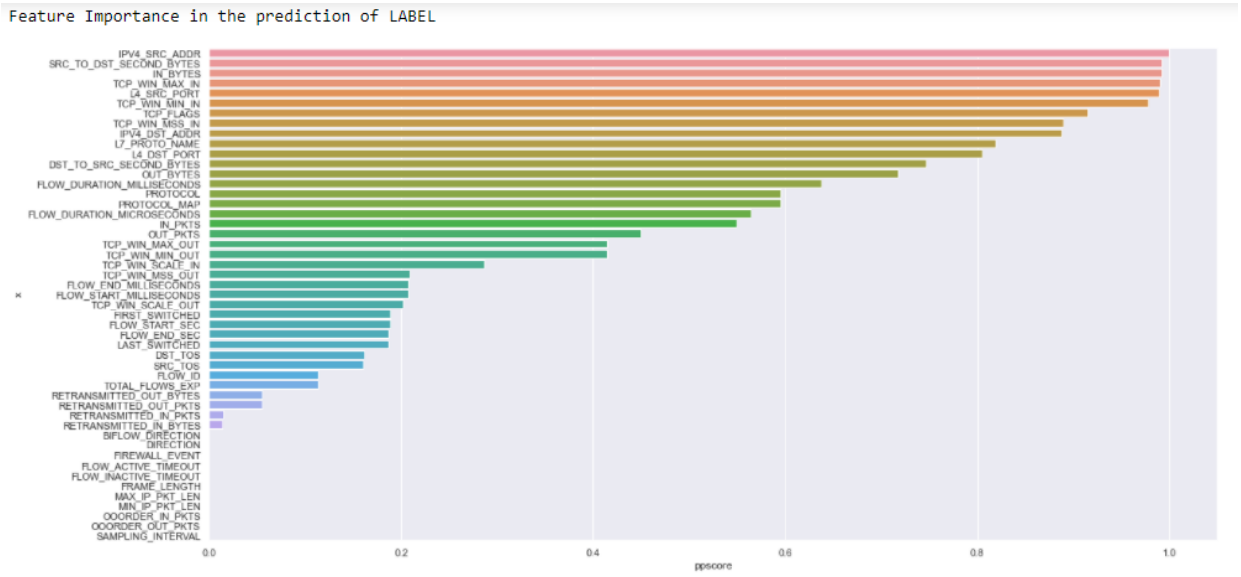


*Imagen 1.* Características importantes en la predicción de LABEL

## Preprocesamiento y selección de características

Se evaluaron los datos y se encontró que no existían nulos, ni duplicados por tanto se ahorró el paso de tomar decisiones con respecto a la información.

Encontramos que algunas características únicamente cuentan con un único valor en todo el dataset, por tanto no están aportando información nueva. También se encontró información redundante ya que los valores se encontraban en dos características en microsegundos y milisegundos, ya que la mayor parte de características se encuentran en milisegundos decidimos mantener esa medida. Además se descartaron algunas columnas como el ID ya que no es información relevante para el modelo. Finalmente, las columnas eliminadas fueron las siguientes:

- SAMPLING_INTERVAL
- MIN_IP_PKT_LEN
- MAX_IP_PKT_LEN
- FRAME_LENGTH
- FLOW_INACTIVE_TIMEOUT
- FLOW_ACTIVE_TIMEOUT
- FIREWALL_EVENT
- BIFLOW_DIRECTION
- FLOW_DURATION_MICROSECONDS
- FLOW_END_SEC
- FLOW_START_SEC
- L7_PROTO_NAME
- SRC_TO_DST_SECOND_BYTES
- DST_TO_SRC_SECOND_BYTES
- FLOW_ID
- IPV4_DST_ADDR
- IPV4_SRC_ADDR

Las características de PROTOCOL_MAP y PROTOCOL se convirtieron en categóricas utilizando One Hot Encoding. Después procedimos a evaluar las características para seleccionar las más significativas utilizando SelectKBest con la función chi2. En base a esto se seleccionaron las siguientes características:

- FLOW_DURATION_MILLISECONDS
- FLOW_END_MILLISECONDS
- FLOW_START_MILLISECONDS
- IN_BYTES
- L4_DST_PORT
- L4_SRC_PORT
- OUT_BYTES
- RETRANSMITTED_OUT_BYTES
- TCP_WIN_MAX_IN
- TCP_WIN_MAX_OUT
- TCP_WIN_MIN_IN
- TCP_WIN_MIN_OUT
- TCP_WIN_MSS_IN
- TCP_WIN_MSS_OUT
- TOTAL_FLOWS_EXP

## Implementación de modelos

Para entrenar los modelos se seleccionó una parte del dataset aplicando un shuffle a todo el dataset, luego se fraccionó en chunks y de esos se seleccionó el 12%, para luego aplicarle un oversampling utilizando la técnica SMOTE para trabajar con el desbalance de las clases. Además escalando la información del dataset con MinMaxScaler. Debido a que la cantidad de datos después del oversampling era de casi cuatro millones, se modificó a seleccionar un 15% de cada chunk, y no aplicarle ningún tipo de balanceo ya que se consideró que la diferencia entre clases ya no era tan significativa.

Luego de separar la data en training y testing se indagó sobre modelos de Machine Learning eficaces en la clasificación de datos multi-label ya que el dataset del proyecto contaba con dicha característica. Se optó por implementar el modelo de Gradient Boosting Classifier, Random Forest Classifier y Adaboost Classifier.

El modelo Random Forest Classifier fue el más difícil de implementar debido a la falta de recursos en memoria. Se realizaron varios intentos en distintos equipos e incluso antes de lograr este modelo se intentó implementar el Classifier Chains, pero no fue posible porque no se contaba el espacio en memoria que su predicción requería. Se indagó más y se encontró una optimización para la división de la data con Random Under Sampler, lo cuál permitió la implementación del modelo de Random Forest exitosamente.

# Resultados

Luego de dividir la data de entrenamiento, testing y validación sin realizar oversampling, las métricas de los modelos variaron y los tiempos de ejecución se redujeron a comparación de

la data dividida con Smote. A continuación se muestran las curvas ROC de cada modelo, así como la cross validation con k = 10.

## Gradient Boosting Classification

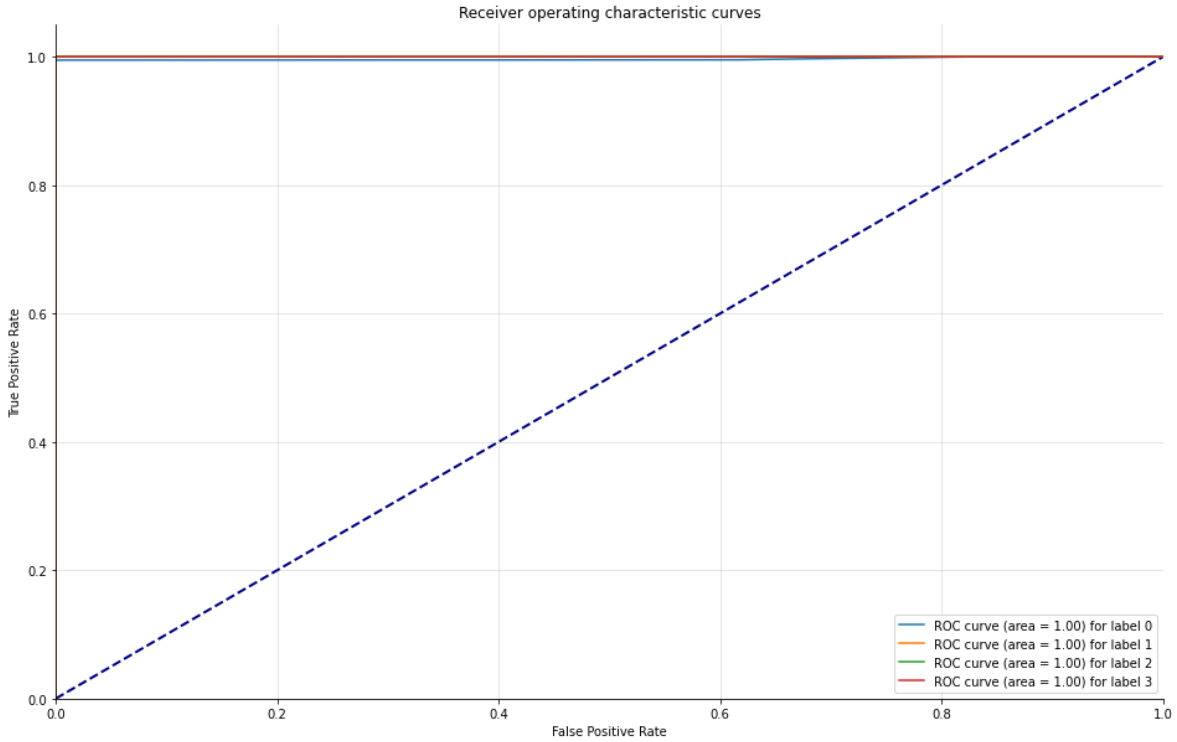


*Imagen 2.* Curva ROC de Gradient Boosting Classification para cada valor del target (LABEL)

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 68465 |
| 1 | 1.00 | 1.00 | 1.00 | 64961 |
| 2 | 1.00 | 1.00 | 1.00 | 92344 |
| 3 | 1.00 | 1.00 | 1.00 | 46161 |
| accuracy | | | 1.00 | 271931 |
| macro avg | 1.00 | 1.00 | 1.00 | 271931 |
| weighted avg | 1.00 | 1.00 | 1.00 | 271931 |

*Imagen 3.* Reporte de clasificación de validation data.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 138549 |
| 1 | 1.00 | 1.00 | 1.00 | 133293 |
| 2 | 1.00 | 1.00 | 1.00 | 187447 |
| 3 | 1.00 | 1.00 | 1.00 | 92811 |
| accuracy | | | 1.00 | 552100 |
| macro avg | 1.00 | 1.00 | 1.00 | 552100 |
| weighted avg | 1.00 | 1.00 | 1.00 | 552100 |

*Imagen 4.* Reporte de clasificación de test data

## Random Forest Classification

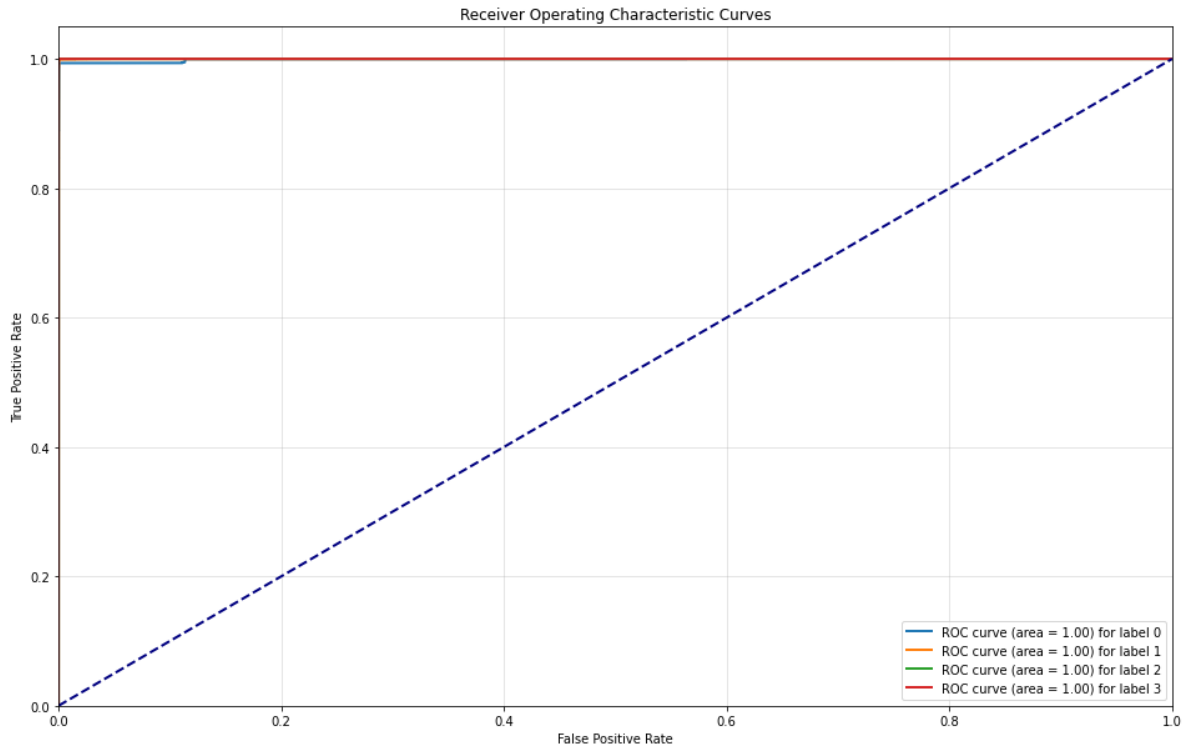


*Imagen 5.* Curva ROC de Random Forest para cada valor del target (LABEL)

```
rf_cv_score=cross_val_score(estimator=clf, X=X_train, y=y_train, cv=10, n_jobs=-1)
print(rf_cv_score)
```

```
[0.97336047 0.97434344 0.97274487 0.97404557 0.97549521 0.97372785
 0.97421437 0.97182148 0.97431366 0.97426376]
```

*Imagen 6.* Cross Validation de Random Forest donde se observan los excelentes resultados de precisión, Accuracy

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 0.99 | 0.99 | 296629 |
| 1 | 0.99 | 1.00 | 0.99 | 112819 |
| 2 | 0.96 | 1.00 | 0.98 | 103440 |
| 3 | 1.00 | 0.94 | 0.97 | 39212 |
| accuracy | | | 0.99 | 552100 |
| macro avg | 0.99 | 0.98 | 0.98 | 552100 |
| weighted avg | 0.99 | 0.99 | 0.99 | 552100 |

*Imagen 7.* Reporte de clasificación de test data

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 0.99 | 0.99 | 146418 |
| 1 | 0.99 | 1.00 | 0.99 | 55607 |
| 2 | 0.96 | 1.00 | 0.98 | 50707 |
| 3 | 1.00 | 0.94 | 0.97 | 19199 |
| accuracy | | | 0.99 | 271931 |
| macro avg | 0.99 | 0.98 | 0.98 | 271931 |
| weighted avg | 0.99 | 0.99 | 0.99 | 271931 |

*Imagen 8.* Reporte de clasificación de validation data

## Adaboost Classifier

Para el modelo Adaboost Classifier se obtuvieron los siguientes resultados:

```
abc.score(X_validation, y_validation)
```

```
0.7060063030695287
```

*Imagen 7. Accuracy de Validation Data*

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.78 | 0.71 | 0.74 | 68465 |
| 1 | 0.89 | 0.79 | 0.84 | 64961 |
| 2 | 0.61 | 1.00 | 0.76 | 92344 |
| 3 | 0.00 | 0.00 | 0.00 | 46161 |
| accuracy | | | 0.71 | 271931 |
| macro avg | 0.57 | 0.62 | 0.58 | 271931 |
| weighted avg | 0.61 | 0.71 | 0.64 | 271931 |

*Imagen 8. Reporte de clasificación de Validation Data*

```
# Model Accuracy, how often is the classifier correct?
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7067342872667995
```

*Imagen 3: Accuracy de Test Data*

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.78 | 0.70 | 0.74 | 138549 |
| 1 | 0.89 | 0.79 | 0.84 | 133293 |
| 2 | 0.61 | 1.00 | 0.76 | 187447 |
| 3 | 0.00 | 0.00 | 0.00 | 92811 |
| accuracy | | | 0.71 | 552100 |
| macro avg | 0.57 | 0.62 | 0.58 | 552100 |
| weighted avg | 0.62 | 0.71 | 0.64 | 552100 |

*Imagen 9. Reporte de clasificación de Test Data*

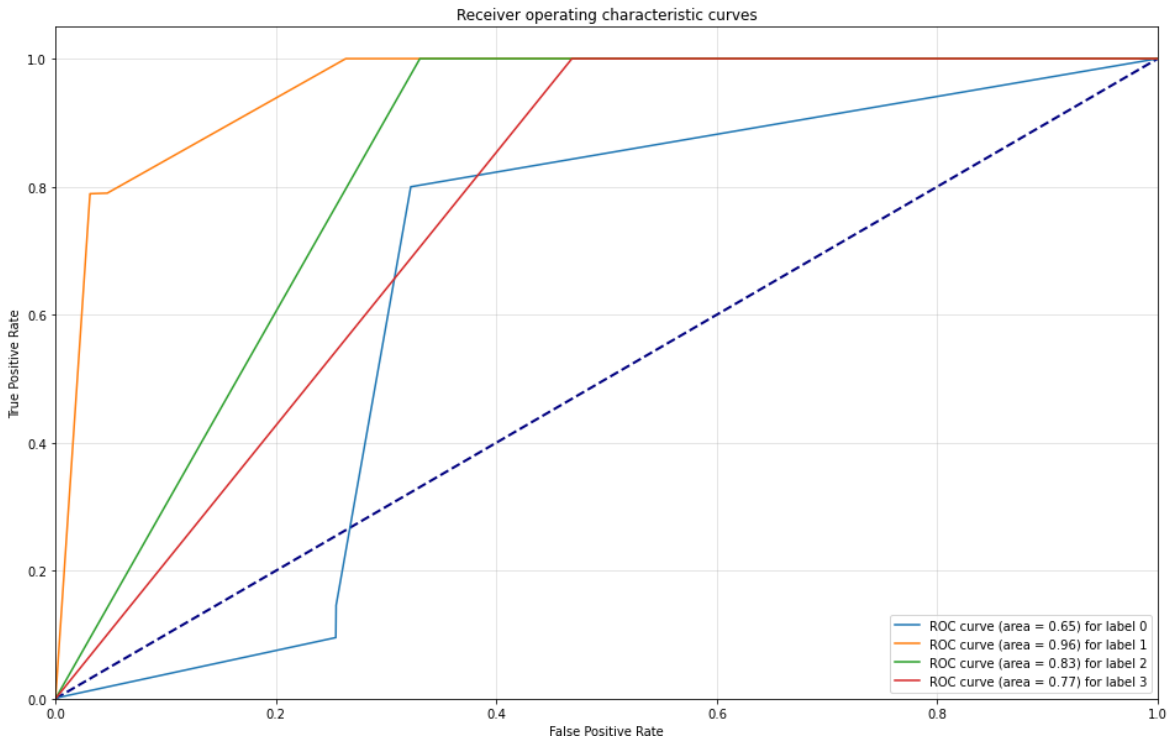


*Imagen 10. Gráfica curva ROC*

```
# report performance
print('Accuracy: %.3f (%.3f)' % (np.mean(scores), np.std(scores)))
```

```
Accuracy: 0.706 (0.001)
```

*Imagen 11. Cross Validation*

Como se puede observar, es evidente que el modelo Adaboost dio resultados bastante mediocres y no aceptables, dado que su accuracy es en promedio 0.71. Los resultados de la curva ROC fueron especialmente buenos para los labels 1 y 2, dado que se mantienen más cercanas a la esquina superior izquierda, pero los demás no se alejaron tanto. A pesar de este ser un modelo incluso premiado y muy utilizado no fue de mucho valor para el dataset dado, lo cual demuestra que uno debe evaluar cercanamente varios modelos con un dataset para solo seleccionar aquellas que se acoplen correctamente a las necesidades. Es una lástima que este modelo fuera tan poco preciso, dado que su tiempo de entrenamiento y procesamiento fueron bastante rápidos, pero probablemente esta sea la razón por la cual sus resultados fueron tan bajos.

# Conclusiones

- Todos los modelos de Machine Learning seleccionados para clasificar la información funcionan bien con datos multi-label, sin embargo el tiempo de entrenamiento y predicción que cada uno requiere es distinto y por lo tanto sus métricas de evaluación varían.
- Se concluye que el modelo más veloz para entrenar y predecir fue el Random Forest, y también fue el que tenía accuracy y precisión más alto.
- La capacidad de memoria de las computadoras limita el número de observaciones a utilizar del dataset, lo que descarta gran cantidad de observaciones de manera aleatoria.
- El modelo de Adaboosting no cumplió con nuestras expectativas, dado que su accuracy fue bajo, de 0.71, demostrando que no fue el modelo ideal para nuestro dataset.

## Recomendaciones

- Probar los modelos aplicando un undersampling para evaluar los resultados.
- Realizar validaciones con información de otros datasets que aún no conozca el modelo para comprobar los resultados obtenidos.
- Empezar probando los modelos con dataset parciales del dataset original para hacer el proceso de pruebas más rápido, y luego ya evaluarlo con un dataset más grande.
- Utilizar modelos que funcionen para dataset desbalanceados, en lugar de intentar balancear las clases.

# Referencias

Amat, R. (s. f.). Gradient Boosting con Python. Ciencia de datos. Recuperado 21 de febrero de 2022, de https://www.cienciadedatos.net/documentos/py09_gradient_boosting_python.html

Gonzalez, L. (2020, 19 agosto). Aprendizaje Supervisado: Random Forest Classification. https://aprendeia.com/aprendizaje-supervisado-random-forest-classification/

Heras, J. M. (2020, 18 septiembre). Random Forest (Bosque Aleatorio): combinando árboles.https://www.iartificial.net/random-forest-bosque-aleatorio/#%C2%BFQue_es_un_Random_Forest

Saini, A. (2021, 15 septiembre). *AdaBoost Algorithm - A Complete Guide for Beginners*. Analytics Vidhya. Recuperado 25 de febrero de 2022, de https://www.analyticsvidhya.com/blog/2021/09/adaboost-algorithm-a-complete-guide-for-beginners/

Wikipedia contributors. (2022, 11 febrero). *AdaBoost*. Wikipedia. Recuperado 25 de febrero de 2022, de https://en.wikipedia.org/wiki/AdaBoost