Universitat Oberta de Catalunya (UOC)

Máster Universitario en Ciencia de Datos (*Data Science*)

TRABAJO FINAL DE MÁSTER

Área: Minería de datos y Machine Learning

Detección de anomalías en entorno del Internet de las cosas

Autor: Gonzalo Pedro Mellizo-Soto Díaz

Tutor: Carlos Hernández Gañán

Profesor: Jordi Casas Roma

Madrid, 8 de junio de 2019

Copyright

FICHA DEL TRABAJO FINAL

Título del trabajo:	Detección de anomalías en el entorno del Internet de las cosas
Nombre del autor:	Gonzalo Pedro Mellizo-Soto Díaz
Nombre del colaborador/a docente:	Carlos Hernández Gañán
Nombre del PRA:	Jordi Casas Roma
Fecha de entrega (mm/aaaa):	06/2019
Titulación o programa:	Máster Universitario en Ciencia de Datos
Área del Trabajo Final:	Minería de datos y Machine Learning
Idioma del trabajo:	Español
Palabras clave	Machine Learning, IOT, Anomaly Detection

Cita

"Nuestro lema es: más humanos que los humanos"

Eldon Tyrell, *Blade Runner*

Abstract

In recent years the amount of connected devices has greatly increased, with an increasing number of applications in the industry each day. This devices can be subject of attacks causing instability or data leaks that can be dangerous both for the users and the enterprises, in order to avoid or confront them, security and early detection are becoming a must in a connected world. The focus is the monitoring and detection of the attacks in Internet of Things devices using state of the art Machine Learning techniques. Models such as SVM, DBScan or Isolation Forests have been used and assembled in order to identify with a better accuracy when an attack is happening. With this assembly, attack detection has increased up to 15 % comparing to traditional methods and individual model usages and times have been considerably reduced. An active use of Machine Learning models has shown a great improvement at anomaly detection by securing the devices and decreasing the reaction times when facing attacks.

Durante los últimos años se encuentra una creciente cantidad de dispositivos conectados entre sí, cada vez con más aplicaciones en la industria. Estos dispositivos pueden ser atacados y provocar inestabilidad o una fuga de datos, por lo tanto la protección y la pronta detección de ataques y/o anomalías es vital en un mundo cada vez más conectado. El objetivo es la monitorización y detección de estos ataques en dispositivos del *Internet of Things* utilizando técnicas del estado del arte de Machine Learning para su detección y poder así responder con una mayor rapidez a los ataques. Para la detección se han utilizado modelos estadísticos, como SVM, DBScan o Isolation Forests, que en su conjunto permitan identificar con mayor precisión cuando se está produciendo un ataque. El conjunto de la clusterización con la clasificación de puntos anómalos muestra una mayor robustez, frente al uso individual de cada uno de los modelos aumentando la detección en hasta un 15 %. Se demuestra cómo el uso de los modelos permite proteger los dispositivos y mejorar la seguridad al disminuir los tiempos de reacción frente a los ataques.

Palabras clave: Machine Learning, IOT, Anomaly Detection

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Contexto y justificación del Trabajo

Cada vez se encuentran más dispositivos conectados entre sí, no solo en la industria, sino también en los hogares, esta conexión entre dispositivos los hace vulnerables a ataques informáticos que pueden afectar en gran medida a los usuarios, no solo pueden provocar un mal funcionamiento de los mismos, sino que también puede provocar la fuga de datos de distinta sensibilidad. La previsión en los futuros años es estar cada vez más conectados y una pronta detección de los ataques puede ayudar a evitar los problemas derivados de los mismos, mediante una pronta reacción o haciendo frente a la previsión de un ataque.

Actualmente se utilizan distintas medidas de seguridad como control de acceso físico al dispositivo, encriptación de datos, firewalls, securización de red, etc... Sin embargo, en muchos casos la detección de anomalías aplica un umbral estacionario, provocando que en el caso de un ataque ya sea demasiado tarde para reaccionar o no se sea capaz de identificar patrones extraños previos al ataque. Con la aplicación de nuevas técnicas se pretende mejorar los tiempos de reacción e incluso predecir cuándo puede suceder un ataque.

1.2. Explicación de la motivación personal

La razón principal de la elección del proyecto es la posibilidad de aprender y utilizar técnicas de Machine Learning en un sector desconocido que permita diversificar conocimientos. Todo se encuentra cada vez más conectado e investigar cómo clasificar cuando se está produciendo un ataque permite profundizar en conocimientos de seguridad y aplicarlos en un problema real.

La aplicación de técnicas en un problema real ayuda a comprender mejor el uso de las herramientas y el porqué y cuándo se deben de utilizar. De este modo, se añade un nuevo conocimiento que aportar al ámbito profesional y puede utilizarse también en un entorno privado.

1.3. Objetivos del Trabajo

Los objetivos del trabajo son los siguientes:

- Adquisición de conocimiento del sector y de los dispositivos IOT.
- Lectura y comprensión de las técnicas del estado del arte en detección de anomalías en dispositivos conectados.
- Obtención de datos reales de conexiones a dispositivos IOT, en caso de no ser posible, generación de datos sintéticos.
- Prueba de las técnicas encontradas y evaluación en el problema actual.
- Investigación de algoritmos tradicionales de Machine Learning y su utilidad.
- Desarrollo de la solución utilizando los algoritmos más aptos.
- Evaluación de la detección de patrones y ataques de las técnicas utilizadas.
- Comparación de los resultados obtenidos con el estado del arte.

1.4. Descripción general del problema

La cantidad de dispositivos informáticos existentes que pueden ser víctimas de un ataque es masiva, por lo que securizar correctamente los dispositivos es fundamental. A pesar del uso de distintos protocolos de seguridad, se generan nuevos tipos de ataque todos los días, por lo tanto la detección es una necesidad a la hora de evitar los problemas derivados.

La pérdida de control de los dispositivos o la fuga de información de los mismos puede provocar pérdidas millonarias a las empresas o provocar una gran inseguridad a los usuarios de productos IOT, así como causar posibles daños a infraestructuras o personas. Por otro lado, muchos de los dispositivos pueden no tener la capacidad de computación necesaria para incluir una capa de seguridad robusta, que puede compensarse con una detección temprana.

1.5. Enfoque y método seguido

El enfoque pasa por conocer el problema y dar respuesta a preguntas específicas, mediante la aplicación de los conocimientos obtenidos y la evaluación de la propia aplicación de los mismos.

El método seguido durante el desarrollo se basa en el marco de trabajo *Agile* pensando en el desarrollo de la solución como un producto. De este modo se podrá iterar sobre una arquitectura definida y enfocarse en el desarrollo del producto en su totalidad, frente a un modelo tradicional de cascada donde cada fase de desarrollo recae en una sola parte del proyecto.

El formato de entrega será mediante un *minimum viable product* (MVP) en sprints de dos semanas durante el periodo de desarrollo y evaluación de las necesidades según la evolución del producto.

1.6. Planificación del Trabajo

Para la planificación del trabajo se han subdividido las tareas principales y se han mostrado en el siguiente diagrama de Gantt 1.1.

- Pec01 - Definición

- Pec01 - Planificación
- Pec02 - Búsqueda de fuentes del estado del arte
- Pec02 - Lectura de estado del arte
- Pec02 - Justificación de estado del arte
- Pec02 - Redacción
- Pec02 - Refinamiento de objetivos
- Pec03 - Planificación de Sprints
- Pec03 - Sprint 1
- Pec03 - Sprint 2
- Pec03 - Sprint 3
- Pec03 - Refinamiento/Redacción
- Pec04 - Revisión de apartados anteriores
- Pec04 - Redacción de nuevos apartados
- Pec05 - Presentación y defensa

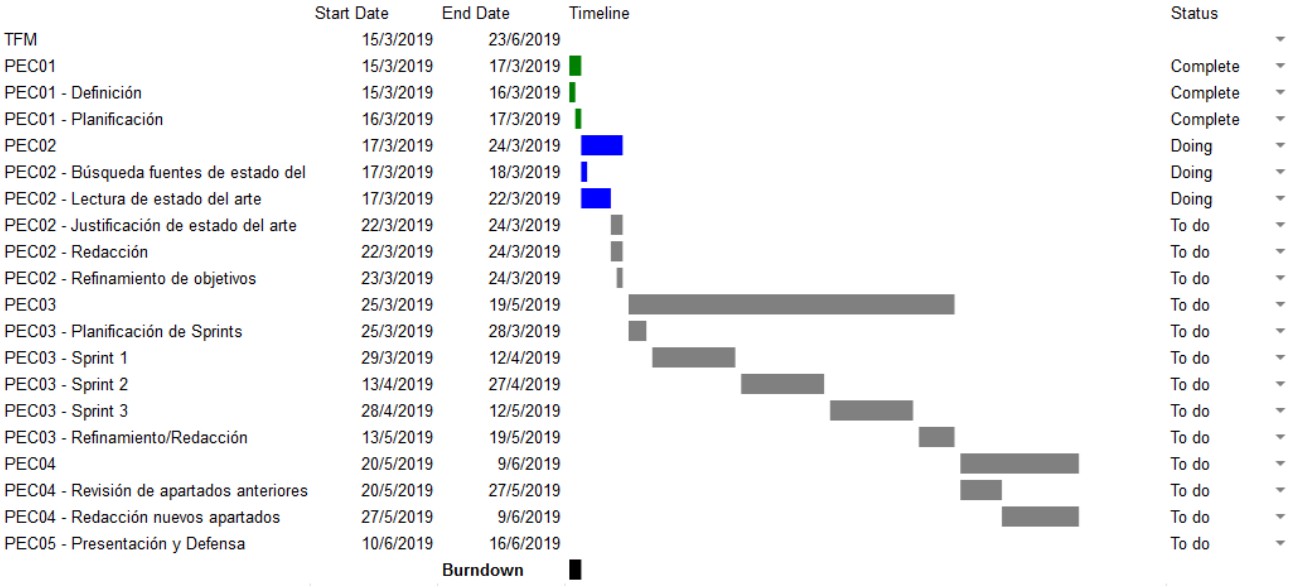


Figura 1.1: Planificación de tareas

Capítulo 2

Estado del Arte

Una de las aplicaciones más comunes dentro de la detección de anomalías es para la detección de intrusos (*Intrusion Detection*) y la creación de sistemas de detección de intrusos (*Intrusion Detection System, IDS*). Se trata de sistemas cuyo objetivo es la monitorización de los sistemas y redes informáticas, con el fin de alertar en caso de que puedan existir brechas de seguridad[1].

Con la ingente cantidad de redes y sistemas que se pueden encontrar a día de hoy es necesario incluir uno de estos sistemas, con el fin de mantener la integridad y disponibilidad de los mismos. Dentro de los IDS, se pueden identificar dos grandes implementaciones de estos sistemas, en los sistemas de detección de intrusos en Host (*Host-Based Intrusion Detection Systems, HIDS*) y los sistemas de detección de intrusos en red (*Network Intrusion Detection Systems, NIDS*).

- **HIDS:** Estos sistemas se caracterizan por implementarse en el Host utilizando información del propio sistema operativo para detectar actos maliciosos [2]. Esta información tiene distintos niveles de información, pero por lo general tienden a ser de bajo nivel sobre operaciones que se pueden estar realizando dentro del sistema. Esta información se consulta dentro de logs, por lo que el análisis de la información es más lento.

- **NIDS:** Para el segundo caso la monitorización se realiza a sistema de red, es decir, de comunicaciones entre distintos nodos y la monitorización de los paquetes que viajan entre ellos [1]. Esta información puede ser consumida en tiempo real, por lo que la reacción ante algún evento es más rápida que en los HIDS que necesitan revisar las acciones.

2.1. Métodos tradicionales de detección de anomalías

En la sección actual se describen algunos de los métodos tradicionales utilizados en IDS, tanto para HIDS como para NIDS.

- Network Security Monitor (NSM): se trata de uno de los primeros sistemas que permitió auditar el tráfico que circulaba dentro de la red [3]. El sistema escucha pasivamente dentro de la red y detecta si existe una conducta sospechosa al desviarse de patrones de conducta. La mayor parte de la monitorización se basa en protocolos estándar como *telnet, ftp, TCP/IP, etc.* por lo que le permitía utilizar una gran cantidad de datos heterogéneos.

- State transition analysis (USTAT): el sistema parte de que el host en un momento se encuentra en un estado seguro y que según las acciones que se realizan sobre el mismo el host cambia de estado, hasta que llega a un estado en el que compromete la seguridad [3]. Este sistema analiza los estados por los que ha pasado la máquina desde el estado seguro al comprometido.

- GrIDS: se trata de un IDS que utiliza un sistema de construcción de grafos basados en la red, donde cada nodo representa a un host y las aristas las conexiones entre los mismos. La representación gráfica de la actividad de la red permite ayudar al espectador en identificar qué está sucediendo [3].

- Haystack: en este caso el IDS se ayuda de métodos estadísticos para la detección de anomalías, definiendo estrategias para usuarios y grupos, además de definir variables del modelo como variables gaussianas independientes [4]. Para la detección se incluyen una serie de intervalos en los valores que en el momento que salen del rango normal, se calcula la distribución de probabilidades y si el *score* o puntuación es demasiado grande se genera una alerta.

Los métodos/sistemas listados se desarrollaron durante los años noventa, la tecnología ha evolucionado desde entonces y los sistemas se han vuelto más complejos y más propensos a los ciberataques. Por ello, se han desarrollado nuevas técnicas que se apoyan en el uso de técnicas de minería de datos (*Data Mining*) y las técnicas que se describirán a continuación de *Machine Learning*.

2.2. Machine Learning y la detección anomalías

El *Machine Learning* es una rama de la inteligencia artificial cuya premisa es hacer que la máquina aprenda una tarea sin haber sido específicamente programada para ello. El término fue descrito por Arthur L. Samuel en 1959 en un artículo en el que explica estudios de *Machine Learning* aplicado al juego de las damas [5], utilizando en una primera instancia métodos de aprendizajes más generales, como las redes neuronales de las que hablaremos más adelante, y otro métodos que tendrán que ser parametrizados para sus distintos usos.

La inteligencia artificial se puede entender como la inteligencia ejercida por las máquinas, al contrario que la inteligencia natural inherente a los humanos, que son capaces de realizar tareas cognitivas que permiten potenciar la resolución de las mismas e imitar comportamientos como el aprendizaje [6]. Dentro de la inteligencia artificial se pueden encontrar distintas definiciones según si se centran en el razonamiento o en el comportamiento:

- Actuar humanamente: este enfoque se basa en que las máquinas actúen como humanos más centrado en la interacción de máquinas con personas, más que en la resolución de problemas. Esta interacción se puede ver reflejada en el test ideado por Alan Turing en 1950, en el que se somete a la máquina ïnteligente.ª un interrogatorio realizado por un humano, donde éste no sabe que está hablando con una máquina, por lo tanto si es incapaz de detectar que se trata de una máquina esta ha actuado como un humano y puede considerarse inteligente.

- Pensar humanamente: esta vertiente se centra en imitar el pensamiento humano, entender cómo funcionan las mentes de los mismos y replicarlo en máquinas. Siguiendo esta corriente, si se consigue imitar el pensamiento humano, una máquina que resuelva el problema utilizará razonamientos humanos, en lugar de solucionar problemas a toda costa, independientemente de como lo realizan los humanos.

- Pensar racionalmente: basado en la lógica formal desarrollada en en siglos XIX y XX que permite formular los problemas en un lenguaje formal, utilizando el razonamiento matemático para resolver éstos.

- Actuar racionalmente: se centra en realizar el comportamiento más efectivo en un momento dado. Antes las distintas situaciones no siempre existe una acción correcta, pero si se puede llegar realizar una acción que minimice los riesgos.

Los enfoques mostrados conllevan sus propios enfoques filosóficos, sin embargo, han influenciado en gran medida a cómo se afrontan los problemas y cómo se han desarrollado las técnicas de inteligencia artificial que están en uso actualmente.

Como se ha comentado con anterioridad, el *Machine Learning* es una rama de la inteligencia artificial, dentro de la cual existe otra rama, *Deep Learning*. Las relaciones entre los términos se puede observar en la imagen 2.1 y como la inteligencia artificial engloba ambas ramas.

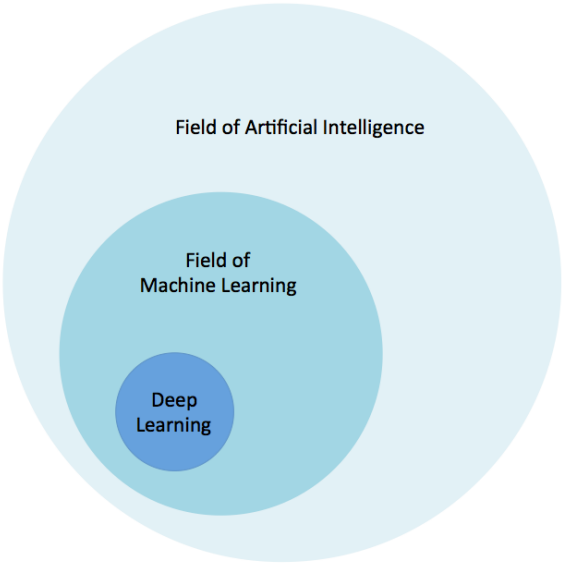


Figura 2.1: Relación entre Inteligencia Artificial, Machine Learning y Deep Learning

El *Deep Learning* se diferencia del *Machine Learning* convencional, en que están formados por modelos con distintas ramas de procesamiento que son capaces de extraer las representaciones de los datos con varios niveles de abstracción [7]. Mucha de la teoría relacionada se puede encontrar en los años 50 y 60, sin embargo, las aplicaciones y la capacidad de cómputo eran limitadas en la época, hasta ahora donde la capacidad de computación ha crecido a gran escala y en conjunto con las nuevas técnicas propuestas, una mayor cantidad de datos y la aplicación de transformaciones no lineales hacen que esta vertiente viva su época dorada.

Una de las implementaciones más representativas del *Deep Learning* se trata de las redes neuronales, las cuales se encuentran basadas en el funcionamiento del cerebro humano, utilizando el concepto de neuronas que pueden realizar cálculos, la conexión entre las mismas, la función de realizar una tarea específica, etc. Añadir, que se encuentra basado y que el cerebro humano es un sistema mucho más complejo que las redes neuronales y tiene muchos más comportamientos [8].

Dentro del *Machine Learning*, los algoritmos pueden dividirse según como se realice el entrenamiento del modelo, afectando también a la evaluación y las aplicaciones del mismo. Estas categorias son: *Aprendizaje Supervisado (Supervised Learning), Aprendizaje No Supervidado (Unsupervised Learning) y Aprendizaje Semi-Supervisado (Semi-Supervised Learning)*

2.2.1. Aprendizaje Supervisado

El aprendizaje supervisado se caracteriza por utilizar un conjunto de variables de entrada y encontrar la función que más se aproxime a los valores de salida [9]. Actualmente es una de las metodologías más utilizadas en *Machine Learning* y es una de las más avanzadas en el campo, sin embargo, la necesidad de los valores de salida (variable dependiente o *target*) requiere que el conjunto de datos se encuentre etiquetado, es decir, para las observaciones de las variables de entrada debe existir la variable de salida para poder general el modelo. En muchos casos este etiquetado se debe de realizar manualmente, por lo que es un proceso que requiere mucho tiempo.

Por otro lado, cabe mencionar que el uso de métodos de aprendizaje supervisado tienen una mayor efectividad que el semi y no supervisado, sin embargo, la necesidad de tener las etiquetas y el coste que ello supone para grandes cantidades de datos, como es en el caso de la detección de anomalías, provocan que se busquen otras soluciones con aprendizajes semi y no supervisado. Además, suele suceder que en la detección de anomalías, éstas sean la menor parte de todos los datos, en otras palabras, la mayor parte de los datos son normales y solo una pequeña porción se trata de anomalías como tal, provocando que el conjunto de datos esté poco compensado.

Uno de los modelos más sencillos de aprendizaje supervisado es la regresión lineal (simple), ésta permite ajustar a una serie de puntos conocidos , x, y su respuesta y [10], generando una función que permite predecir los valores de y con valores no conocidos de x:

$$y = ax + b \tag{2.1}$$

En este caso se predicen los nuevos valores utilizando la función ajustada, existen distintos métodos para encontrar la recta que ajusta los puntos, siendo uno de los más utilizados el ajuste

por mínimos cuadrados:

$$a = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2} \tag{2.2}$$

$$b = y - a\bar{x} \tag{2.3}$$

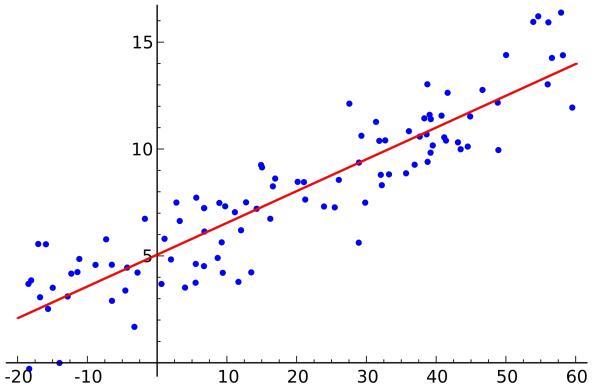


Figura 2.2: Regresión Lineal

Como se puede observar en la imagen 2.2, se ha generado una recta capaz de ajustarse a los puntos y predecir el valor para nuevos puntos. Para generar la función de la recta se ha necesita utilizar la variable dependiente y en el entrenamiento para poder generar la función que aproximará los nuevos puntos.

Durante los siguientes apartados se van a mostrar los distintos modelos utilizados en aprendizaje supervisado para la detección de anomalías, como las redes neuronales artificiales (*Artificial Neural Networks, ANN*, *Gradient Boosting Machines* y *Support Vector Machines*

2.2.1.1. Feedforward Neural Network

Tal y como se ha comentado anteriormente, las redes neuronales intentan imitar el concepto de la conexión de neuronas del cerebro humano, éstas están formadas por una capa de entrada de datos (*Input*), una capa de salida (*Input*) y un conjunto de capas intermedias, también llamadas capas ocultas (*Hidden*) [8]. En las capas intermedias se encuentran las neuronas conectadas entre sí y pueden estar formadas desde una sola capa, hasta n capas, sin embargo, cuanto mayor sea

el número de capas, mayor coste de computación asociado. En la siguiente imagen 2.3 podemos ver como sería una arquitectura de una red neuronal:

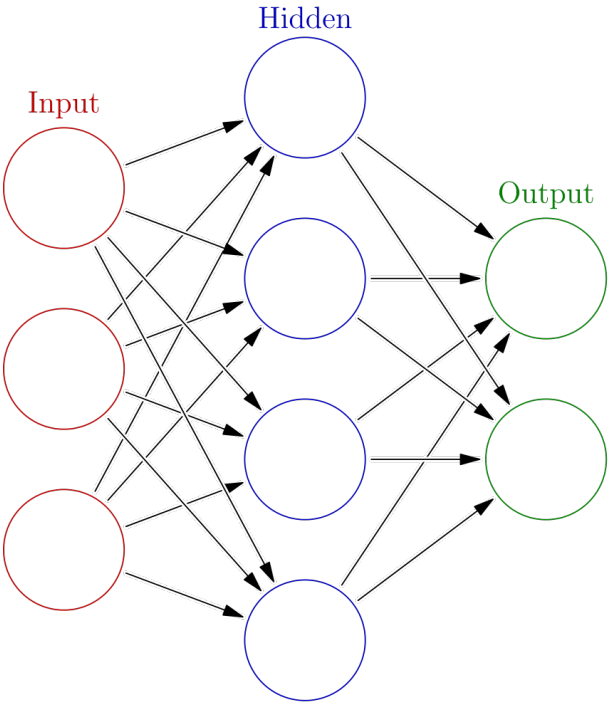


Figura 2.3: Red Neuronal Artificial

Dentro de las redes neuronales existe un caso base conocido como perceptron, un clasificador binario formado por la capa de entrada, una capa oculta de una sola neurona y una salida [11]. El perceptrón ayudará a incluir el concepto de los pesos (*weights*) y el *bias*.

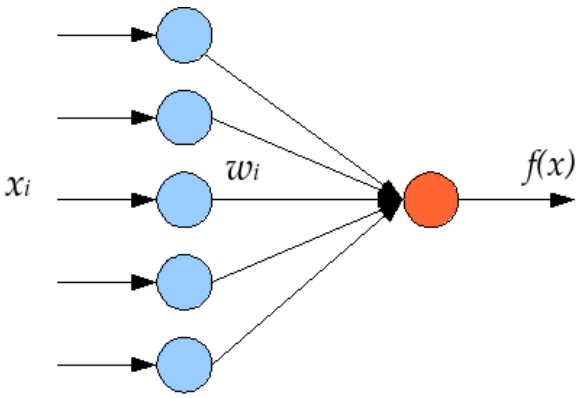


Figura 2.4: Arquitectura de un Perceptron

Los valores de entrada X son multiplicados por una serie de pesos W, en una primera instancia se inician aleatoriamente, y se realiza la suma de esta multiplicación, tras la cual se pasará a una función de activación. La función de activación define el valor de salida de la neurona y que se utilizará como valor de entrada para la siguiente neurona o como es en este caso como el valor de salida final. Un ejemplo de una función de activación es la función escalón, donde los valores menores de cero son iguales a cero y los mayores de cero son igual a uno. El *bias* se trata de un pequeño valor que modifica el output sin interactura con las neuronas. De este modo la clasificación de 0 o 1 en un perceptron sería así:

$$f(x) = g(\sum XW + b) \tag{2.4}$$

Donde $g(x)$ es la función de activación, X el vector de datos de entrada, W el vector de pesos y b el vector de bias.

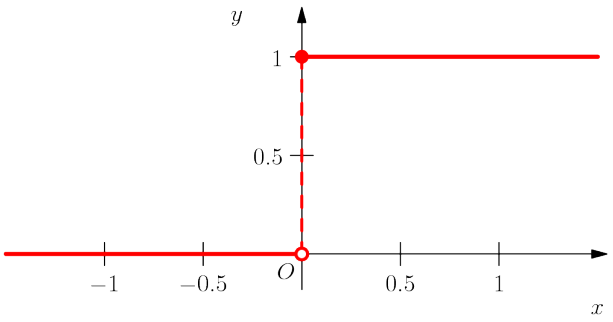


Figura 2.5: Función de activación: Función Escalón

Aumentando la complejidad de la arquitectura, es decir, el número de neuronas, el numero de capas y distintas funciones de activación, se consiguen modelos más complejos pero mantienen el enfoque que se realiza en el perceptron.

Para optimizar la salida, se utilizan varias técnicas de optimización que permiten actualizar los pesos W y disminuir el error $E = f(x) - y$, una de las más utilizadas es el *backpropagation*, basado en el descenso del gradiente cuyo objetivo es obtener el gradiente de la función (vector de mayor pendiente) pero utilizar el valor opuesto del mismo multiplicado por un coeficiente de aprendizaje (*learning rate*), una analogía muy utilizada es imaginar como encontrar el punto más bajo del valle, para ello lo mejor es fijarse en que lugar del punto en el que nos encontremos tiene más pendiente negativa y seguir ese camino. El backpropagation utiliza esté método, pero en lugar de calcular el gradiente sobre la función total (gran coste) se realiza el cálculo de los

gradientes locales, empezando desde el punto final de la función.

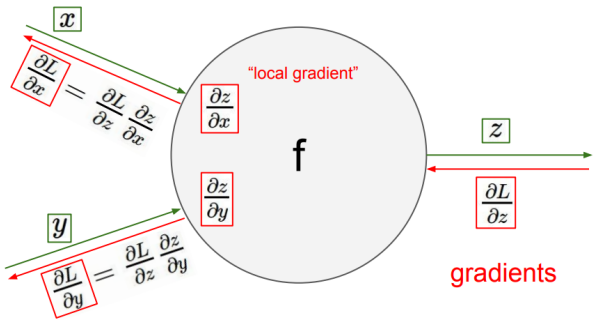


Figura 2.6: Backpropagation

La fórmula de la actualización de los pesos en una ANN sería similar a la siguiente:

$$W_{i+1} = W_i - \gamma dW \tag{2.5}$$

Donde γ es el *learning rate* y $L(x)$ es la función de pérdida.

La última capa, la capa de *outputs*, define el número de neuronas de salida, que puede ser igual al número de clases etiquetadas o una sola neurona en el caso de que existan solo dos clases, clasificación binaria. El valor de las salidas irá definido por la función de activación de la última capa de la capa oculta.

2.2.1.2. Gradient Boosting Machines

Las *Gradient Boosting Machines* son uno de los métodos más utilizados actualmente, dada la gran precisión que aportan a la resolución de los problemas. Estos modelos se basan en realizar el ensamblado de modelos débiles, normalmente árboles de decisión, para generar un predictor "fuerte"de una manera iterativa.

El término *Boosting* se refiere al método utilizado para mejorar el aprendizaje de un modelo mediante el ensamblado de modelos débiles, esto quiere decir que son modelos con una capacidad de predicción mejor que la pura aleatoriedad [12].

Concretamente el *Gradient Boosting* trata de generar un primer árbol para realizar las predicciones, es decir, crea un función $F(x)$ para aproximar y, posteriormente se utiliza la

función de coste definida (puede ser un error cuadrático medio para una regresión, por ejemplo) para ver como de bien ha realizado la predicción. Los residuales obtenidos se utilizan para crear un nuevo modelo que utilice estos residuales con el fin de volver a minimizar el error y finalmente se genera un nuevo modelo que utilice ambos. De esta forma en cada iteración su puede añadir n cantidad de nuevos modelos que ayuden a mejorar las predicciones mediante la corrección de los errores de los modelos previos [13].

Se crea el primer modelo

$$F_1(x) = y \tag{2.6}$$

Se crea un segundo modelo con los residuales

$$h_1(x) = y - F_1(x) \tag{2.7}$$

Se genera un nuevo modelo con los anteriores

$$F_2(x) = F_1(x) + h_1(x) \tag{2.8}$$

Dentro del ajuste a los residuales es donde entra la parte del gradiente, tal y como hemos explicado anteriormente se utiliza el método del descenso del gradiente (*Gradient Descent*) para la optimización de la función de pérdida. Para cada paso que se realiza, en vez de calcular los residuales como en la fórmula 2.7 se ajustan calculando el gradiente de la función de pérdida y ajustando un nuevo modelo, h, con los nuevos residuales y en la generación del nuevo modelo incluirle *gamma* (γ), similar al concepto explicado de *learning rate*:

$$r = -[\frac{\partial L(F(x_i), y)}{\partial F(x_i)}] \tag{2.9}$$

$$F_2(x) = F_1(x) + \gamma h_1(x) \tag{2.10}$$

En la práctica el uso de estos modelos se ha comprobado que es realmente eficaz para la resolución de problemas. Dentro de estos algoritmos se pueden destacar dos en concreto, cuyas implementaciones se han utilizado en varias soluciones dentro de las competiciones realizadas por Kaggle.

XGBoost

Diminutivo de *eXtreme Gradient Boosting* se trata de una nueva implementación de GBM realizada por Chen, Tianqi, et al [14], que demuestra una mayor escalabilidad y velocidad que los métodos tradicionales, utilizando una menor cantidad de recursos de los sistemas.

LightGBM

Desarrollado por Microsoft [15] se propone mejorar los problemas encontrados en algoritmos como *XGBoost* cuando existe una alta cantidad de variables, esto es ocasionado por la necesidad de evaluar el punto óptimo para la división de los árboles de decisión. Propone su propia forma de efectuar la ganancia de información (por división) y demuestra una mejora en el tiempo de cómputo sobre los modelos convencionales.

Algunas de las competiciones donde se han utilizado los algoritmos (también ensamblándolos) son las siguientes:

- Home Credit Default Risk: Can you predict how capable each applicant is of repaying a loan? [16].
- Corporación Favorita Grocery Sales Forecasting: Can you accurately predict sales for a large grocery chain? [17].
- Google Analytics Customer Revenue Prediction: Predict how much GStore customers will spend [18].

2.2.1.3. Support Vector Machines

Se trata de otro algoritmo enfocado a la clasificación mediante la separación de los puntos por un hiperplano [10]. Por ejemplo, para un set de datos etiquetados con dos categorías (clasificación binaria) se busca encontrar la línea (hiperplano) que mejor separe ambos puntos, como puede observarse en la imagen 2.7 existen distintos hiperplanos (H_1, H_2, H_3) que separan las distintas clases, como puedo observarse tanto H_1 como H_2 son capaces de separar las dos clases por completo, mientras que H_3 fallaría en la tarea de clasificación .

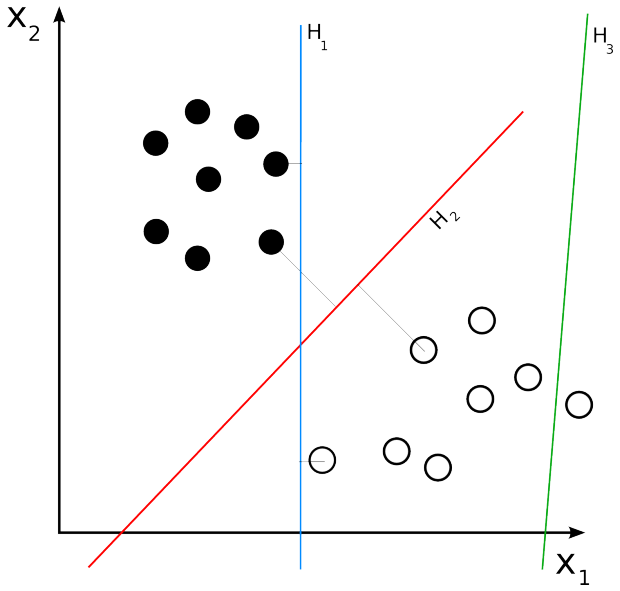


Figura 2.7: Hiperplanos SVM

Se puede definir el hiperplano como la "línea"que puede dividir los datos entre las dos clases, es decir, se pretende separar el espacio en dos mitades. Esta "línea"para un problema de p dimensiones sería de $p-1$ dimensiones, en otras palabras, para un un problema de dos dimensiones sería una línea, para uno de tres sería un plano y así sucesivamente hasta n dimensiones. Esta separación indica que según donde se encuentren los puntos con respecto al hiperplano se clasificarán como una clase u otra.

$$\beta_0 + X_1\beta_1 + X_2\beta_2 + ... + X_n\beta_n = 0 \tag{2.11}$$

Como se ha mencionado en la imagen 2.7, existen varios hiperplanos, sin embargo, se debe de buscar el hiperplano óptimo de los posibles, siendo este el que se encuentra más alejado de los puntos de entrenamiento, es decir, es el hiperplano con mayor margen entre los puntos y el hiperplano. Sin embargo, si solo nos basamos en encontrar el mejor hiperplano según los márgenes se puede dar el caso de que un nuevo punto afecte en gran medida al hiperplano, por maximizar los márgenes, por ello se utiliza un parámetro C que permite .[a]blandar.[el] clasificador y permitir que ciertos puntos puedan encontrarse entre el hiperplano y el margen, la mala clasificación de unos poco puntos permitidos evita sobreentrenar el modelo y permitir generalizar mejor.

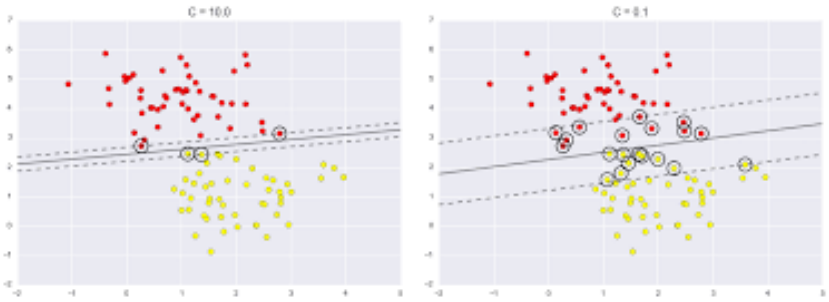


Figura 2.8: Hiperplanos para distintos valores de C

Por ahora, se ha hablado de separar los datos linealmente y en dos clases, sin embargo, ese no es siempre el caso, podemos tener más de dos clases o por ejemplo tenemos datos que no sean separables por una "línea", pensemos en dos conjuntos de puntos que forman una serie de circunferencias concéntricas (2.9), para este caso una línea nunca logrará separar bien los puntos. Para está última, la *SVM* puede utilizar los *kernels*, se trata de funciones que permiten cambiar la dimensionalidad del espacio para ayudar a la separación de las clases.

Kernel lineal

$$K(x_i, x_{i'}) = \sum_{j=1}^{p} x_i j x_{i'j} \tag{2.12}$$

Kernel polinomial

$$K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_i j x_{i'j})^d \tag{2.13}$$

Kernel radial

$$K(x_i, x_{i'}) = exp(-\gamma \sum_{j=1}^{p} (x_i j x_{i'j})^2) \tag{2.14}$$

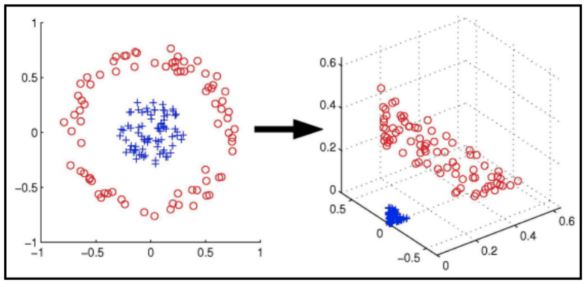


Figura 2.9: Efecto de aplicación de *Kernel* radial en SVM

Para el caso de tener más de dos clases se pueden aplicar dos enfoques distintos:

- Clasificación uno contra uno: para este caso se crean una SVM por cada par de clases y realiza la clasificación para ellos, luego el resultado es elegido observando cuantas veces el punto cae en una categoría para las distintas SVM clasificándolo a la clase más frecuente.

- Clasificación uno contra todos: en este caso se realiza una SVM por clase, de modo que se compare la clase escogida con el resto y valorando en cual de los modelos generados es más apta la clasificación del punto.

2.2.2. Aprendizaje No Supervisado

Este nuevo tipo de aprendizaje se distingue porque no necesita ese conjunto de etiquetas que se utiliza por observación, es decir, en este caso no se tiene el valor de la clase o el valor numérico correspondiente a cierto conjunto de datos. Para este caso no se enfoca a la predicción de los valores, debido a que no tenemos esa variable *target* para la que queremos desarrollar la función que la aproxime lo máximo posible, en este caso, el objetivo principal es intentar descubrir nuevos patrones o información oculta dentro de los datos [10].

Muchas veces el uso del aprendizaje no supervisado no se encuentra con el simple objetivo de predecir, si no que puede ser parte de un proceso de análisis de los datos, de modo que permitan encontrar más información útil de los mismos, como por ejemplo, se puede realizar una segmentación de usuarios, de modo que posteriormente se pueda estudiar que causa esta segmentación de los mismos y mejorar los servicios que se les puedan ofrecer.

El uso del aprendizaje no supervisado esta creciendo cada vez más, dado que en la mayor parte de los casos los datos no vienen etiquetados, tal y como se ha comentado anteriormente el etiquetado de los mismos es un proceso costoso y además mientras que en el supervisado se puede "supervisar"si el resultado es correcto en la fase de entrenamiento, mientras que para el no supervisado, no se tiene esta capacidad de conocer cual es el valor real, dado que es imposible saber cual es la respuesta correcta.

2.2.2.1. Análisis de Componentes Principales

Se trata de uno de los algoritmos más conocidos cuya finalidad es explicar en una serie de çomponentes", siendo la cantidad de éstos menor a la cantidad de variables de los datos iniciales, que permiten explicar la mayor parte de la varianza de los datos originales. Estos çomponentes"se trata de un conjunto de nuevas variables que son capaces de explicar una gran parte de la varianza de un conjunto mayor, en otras palabras, permite obtener gran parte de la varianza explicada en un dimensionalidad menor.

En las siguientes imágenes se ha utilizado un set de datos sintéticos de dos dimensiones, en el que se muestra como en la imagen de la izquierda 2.10 se encuentran nubes de datos que cambian en función de los ejes x e y y como la aplicación del análisis de componentes principales 2.11, *PCA* de ahora en adelante, genera nuevas variables (dos componentes principales) que tienden a explicar la varianza de las variables originales. En este caso se puede apreciar como la aplicación del algoritmo "mueve"las observaciones, haciendo que así sean más identificables en a lo largo de los ejes.

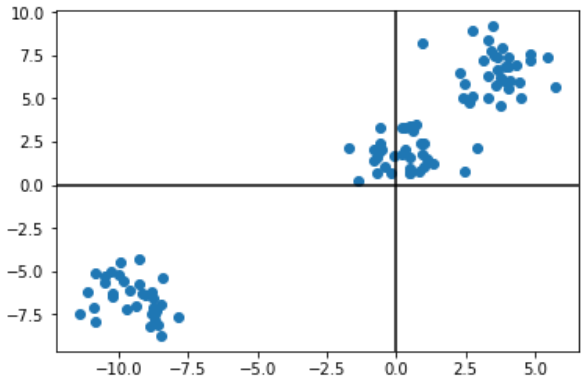


Figura 2.10: Datos de dos variables sin PCA

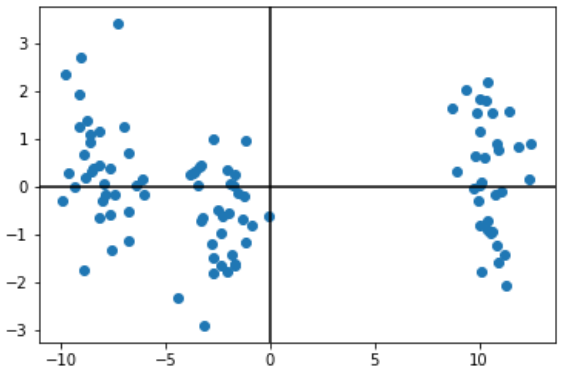


Figura 2.11: PCA aplicado a dos variables

Este ejemplo se trata de un caso sencillo, por lo general esta técnica también se puede utilizar para visualizar los datos de una gran dimensionalidad en dos o tres variables, utilizando ese número de componentes principales, que se tratan de una combinación lineal de las variables anteriores [10].

Para realizar el cálculo del PCA, se puede realizar mediante el uso de álgebra lineal, más en concreto con una descomposición de autovalores y autovectores que siguen los siguientes pasos

para una matriz X con p variables y n observaciones [19]:

Se calcula la media de cada una de las columnas

$$\overline{X} = \overline{X_1}, \overline{X_2}, ...\overline{X_p} \tag{2.15}$$

Se normaliza el valor de las columnas restando la media de cada columna a todos los valores de la matriz

$$X_{norm} = X - \overline{X} \tag{2.16}$$

Se genera la matriz de covarianza con la matriz normalizada, siendo E la media, por cada par de variables [20]

$$Cov[X_i, X_j] = E[(X_i - E[X_i])(X_j - E[X_j])] \tag{2.17}$$

De la matriz de covarianza se calculan los autovalores y autovectores

$$\lambda, \nu < -X_{cov} \tag{2.18}$$

Una vez obtenidos los autovectores y autovalores, se seleccionarán un cantidad m que será la cantidad de componentes principales elegidos

$$B = [\lambda_1 ... \lambda_m, \nu_1 ... \nu_m] \tag{2.19}$$

Y con estos calculamos la proyección P de la matriz original X con m componentes principales

$$P = B^T.A \tag{2.20}$$

2.2.2.2. Agrupación

Tal y como el propio nombre indica, actualmente se utilizan métodos de agrupación en el que podemos encontrar grupos o *clusters*. Esta agrupación se realiza mediante la similitud entre observaciones, donde la similitud puede ser la distancia euclídea entre las observaciones [10], útil cuando todas las variables son numéricas. En la imagen 2.12 podemos ver un ejemplo donde datos (sin etiquetar, imagen de la izquierda) se analizan y se agrupan (imagen de la derecha). En este caso la agrupación es clara, sin embargo, la agrupación se puede realizar con datos más abstractos y menos representables dada su alta dimensionalidad.

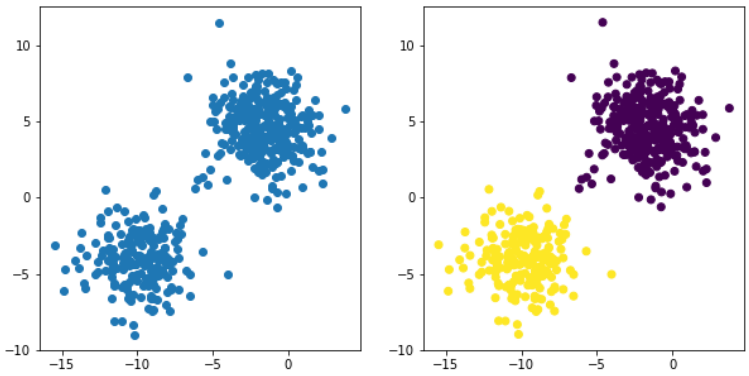


Figura 2.12: Ejemplo de agrupamiento

Como se ha comentado, la similitud entre observaciones ayuda a encontrar que grupos son más similares, por lo tanto dependiendo del problema se pueden utilizar distintas distancias como la euclídea o manhattan, siendo estas de las más utilizadas:

Distancia euclídea

$$d(X_1, X_2) = \sqrt{(X_{11} - X_{21})^2 + (X_{12} - X_{22})^2 + ... + (X_{1n} - X_{2n})^2} \tag{2.21}$$

Distancia manhattan

$$d(X_1, X_2) = \sum_{i=1}^{n} |X_{1i} - X_{2i}| \tag{2.22}$$

Estas distancias se aplican a datos con variables numéricas por lo que no son adecuadas cuando existe variables discretas, sin embargo, se pueden realizar transformaciones en estas

variables a numéricas para poder utilizarlas. En el caso donde la mayor parte de las variables sean categóricas esta transformación no daría bueno resultados, por lo que se pueden utilizar distancias específicas para variables discretas, como *Goodall, Lin* o *Smirnov* [21].

Dentro de los algoritmos de clusterización los más conocidos son los siguientes:

- KMeans o K-medias: es un algoritmo de clusterización en el cual se indica a priori el número clústers K que se quieren realizar [10]. Una vez definido se generan k centroides aleatorios, se calculan las distancias de estos centroides al resto de puntos de modo que se asocia a cada punto el centroide que más cercano se encuentre (se puede decir que se le asigna una clase o grupo) y finalmente se actualiza el centroide utilizando los valores medios de todos sus puntos asociados. Se vuelven a calcular las distancias, asignar de nuevo los centroides y la actualización de éstos hasta que no se realizan grandes cambios, es decir, converge o hasta un número máximo de rondas.

- DBSCAN: este algoritmo se basa en la generación de los grupos mediante la identificación de áreas de alta densidad (de puntos) frente a zonas de poca densidad [22]. En este caso no se necesita indicar el número de clústers a priori, pero si un número mínimo de observaciones como para poder considerar que se trata de un nuevo grupo. En este algoritmo no todas las observaciones se agrupan, si no cumplen la cantidad mínima esta observaciones se pueden considerar como *outliers* o anomalías.

- Algoritmos jerárquicos: en este tipo de algoritmos se define una medida de disimilitud (distancia euclídea, por ejemplo) entre los distintos puntos (considerados como un grupo cada uno), de modo que aquellos puntos con la menor disimilitud se agrupan en un nuevo clúster, generando un nuevo grupo de puntos. Con este grupo de puntos surge una nueva medida que permite identificar la disimilitud entre los mismos, *linkage*. Por lo general existen distintos modos de *linkage* siendo los más comunes: completo (se calcula la disimilitud entre todos los puntos de los grupos y se selecciona la mayor), *single* (se selecciona la menor), media (se hace la media de todas las disimilitudes) y centroide (se calcula la disimilitud entre los centroides de los grupos)

Tal y como se ha comentado anteriormente, generalmente se suelen utilizar conjuntos de varias técnicas que permitan mejorar los resultados obtenidos utilizando una técnica sola, un ejemplo en la detección de anomalías es el uso de PCA y KMeans, arrojando buenos resultados en dispositivos IOT [23].

2.2.2.3. Autoencoders

Los autoencoders son redes neuronales con una determinada estructura, de modo que no se necesite un set de datos etiquetados (variable dependiente o *target*), el objetivo de esta red neuronal es reconstruir los datos de entrada en la capa de salida. Esto se realiza utilizando el mismo número de neuronas de entrada en la salida, de modo que utilizando las técnicas de las redes neuronales, como la optimización por descenso del gradiente y el *backpropagation* se puedan general nuevas variables (*features*) que reconstruyen los datos de entrada [24].

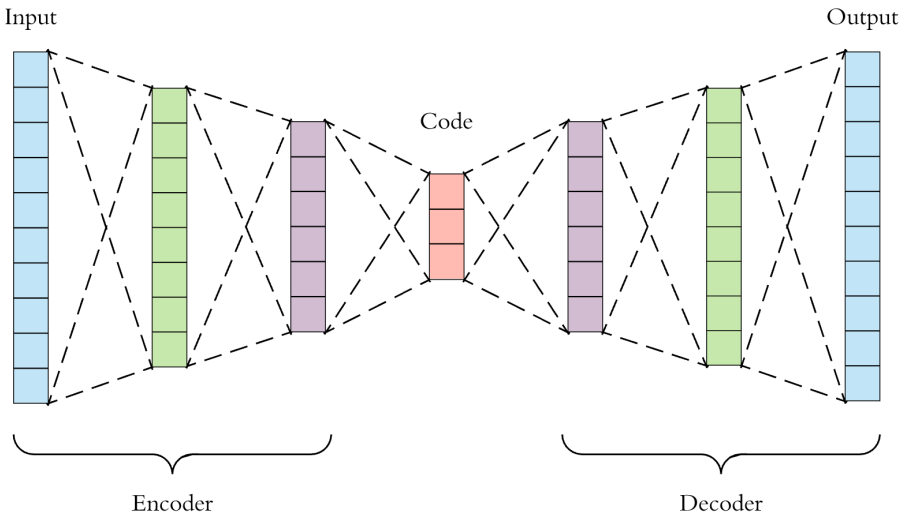


Figura 2.13: Arquitectura de un Autoencoder

Como se puede observar en la imagen 2.13 las dimensiones de entrada son iguales en la salida, y las capas intermedias se pueden dividir en dos partes:

- *Encoder*: permite codificar la entrada a una dimensionalidad menor, de modo que se genera una representación de menor dimensionalidad de la entrada y se generan las nuevas *features*. La capa intermedia se puede utilizar como representación de menor dimensionalidad, similar a la reducción de dimensionalidad del PCA, que puede utilizarse como variables para otro algoritmo.

- *Decoder*: decodificador utilizando la capa intermedia donde se pretende reconstruir los datos de entrada, utilizando las *features* generadas por el *Encoder* hasta llegar a la dimensionalidad de entrada.

Utilizando este método es posible, a partir de datos sin etiquetar, entrenar una red neuronal que permita reconstruir fielmente los datos de entrada, pero en el caso de las anomalías esta reconstrucción no seria del todo fiel (o similar a datos comunes), en otras palabras, el error de la reconstrucción sería mayor que el de una representación normal. De este modo se permite identificar picos en el error que muestren que la reconstrucción no es fiel y que la observación en cuestión se pueda tratar de una anomalía.

En la imagen 2.14 se puede observar un ejemplo del uso de autoencoders que permite clasificar una transacción fraudulenta (clase 1) de una no fraudulenta (clase 0). En este caso la reconstrucción de los datos no fraudulentos representan un error muy pequeño, en comparación a sus contrapartidas, y se puede observar como la mayor parte de estos se encuentran debajo del umbral definido. Aquellos datos anómalos no acaban de reconstruirse con total fidelidad aumentando el error definido (en este caso el error cuadrático medio). No todos son identificados correctamente, pero si permite aumentar la detección de los mismos, en algunos casos donde la cantidad de datos es muy grande y la cantidad de anomalías es muy pequeña este tipo de enfoques puede ayudar a disminuir la cantidad de trabajo necesario para confirmar el fraude, un ejemplo sería en el blanqueo de capitales donde la mayor parte de casos no constituyen blanqueo, reducir el número de casos a revisar puede ahorrar una gran cantidad de costes y tiempo.

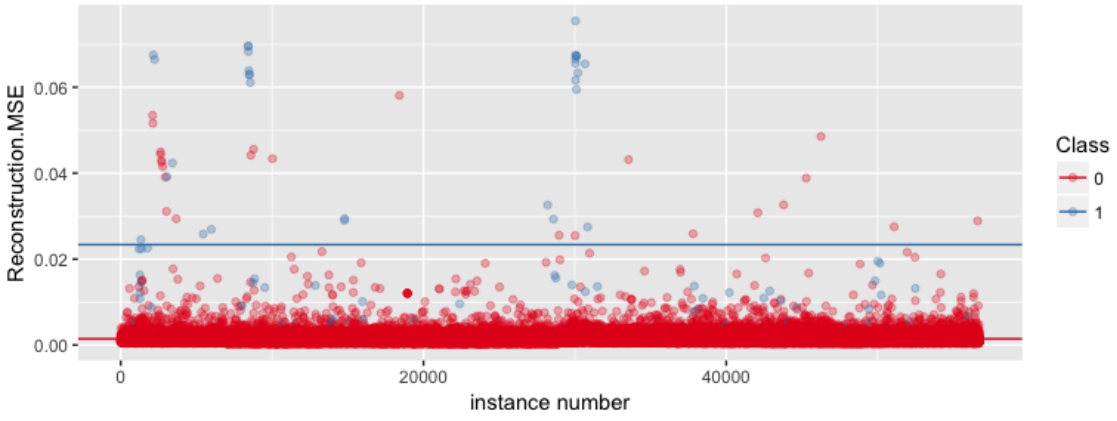


Figura 2.14: Ejemplo de detección de anomalías con Autoencoders

Capítulo 3

Desarrollo

En esta parte de la memoria se detallarán los pasos seguidos durante el desarrollo y la aplicación de los algoritmos de *Machine Learning* sobre los datos utilizados. Asimismo, se profundizará en los datos utilizados y cómo se han preprocesado para su posterior aplicación en los algoritmos.

Generalmente, la mayor parte del tiempo de un proyecto de *Data Science* se invierte en la extracción y preprocesado de los datos, hasta que tengan la suficiente calidad como para arrojar resultados fiables tras aplicar los algoritmos. Este ha sido el caso del presente proyecto, la mayor parte de los esfuerzos han recaído en la comprensión de los datos y en cómo hacer frente (tanto por tiempo como por tamaño) a los datos disponibles.

Tanto el preprocesado como la aplicación de los algoritmos de *Machine Learning* se han realizado utilizando el lenguaje de programación *Python* en su versión 3.6 y la exploración y análisis de los datos se han realizado en *Jupyter Notebooks* para facilitar la visualización de los datos y las acciones tomadas. Todo estos procesos se han realizado en una máquina remota proporcionada por la UOC con 10 núcleos y 64gb de memoria RAM.

3.1. Datos utilizados

Para este proyecto se va a utilizar un *dataset* con información de las conexiones entre dispositivos IoT, mediante el análisis de los paquetes obtenidos por un analizador de paquetes o *sniffer*, un software que permite observar y recopilar las acciones que desarrollan dentro de una red [25]. Utilizando los datos recopilados por este software se preprocesarán de modo que se pueda extraer la información y posteriormente transformarla y crear nuevas variables.

Los datos iniciales fueron preprocesados inicialmente con la información de las anomalías (únicamente anomalías) que se habían recopilado en un espacio de tiempo de dos meses. En este caso los datos solo contaban con tráfico ilegítimo, por lo que no se podía hacer una detección de anomalías como tal.

Posteriormente, se proporcionaron los datos en bruto en un fichero de extensión *pcap* resultante del *sniffer* mencionado anteriormente. Este fichero contiene información de las distintas capas en las que se almacenan los datos, siendo las de más interés las capas *TCP* e *IP*.

- *Transmission Control Protocol*: originario de los años 70/80, consiste en un protocolo para la transmisión de paquetes en conexiones *host* a *host* en redes [26].
- *Internet Protocol*: protocolo que permite la transmisión de datos de origen a destino donde el origen y el destino son direcciones de longitud fija y proporciona los medios para la división y reagrupación de grandes datos en grupos más pequeños [27] .

Sobre estas capas se obtiene la información disponible al más bajo nivel y posteriormente se generarán nuevas variables en base a los datos obtenidos. Siendo los datos de la capa TCP los siguientes:

- Puerto de origen: indica el puerto de origen utilizado para realizar la conexión desde la IP de origen.
- Puerto de destino: indica el puerto de destino utilizado en la conexión entre *hosts*.

Los datos obtenidos de la capa IP:

- IP de origen: dirección con longitud fija del origen de la conexión.
- IP de destino: dirección con longitud fija del destino de la conexión.
- *Length*: longitud/tamaño de los paquetes enviados durante la conexión.

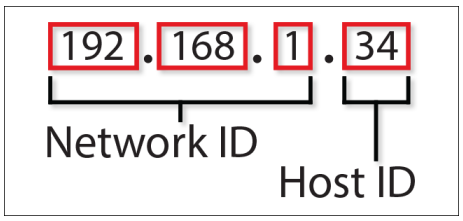


Figura 3.1: Ejemplo de dirección IPv4

Otros datos:

- *Payload*: contenido de los datos transmitidos, pudiendo ser texto, imágenes o comandos a ejecutar en el *host* de destino, con posibilidad de estar cifrado.
- *Timestamp*: marca de tiempo que indica en qué momento se realizó la conexión.

Un ejemplo más visual de los datos a bajo nivel se puede realizar utilizando la herramienta *Open Source, Wireshark*, fundamental a la hora de entender cómo están distribuidos los datos en las capas.

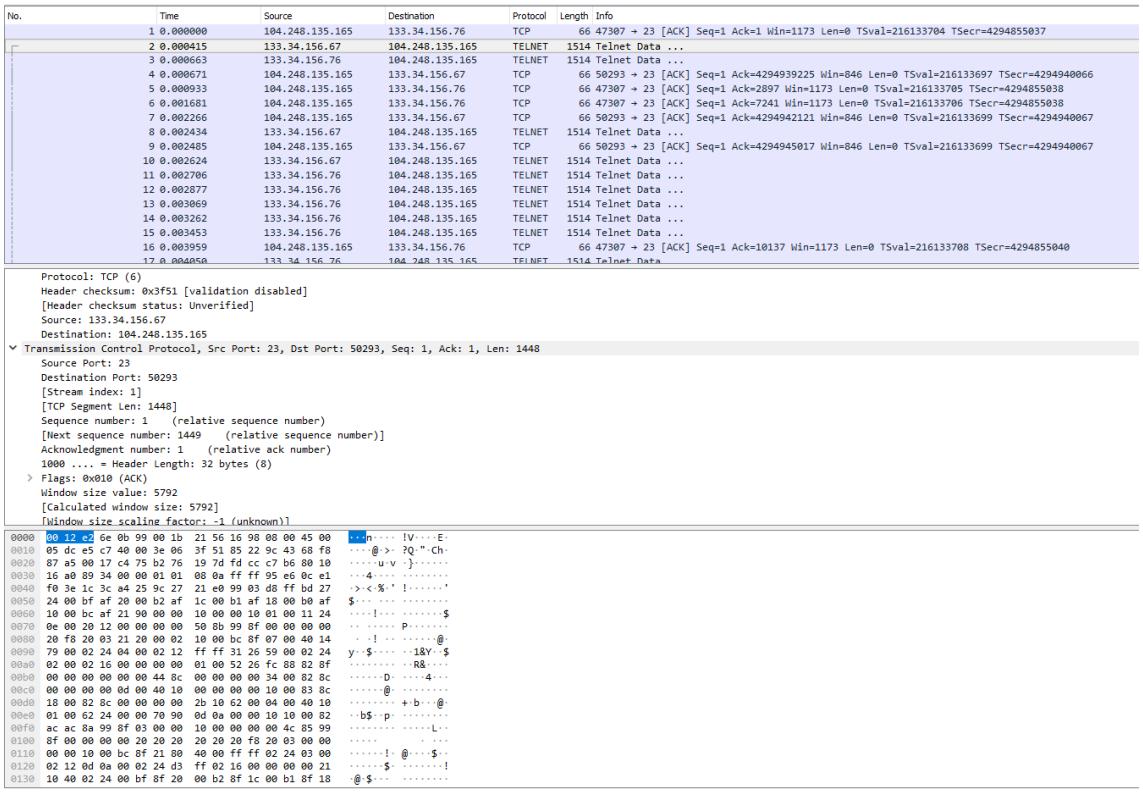


Figura 3.2: Ejemplo de datos pcap en Wireshark

El *Timestamp* puede indicar que el problema debería tratarse como una serie temporal y generar variables en base a esta característica.

Los datos provienen de la recolección de información realizada el 2 de Febrero de 2019, contando tanto con tráfico legítimo como ilegítimo. Sin embargo, no existe clasificación de los datos, por lo que se trata de un problema de carácter no supervisado y como consecuencia se utilizarán técnicas como las descritas en el capítulo 2 de aprendizaje no supervisado.

3.2. Exploración y Preprocesamiento

En este aportado se van a mostrar los pasos seguidos durante la exploración y el procesamiento asicomo la información obtenida durante el proceso. Ambas etapas se realizan de manera conjunta, es decir, la exploración conlleva la realización de distintos preprocesamientos o otras palabras, se realiza *Feature Engineering*. Se puede definir como el proceso de exploración y procesamiento de datos que ayudan a la creación de nuevas variables que puedan mejorar la eficiencia del modelo[28].

El paso inicial era la extracción de los datos del fichero *pcap*, el tamaño total era cercano a los 14 GB, de modo que en el momento de cargar la información en memoria, el uso aumentaba considerablemente dificultando su manipulación, por lo que el primer paso fue investigar si existía alguna forma de dividirlo. El resultado fue el uso de la herramienta *tcpdump*, una línea de comandos utilizada para el análisis de paquetes escrita en C/C++ [29], la misma permitía la división del fichero original en pequeños ficheros con el tamaño deseado. La división se realizó con un tamaño máximo de 500 MB, para permitir la posterior paralelización de los ficheros y evitar un uso muy alto de memoria, dando como resultado 15 ficheros.

Una vez obtenidos los ficheros se deben procesar de modo que el resultado sean datos de forma tabular, que puedan ser explorados y preprocesados con mayor facilidad. Para realizar este proceso se utilizó el módulo de *python*, *scapy*, utilizado comúnmente en tareas de análisis de redes. Gracias a la granularidad de los nuevos ficheros y a utilizar *python* se procesan 9 ficheros simultáneamente utilizando el módulo mencionado, éste se encarga de recorrer las capas mencionadas y extraer los datos y organizarlos de forma tabular, concretamente en *csv*. En ciertos casos, los paquetes pueden contener el *payload*, cuando este es el caso se extrae y se añade como una columna más, para el resto se trata como si fueran *NA* o datos sin registro.

Una vez terminado el proceso se obtienen los primeros datos en brutos de carácter estructurado. Se genera una cantidad análoga de ficheros con un volumen tres veces superior a los datos originales, con las variables mencionadas en la sección anterior. Antes de realizar ningún tipo de exploración se realiza un preprocesado en paralelo para generar nuevas variables sencillas que puedan aportar información de cara a la primera exploración. Las nuevas variables son las siguientes:

- Red 1 IP origen

- Red 2 IP origen
- Red 3 IP origen
- Host IP origen
- Red 1 IP destino
- Red 2 IP destino
- Red 3 IP destino
- Host IP destino
- is_busybox
- is_sh
- is_enable

Las variables en referencia a las IPs se generaron haciendo una división de las direcciones:

$$192{,}168{,}1{,}1 => 192_{red1}\ 168_{red2}\ 1_{red3}\ 1_{host} \tag{3.1}$$

El resto de las variables se obtuvieron mediante el *parseo* del *payload* en aquellos registros que así lo tuvieran y buscando la existencia de las palabras clave *sh, enable y busybox*, siendo la mayor parte de los registros sin contenido en el *payload* o incluso cifrado. La razón por la que se eligieron estas palabras clave es por el uso de este tipo de comandos en los dispositivos IoT por *malwares* conocidos, *busybox* se trata de una versión mínima de un sistema operativo *UNIX* utilizado en pequeños dispositivos y que puede ser explotado por *malware*[30]. De la misma manera *sh* y *enable* son comandos comúnmente utilizados en este tipo de ataques. Las tres variables son de tipo binario, es decir, indica si en el registro existe el comando o no.

El proceso de exploración se realiza en *Jupyter Notebooks* con la ayuda del módulo *matplotlib* y *seaborn* para las visualizaciones. Para esta primera exploración, se van a buscar los medios más utilizados en las distintas conexiones, es decir, las IPs más utilizadas, puertos más utilizados, información de las variables binarias y la evolución de la cantidad de conexiones en el tiempo.

Timestamp	IP1	IP1_0	IP1_1	IP1_2	IP1_3	IP2	IP2_0	IP2_1	IP2_2	IP2_3	Length	Payload	Port1	Port2	is_busybox	i
2019-02-01 09:37:56.421462	71.6.146.185	71	6	146	185	133.34.156.185	133	34	156	185	44	b'\x00\x00'	24858	8649	0	
2019-02-01 09:37:56.446036	185.176.27.114	185	176	27	114	133.34.156.140	133	34	156	140	40	b'\x00 \x00\x00 \x00\x00\x00'	52462	6334	0	
2019-02-01 09:37:56.512569	104.248.37.237	104	248	37	237	133.34.156.78	133	34	156	78	52	NaN	43082	23	0	
2019-02-01 09:37:56.549846	209.141.57.185	209	141	57	185	133.34.156.140	133	34	156	140	60	NaN	32860	2323	0	
2019-02-01 09:37:56.550018	151.73.61.209	151	73	61	209	133.34.156.65	133	34	156	65	40	b'\x00 \x00\x00 \x00\x00\x00'	45431	23	0	

Figura 3.3: Datos en forma tabular

Para realizar la exploración se unen todos los datos contenidos en los ficheros generados ordenados por la marca de tiempo. Una vez cargados, se realizan las operaciones básicas para determinar que no faltan datos, es decir, que en las variables originales (salvo el *payload*) no falta ningún tipo de registro. En este caso no hay existencia de *NAs* en las variables, en otras palabras, todos los datos tienen los registros necesarios.

En la siguiente gráfica se puede observar la cantidad de valores únicos en cada variable, en la que se pueden apreciar dos cosas claras:

- Las variables con más únicos son las de los puertos.
- No hay una gran cantidad de IPs únicas.

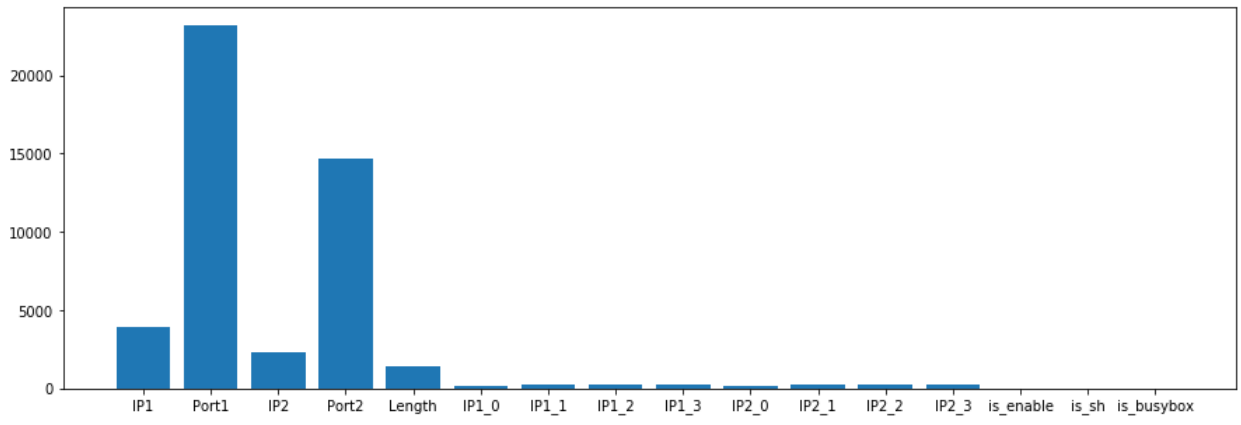


Figura 3.4: Cantidad de valores únicos en las variables

La parte de los puertos en origen (subíndice 1) tiene sentido que sea mayor que en la IP destino, dado que a la hora de iniciar conexiones por lo general el puerto origen es aleatorio y el de destino suele ser fijo, como el puerto 22 para conexiones *ssh*, por lo que puede significar que se estuviera haciendo un escaneo de puertos. Esto se puede ver en el diagrama de barras representando los puertos más utilizados en origen y destino:

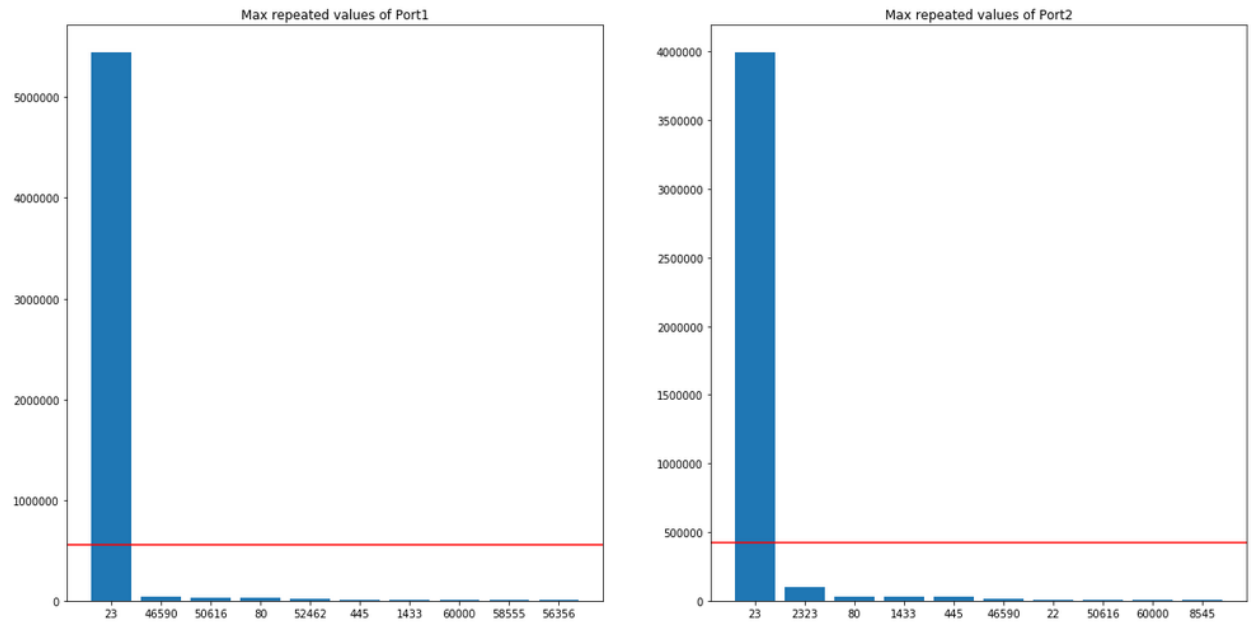


Figura 3.5: Puertos más utilizados

Se puede observar lo siguiente:

- EL puerto más utilizado en ambos casos con diferencia es el puerto 23. Es el puerto utilizado para realizar las conexiones vía *telnet*.
- Seis de los diez puertos más utilizados en la IP de origen son puertos aleatorios, utilizados al realizar la conexión. Los puertos del rango 4xxxx-6xxxx caen en este uso efímero.
- El puerto 80 por lo general es utilizado en conexiones *HTTP* (web).
- El puerto 445 es utilizado para la transferencia de ficheros.
- El puerto 1433 suele utilizarse en conexiones de *Microsoft SQL Server*.
- El 2323 puede ser un puerto alternativo para la conexión vía *telnet*[31].

- El puerto 22 se utiliza para conexiones *ssh*.

- Tanto para destino como para origen se repiten los puertos 46590, 50616 y 60000, cayendo en el rango de puertos temporales.

En el caso de las IPs más utilizadas se puede ver lo siguiente:

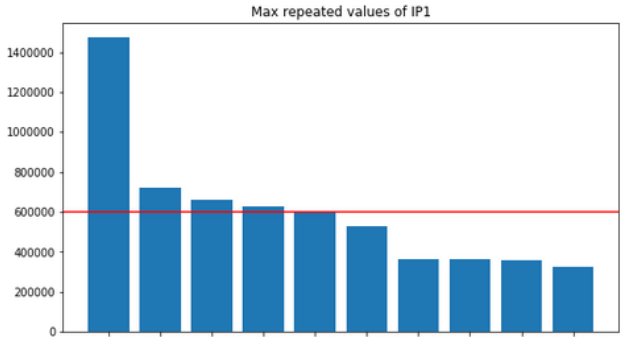


Figura 3.6: IPs de origen con la mayor cantidad de conexiones

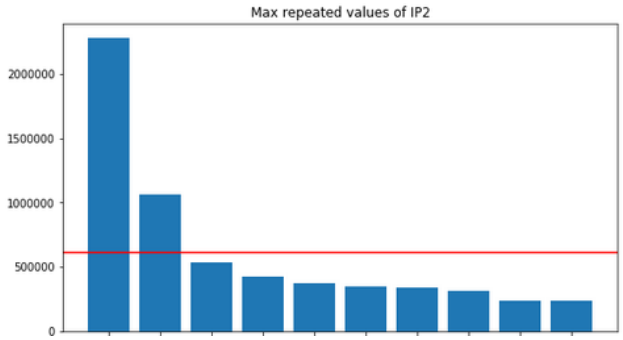


Figura 3.7: IPs de destino con la mayor cantidad de conexiones

Con una cantidad de datos cercana a los 11 millones de registros la mayor parte de las conexiones se registran en estos diez valores máximos con una diferencia de casi el doble de la primera a la segunda tanto para origen como destino.

Dentro de la temporalidad de las conexiones se pueden observar patrones de las conexiones a lo largo del día y su evolución. Debido a la granularidad de los tiempos (capturan con un detalle de microsegundos) estos patrones se han agrupado en distintas ventanas, de modo que en ventanas más pequeñas se puedan detectar ascensos y descensos bruscos, mientras que en las ventanas grandes se puede observar el patrón general. Por ello se han realizado gráficas con ventanas de 1 minuto, 5 minutos y 30 minutos.

Figura 3.8: Evolución de las conexiones a lo largo del tiempo con distintas ventanas

Se observa:

- Un patrón claro de la cantidad de conexiones, comienza a aumentar durante la mañana y la cantidad de peticiones disminuye por la tarde
- En ventanas más pequeñas existe una caída cerca de las 11 horas y un pequeño pico después.
- Se aprecia un pequeño valle a las 14:00, más pronunciado en ventanas pequeñas.
- Gran caída sobre las 18:00.

- En las comparaciones entre ventanas, se aprecia como el patrón es semejante entre ellas.

En cuanto a las variables binarias, se observa que se encuentran muy descompensadas y existen pocas observaciones que las contengan:

	is_busybox	is_enable	is_sh
0	9812548	10114993	9643741
1	358980	56535	527787

Cuadro 3.1: Tabla de frecuencias de las variables binarias

Por último, en la exploración inicial se observo la distribución de la variable *Length*. En la figura 3.9 se puede apreciar como la mayor parte de las conexiones se realizan con una longitud muy pequeña, recayendo la mayor parte de los valores en torno a 40-50 o con una magnitud grande, generalmente en torno a los 1400, mientras que los valores intermedios son minoría.

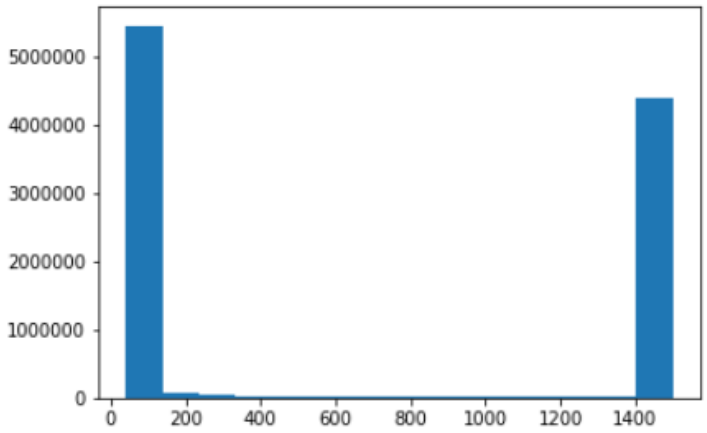


Figura 3.9: Histograma de la variable *Length*

Sobre la variable *Payload* no se realizó ningún tipo de exploración, a parte del parseo, debido a la dificultad de procesar los datos y adaptarlos a un formato más tabular. Se trató de obtener la secuencia de comandos en aquellas observaciones no cifradas sin embargo, no se consiguió obtener esta secuencia que podría haber ayudado en gran medida aportando una gran cantidad de información. Algnas de las variables que se podrían haber obtenido son: cantidad de comandos en secuencia, longitud media de los comandos o incluso técnicas de NLP como *Tf-idf (Term frequency – Inverse document frequency)*.

Con la información obtenida y con un mayor conocimiento de los datos disponibles, se propusieron nuevas variables que podrían aportar más información y podrían ser más útiles a los modelos (en este momento aún por decidir). Tras varios intentos se generaron las siguientes variables:

- *IP1_time_diff_per_value*: tiempo en segundos, de la diferencia de tiempos desde la anterior conexión y la conexión actual de la IP de origen.
- *IP2_time_diff_per_value*: tiempo en segundos, de la diferencia de tiempos desde la anterior conexión y la conexión actual de la IP de destino.
- *IP1_IP2_time_diff_per_value*: tiempo en segundos, de la diferencia de tiempos desde la anterior conexión y la conexión actual entre IPs únicas de origen y destino.
- *Port1_last_1*: cantidad de puertos únicos utilizados por la IP de origen en una ventana de un minuto
- *Port2_last_1*: cantidad de puertos únicos utilizados por la IP de destino en una ventana de un minuto

Todas estas variables se generan en especifico para los valores únicos de las IPs, es decir, cuando se utiliza la IP de origen, se itera entre todos los valores únicos y por cada valor se genera la ventana o la diferencia de tiempo. Con esto se consigue que las nuevas variables solo estén relacionadas con las mismas IPs, tanto de origen, destino o para el caso de *IP1_IP2_time_diff_per_value* de ambas y que no dependan de conexiones entre distintas IPs.

Con las nuevas variables se pretende conseguir distintos tipos de información, con la diferencia de tiempo en la IP de origen, se pretende observar si se está utilizando asiduamente, esta teniendo un pico de conexiones o si por el contrario las establece raramente. Por otro lado, para la IP de destino se pretende observar si esta siendo objetivo de muchas conexiones en cortos periodos de tiempo, que podrían significar ataques *DDoS (Distributed Denial of Service)*. Para la diferencia entre las conexiones únicas de IP se pretende obtener si se pueden estar realizando escaneos de puertos sobre la IP de destino. En el caso de las variables de puertos, su finción es obtener información más precisa de si puede estar realizando un escaneo o existen IPs de origen con muchas conexiones simultáneas que observan distintos destinos.

Una vez generadas estas variables, se necesita observar la cercanía (o distancia) entre las redes, puesto que es muy común que una vez se infecte un dispositivo, éste intente infectar

aquellos dispositivos que se encuentren en su misma red o una red cercana. Inicialmente se planteo utilizar la IP como variable numérica y obtener la distancia euclídea entre la IP origen y destino, sin embargo, no existe ningún tipo de relación entre ellas de esta manera, es decir, la IP 1 y 2 no están a la misma distancia que las IPs 230 y 231. Aprovechando las variables generadas de las IPs con las subredes y el host, se podría calcular la distancia entre ellas, sin embargo, en este caso tomaría como equivalente la distancia entre hosts y la distancia entre redes principales sin embargo, redes con distinto valor no están relacionadas mientras que si la única diferencia es en el host, estos se encuentran en la misma red.

$$dist(IP_{red1}, IP_{red2}) \neq dist(IP_{host1}, IP_{host2}) \tag{3.2}$$

Para ello se tomó la IP como un espacio 4-dimensional (red1, red2, red3 y host) con valores ponderados, 1000 para la primera red, 100 para la segunda, 10 para la tercera y uno para el host.

$$IP_x = (x_1, x_2, x_3, x_4) \tag{3.3}$$

$$dist(IP_x, IP_y) = \sqrt{1000 * (x_1 - y_1)^2 + 100 * (x_2 - y_2)^2 + 10 * (x_3 - y_3)^2 + (x_4 - y_4)^2} \tag{3.4}$$

Con esto una diferencia de uno en la red principal, daría una distancia de $\sqrt{1000} \approx 32$.

Tras realizar pruebas con este tipo de distancia, se decidió utilizar otro enfoque mejorando los resultados. En lugar de tomar la IP como un valor 4-dimensional, se tomo como 32-dimensional, es decir, una dimensión por cada bit.

$$IP = 192{,}128{,}1{,}1 = 11000000100000000000000100000001 \tag{3.5}$$

Para dar aún más peso a los valores iniciales y la cercanía a nivel de red se utilizo el operador *XOR*, devolviendo un 1 para cuando existe diferencia entre los bits de la misma posición o 0 si el valor es igual. Una vez aplicado el operador, se cuentan el número de ceros por la izquierda, en otras palabras, el número de bits iniciales (redes principales) que coinciden, al momento que aparece una diferencia se para de contar. El número resultante se le resta a 32 y se divide entre

32 para normalizar el valor entre 1 y 0, siendo el uno redes completamente distintas y el cero la misma red y el mismo host. Podemos ver el siguiente ejemplo entre IPs que solo se diferencia en el host:

$$IP_1 = 192{,}128{,}1{,}1 = 11000000100000000000000100000001$$
$$IP_2 = 192{,}128{,}1{,}2 = 11000000100000000000000100000010$$
$$XOR(IP_1, IP_2) = 00000000000000000000000000000010$$
$$n_{zeros} = 31$$
$$dist(IP_1, IP_2) = \frac{(32 - n_{zeros})}{32} = \frac{1}{32}$$

En el caso de diferencias entre redes:

$$IP_3 = 255{,}128{,}1{,}1 = 11111111100000000000000100000010$$
$$XOR(IP_1, IP_3) = 00111111100000000000000000000000$$
$$n_{zeros} = 2$$
$$dist(IP_1, IP_3) = \frac{(32 - n_{zeros})}{32} = \frac{30}{32}$$

Como se puede observar la distancia aumenta en gran medida con las redes y en muy poca con los host, aportando más valor como nueva variable que las distancias utilizadas inicialmente.

Una creadas las nuevas variables se realiza otra exploración con el fin de identificar si pueden ser útiles y si es así como tratarlas de cara a la detección de anomalías. Siendo uno de los primeros pasos la correlación entre las variables visualizándolo todo en un *heatmap*:

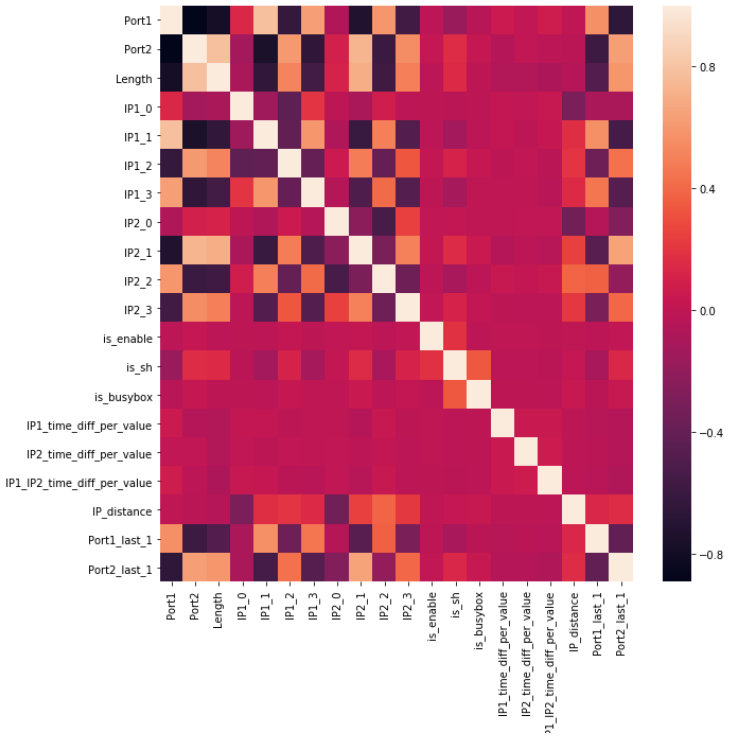


Figura 3.10: Mapa de correlaciones

Se pueden observar lo siguiente:

- Correlación entre los puertos negativa, posiblemente por la asignación de puertos por encima de 40000 en el inicio de las conexiones.
- A mayor número de puerto menor *Length*.
- No existen correlaciones entre las redes, subredes y hosts de las IPs.
- Parece que a mayor puerto de destino, mayor *Length*.
- Las distintas variables de diferencia de tiempo parecen no estar correladas entre sí.
- La cantidad de puertos en el último minuto están negativamente correladas entre ellas.

Para las diferencias de tiempo se observan que la mayor parte se encuentran en valores muy pequeños, inferiores a un segundo en la IP de origen ,3.11 3.12. Este comportamiento se repite también para la IP destino, pero ésta contiene más valores dentro del rango inferior a un segundo que la IP de origen 3.13 3.14. En el caso de la diferencia conjunta, el patrón es muy

similar a la IP de destino pero con una cantidad menor de observaciones, en otras palabras, hay IPs únicas que solo se han conectado una vez y no tienen diferencia de tiempo 3.15 3.16.

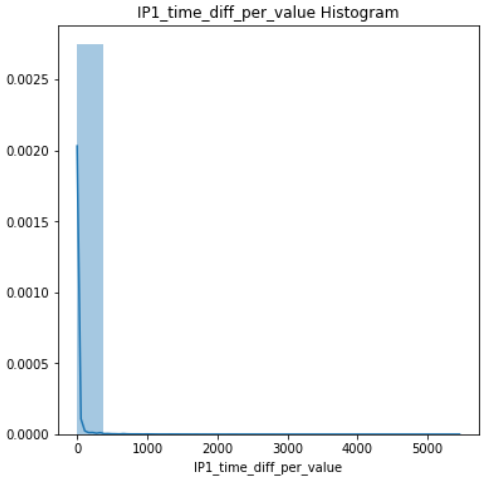


Figura 3.11: Distribución total de *IP1_time_diff*

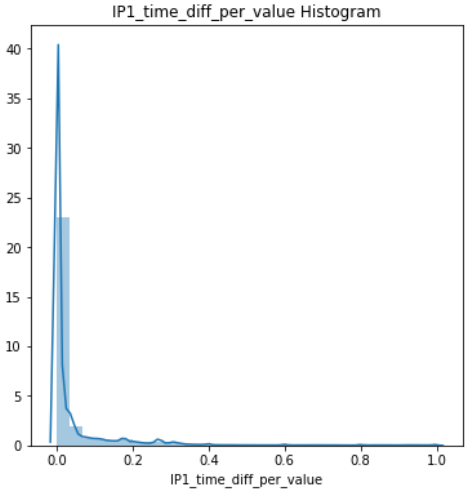


Figura 3.12: Distribución para valores menores a un segundo de *IP1_time_diff*

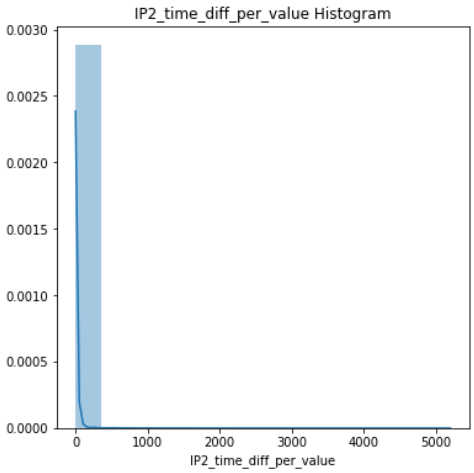


Figura 3.13: Distribución total de *IP2_time_diff*

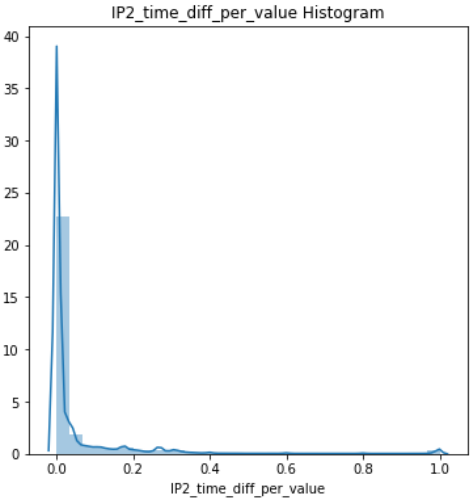


Figura 3.14: Distribución para valores menores a un segundo de *IP2_time_diff*

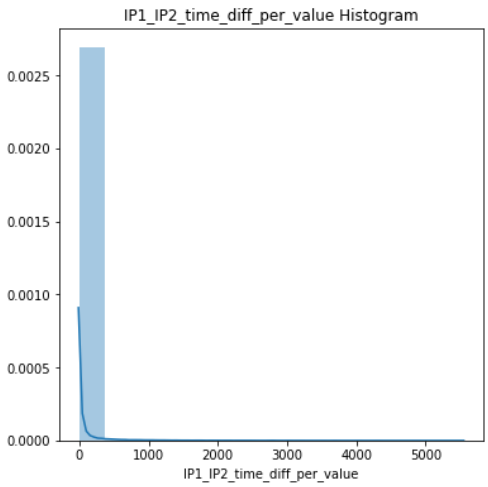


Figura 3.15: Distribución total de *IP1_IP2_time_diff*

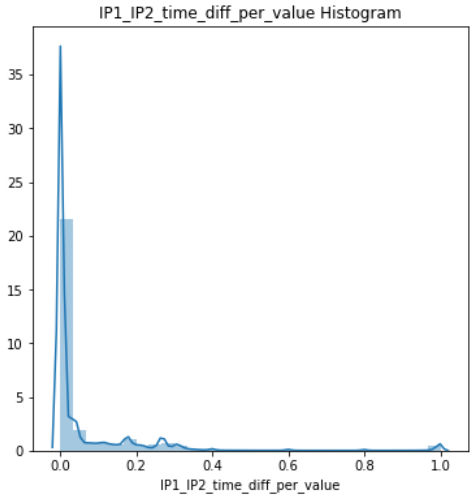


Figura 3.16: Distribución para valores menores a un segundo de *IP1_IP2_time_diff*

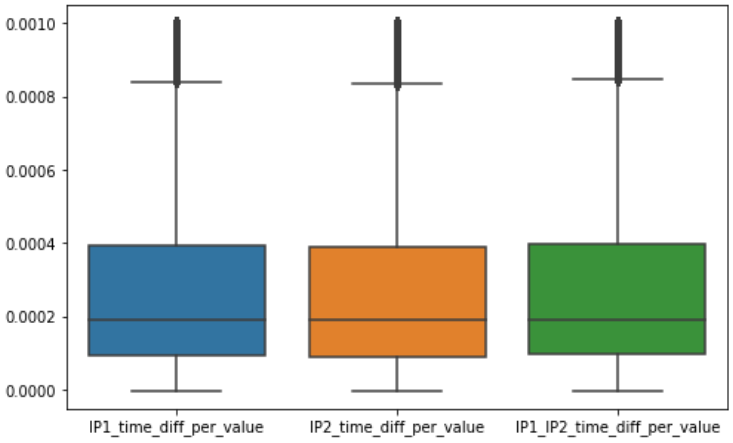


Figura 3.17: Digrama de cajas según diferencia de tiempo

Otra variable a tomar en cuenta es la distancia entre IPs. Dentro de la exploración (3.18) se ha observado como la gran mayoría de las IPs provienen de redes completamente distintas, muy pocas IPs comparten la primera red y solo unas pocas observaciones comparten más que los primeros 8 dígitos.

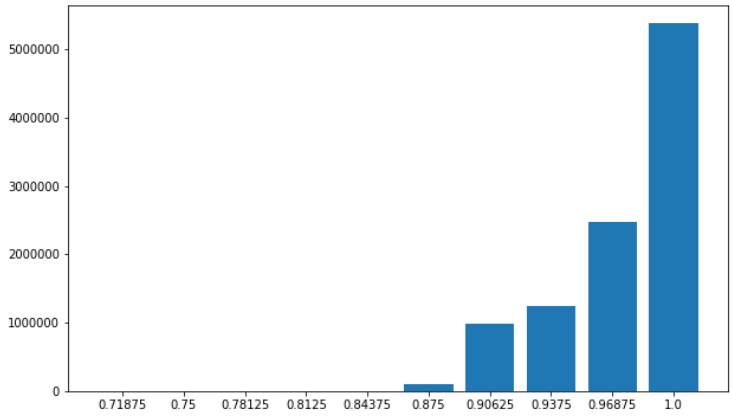


Figura 3.18: Diagrama de barras de la distancia entre IPs

En cuanto a los puertos y al número de puerto utilizados en el último minuto aparecen cosas interesantes:

- La mayor parte de los registros han utilizado de 2 a 7 puertos distintos en el último minuto, tanto para destino como origen.
- La distribución muestra que hay una gran cantidad de registros que han utilizado de 30 a 50 puertos en el último minuto.
- La cantidad de puertos en el último minuto en origen contiene registros con un gran número de puertos.

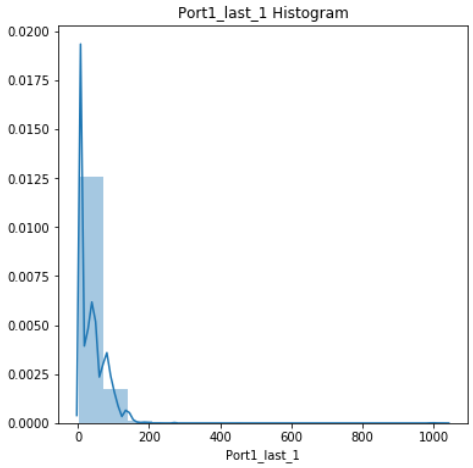


Figura 3.19: Distribución de número de puertos en el último minuto de origen

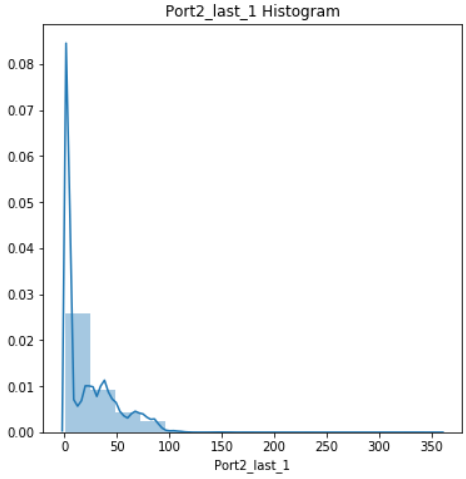


Figura 3.20: Distribución de número de puertos en el último minuto de destino

- Los puertos están más distribuidos en destino.
- Los puertos en origen están más centrados en valores bajos
- Se pueden apreciar patrones de conexión cruzando los puertos

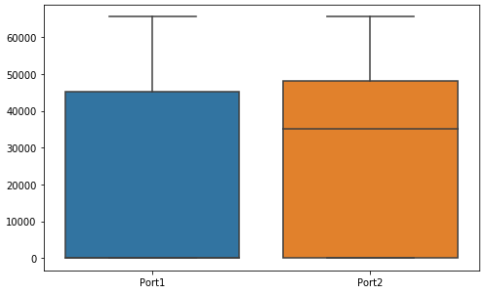


Figura 3.21: Diagramas de caja de los puertos en origen y destino

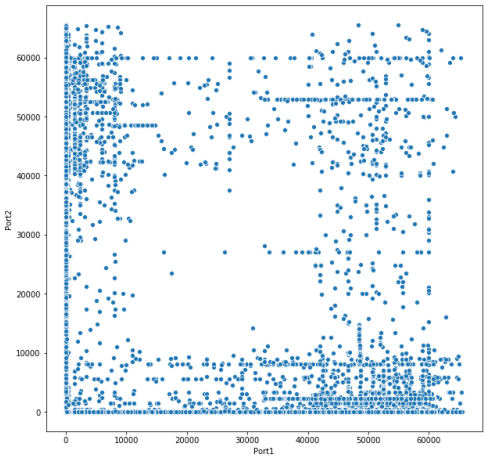


Figura 3.22: Gráfica de puntos entre puertos de origen y destino

Utilizando las variables binarias como ejes, algunas demuestran cierta diferencia en la dispersión, como por ejemplo en las variables de número de puertos.

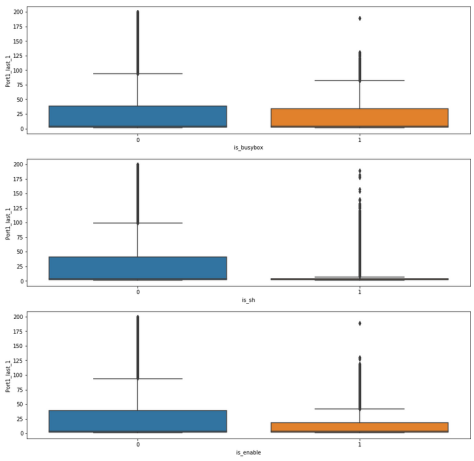


Figura 3.23: Diagramas de caja de número de puertos origen según variables binarias

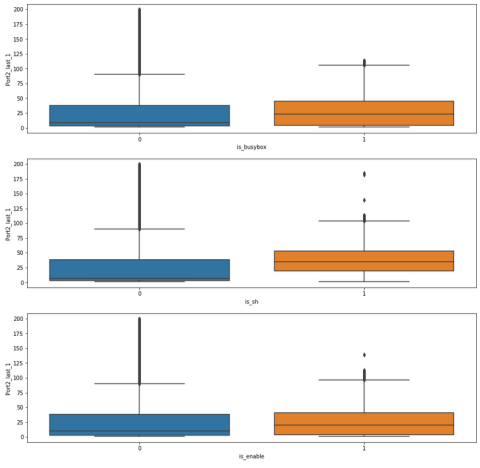


Figura 3.24: Diagramas de caja de número de puertos destino según variables binarias

3.3. Creación del set de datos final

Tal y como se ha comentado anteriormente, por lo general las conexiones realizadas en momentos anteriores no tienen por qué estar relacionadas entre sí, por lo que utilizar la marca de tiempo como índice de los datos para alimentar el modelo no acabaría siendo útil del todo. Por ello, se busca la forma de poder mantener una temporalidad dentro de las variables pero que las observaciones estén relacionadas según las conexiones de origen y destino.

La vía utilizada para generar este set de datos fue basarse en tomar las IPs como índice, es decir, una observación (fila) corresponde a una IP en la que las variables (columnas) tienen la temporalidad. Para ello se establecieron tres ventanas de tiempo distintos, 5 minutos, 10 minutos y 30 minutos, donde cada variable inicial genera nuevas variables como la media, suma, desviación típica y número de registros dentro del periodo total de conexiones (día dos de Febrero).

$$n_{agregaciones} = 4$$
$$n_{periodo} = \frac{T_{fin} - T_{ini}}{ventana}$$
$$n_{porvariable} = n_{agregaciones} * n_{periodo}$$
$$total\_columnas = n_{porvariable} * num\_variables$$

Según el tamaño de la ventana el *dataset* resultante tendrá más o menos tamaño, debido a que una menor ventana implica la creación de más periodos en las variables. Por otro lado, no todas las conexiones tienen observaciones en los datos iniciales, es decir, hay conexiones que se pueden dar por la tarde y no por la mañana, para estas conexiones se rellena el valor a cero, por lo que si se observa la cantidad de ceros, se vera que se trata de una matriz dispersa. Un pequeño ejemplo del dataset resultante sería el siguiente:

El *dataset* tendrá las siguientes características (algunas visibles en 3.25):

- Las filas están formadas por las IPs.
- Se utilizan como índice las IPs únicas de origen y las IPs únicas de destino.
- Se diferencian mediante una variable binaria.

	Port1									
Timestamp	2019-02-01 09:37:00	2019-02-01 09:38:00	2019-02-01 09:39:00	2019-02-01 09:40:00	2019-02-01 09:41:00	2019-02-01 09:42:00	2019-02-01 09:43:00	2019-02-01 09:44:00	2019-02-01 09:45:00	2019-02-01 09:46:00
IP2										
1.168.147.99	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.168.166.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.169.143.118	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.169.39.203	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	81.0
1.174.134.13	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 840 columns

Figura 3.25: Ejemplo de *dataset* final

- La temporalidad se traslada a las columnas.
- Cada variable original se transforma en agregaciones durante el periodo total.
- La dimensionalidad del *dataset* queda definida por la ventana de tiempo escogida.
- Una gran parte de los datos son ceros.
- La cantidad de filas es fija, dado que la cantidad única de IPs es constante independientemente de la ventana de tiempo.

Asimismo, sobre estas variables se podrían seguir generando nuevas variables utilizando distintas ventanas (*rolling window*) sin embargo, para evitar aumentar el tamaño no se crearon nuevas ventanas. Por otro lado, se añadieron nuevas variables binarias indicando el valor de los últimos 8 bits, es decir, el host de las IPs.

Se crearon varios datasets con el fin de identificar anomalías con distintas características, algunas pueden provenir de acciones muy rápidas en un tiempo pequeño que pueden ser detectadas por una ventana pequeña y aquellas acciones que se realicen periódicamente puedan ser detectadas por ventanas más largas.

3.4. Algoritmos de *Machine Learning*

Tras el estudio del estado del arte y las pruebas realizadas durante el preprocesamiento y la creación del *dataset* la técnica utilizada es un ensamblado de dos algoritmos:

- *Isolation Forest*
- *Autoencoder*

La decisión de realizar un ensamblado de varios modelos recae en la capacidad que otorga para generalizar mejor a la hora de realizar las detecciones y así evitar que ciertas observaciones puedan afectar en mayor medida al modelo.

3.4.1. *Isolation Forest*

Como su nombre indica se trata de un modelo basado en árboles de decisión enfocado en el aislamiento de las observaciones. Este tipo de modelo funciona muy bien con *datasets* de alta dimensionalidad y ofrece un rendimiento muy bueno dada su poca complejidad computacional y su bajo uso de memoria [32].

Mientras que la mayor parte de herramientas de detección de anomalías se basan en perfilar el uso normal de los casos y clasificar como anomalía aquello que diverja de este perfil, por ejemplo, una *clusterización* podría indicar que un grupo no es anómalo debido al tamaño del mismo, además de hacer un uso mucho mayor de memoria debido al cálculo de las disimilitudes (o distancias). El *Isolation Forest* se basa en otra metodología, donde en lugar de perfilar, pretende encontrar aquellas observaciones que más aisladas estén. Para ello se basa en dos premisas:

- Las anomalías tienen atributos muy diferentes de los valores normales.
- La cantidad de anomalías presenta un porcentaje muy pequeño dentro de los datos.

En otras palabras, las anomalías son pocas y diferentes, por lo que son más propensas a estar aisladas que las observaciones normales. Este aislamiento se realiza mediante la creación

de arboles de decisión, donde las anomalías necesitarán menos particiones para aislarse que los datos normales.

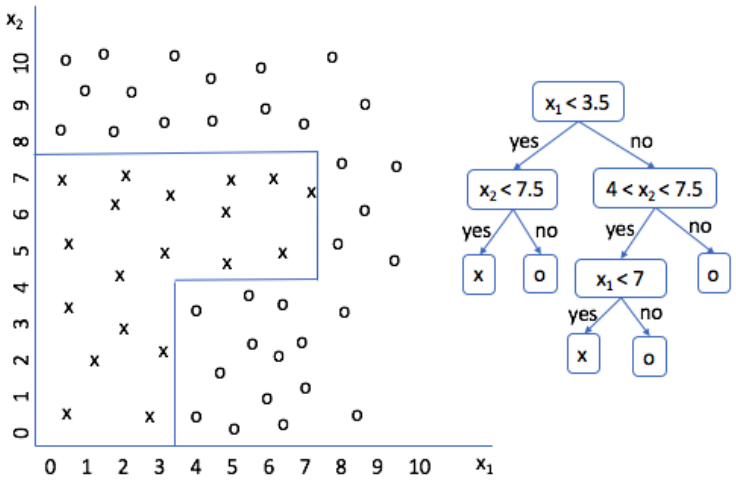


Figura 3.26: Fronteras en un árbol de decisión

La medida utilizada en el *Isolation Forest* es la profundidad del árbol o *path length*, por ejemplo la figura 3.26 tendría una profundidad de tres. Cada árbol realiza particiones aleatorias de modo que cada observación puede formar parte de varios árboles con distintas profundidades, por lo que el *path length* final es la media de todos ellos. En la siguiente figura 3.27 se observa como la primera observación ha necesitado de más particiones (más profundidad del árbol) para poder aislar la observación que el segundo caso, que ha necesitado de muchas menos, por ello este segundo punto tendrá una valoración más alta como anomalía.

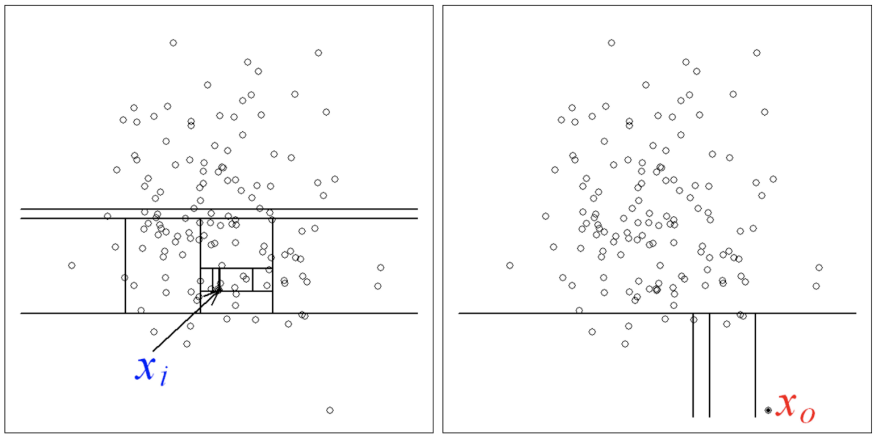


Figura 3.27: Ejemplo de un árbol de *Isolation Forest*

Para la categorización como anomalía se define una puntuación que ayude a determinar que puntos pueden ser anómalos y cuales no. Tomando la profundidad de un árbol como $h(x)$ y la

media de profundidad como $E(h(x))$ se puede definir una puntuación utilizando la profundidad estimada media $c(n)$:

$$c(n) = 2H(n-1) - (2(n-1)/n)$$
$$H(i) = ln(i) + 0{,}5772156649$$
$$s(x,n) = 2^{-\frac{E(h(x))}{c(n)}}$$

Donde n es el número de observaciones totales, x una observación y $s(x,n)$ la puntuación de anomalía. Los resultados de la puntuación se pueden interpretar de la siguiente manera según el cociente de $c(n)$ y $E(h(x))$:

- Un valor cercano a uno, indica que se trato posiblemente de una anomalía.
- Un valor cercano a cero, indica que no es una anomalía.
- Si todas las observaciones devuelven un valor cercano a 0.5 se puede considerar que no existen anomalías.

Además de esta puntuación, para favorecer a la detección de anomalías, *Isolation Forest* utiliza técnicas de submuestreo o *subsampling* que favorece la identificación de las anomalías y disminuye el coste computacional.

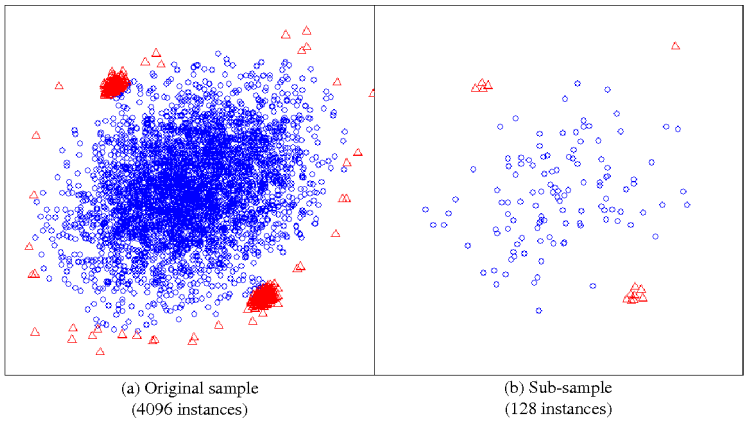


Figura 3.28: Submuestreo aplicado en *Isolation Forest*

3.4.2. *Autoencoder*

El funcionamiento y características generales de los *Autoencoders* se ha explicado en la sección 2.2.2.3, por lo que en este apartado se explicará la arquitectura escogida.

Para este caso se ha construido un *Deep Autoencoder*, es decir, varias capas ocultas que dependen de la cantidad de columnas de los datos de entrada, pero proporcionalmente mantiene la estructura. Consta de una capa de entrada, cinco capas intermedias (dos de *Encoder*, dos de *Decoder* y la representación de menor dimensionalidad) y una capa de salida.

Las dimensiones de las capas intermedias están definidas de la siguiente forma:

- $Encoder_1$ *y* $Decoder_1$: $\frac{ncol}{5}$
- $Encoder_2$ *y* $Decoder_2$: $\frac{ncol}{10}$
- Representación: $\frac{ncol}{25}$

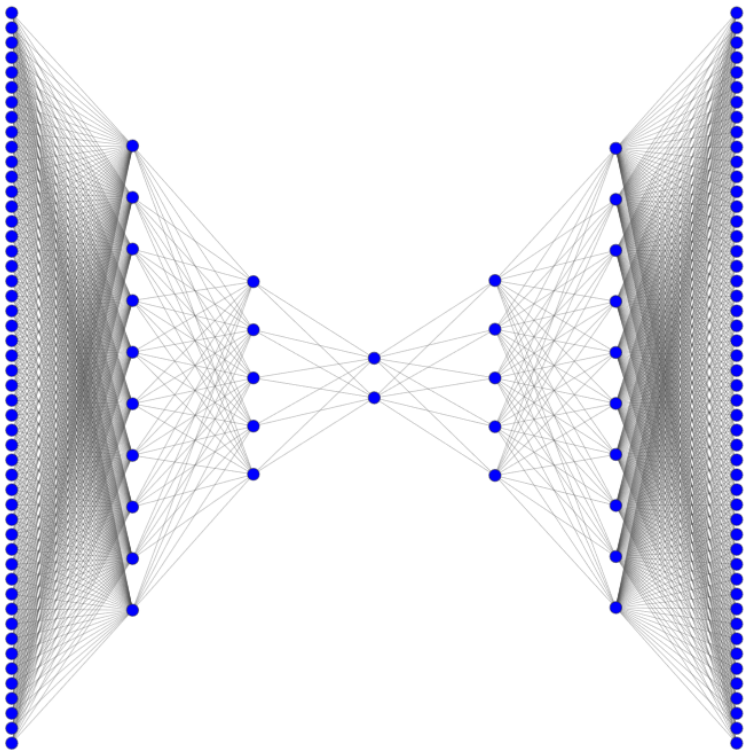


Figura 3.29: Arquitectura básica del *Autoencoder*

Para la arquitectura anterior 3.29 se considera un *dataset* con 50 columnas, por lo que la primera y penúltima capa tendrán 10 neuronas, las siguientes 5 neuronas y la capa central 2. El

número de neuronas es proporcional al tamaño, por lo que crecerá acorde, redondeando hacia arriba para las divisiones no exactas. Las funciones de activación para las capas intermedias se trata de la activación *reLu*, una función a tramos donde los valores cero o menores son cero y los mayores de cero mantienen el mismo valor. Para la de activación de la última capa se utiliza la tangente hiperbólica:

$$S(x) = \frac{(e^x - e^- x)}{(e^x - e^- x)} \tag{3.6}$$

Dado que el entrenamiento depende de la métrica de reconstrucción, esta se ha escogido acorde al tipo de datos con los que se entrena la red neuronal. Debido a las ventanas generadas y las agregaciones el tipo de variables resultantes son numéricas y en muy poca proporción por las variables de host hay 8 binarias, por ello la función de reconstrucción (o de pérdida para la red neuronal) utilizada es el error cuadrático medio 3.7.

$$MSE = \frac{1}{n}\sum_{t=1}^{n} X_i - \hat{X} \tag{3.7}$$

Utilizando esta métrica, la red neuronal buscará disminuirlo lo máximo posible este error, aprendiendo de conexiones legítimas y anómalas. Aunque la red neuronal intente optimizar el error de estas últimas, se considera que la mayor parte de de las conexiones no son anómalas, por lo tanto la optimización en las anomalías no cobran tanto peso durante el entrenamiento y permiten diferenciarse del resto de puntos por el error de reconstrucción.

3.4.3. Ensamblado de modelos

Para el ensamblado de modelos se ha tomado como modelo principal el *Isolation Forest*, esto se debe a que el *Autoencoder* necesita un umbral a la hora de definir cuales son las anomalías, es decir, el valor del error de reconstrucción mínimo para considerarse una anomalía.

Para el *Isolation Forest* se ha utilizado la implementación de *Scikit-Learn* donde se pueden definir distintos hiperparámetros, siendo los más comunes en número de estimadores o número de árboles, la cantidad de observaciones en el muestreo y un factor de contaminación, que define cual es la cantidad aproximada de anomalías. Tras distintas pruebas los hiperparámetros fueron los siguientes:

- *n_estimators*: 500.
- *contamination*: 0.01 (se considera que las anomalías forman el 1 % del total).
- *min_samples*: se utiliza el valor por defecto de 256.

Como entrada al modelo se utilizan los datos finales aplicando una normalización estándar, es decir, se resta la media a todas las observaciones y se divide entre la desviación típica. Por otro lado, los parámetros definidos el modelo encuentra alrededor de un 1 % de anomalías dentro de los *datasets*, la cantidad de anomalías, n , encontradas se utiliza como umbral para el *Autoencoder* rescatando las n observaciones con mayor error de reconstrucción, de ahí utilizar el *Isolation Forest* como modelo principal para definir el posterior umbral.

Para comprobar que tal se está efectuando se realiza una gráfica 3D utilizando los tres primeros componentes principales de un PCA y se utilizan las etiquetas de salida para diferenciar las observaciones. En la gráfica 3.30 se aprecia como las observaciones de color morado (anomalías) se encuentran esparcidas dentro del espacio mientras que existen puntos de gran concentración de color amarillo (no anómalo) con el resto de observaciones. De esta manera se puede comprobar que existen las premisas tomadas por el modelo, donde las anomalías están aisladas y son diferentes al resto.

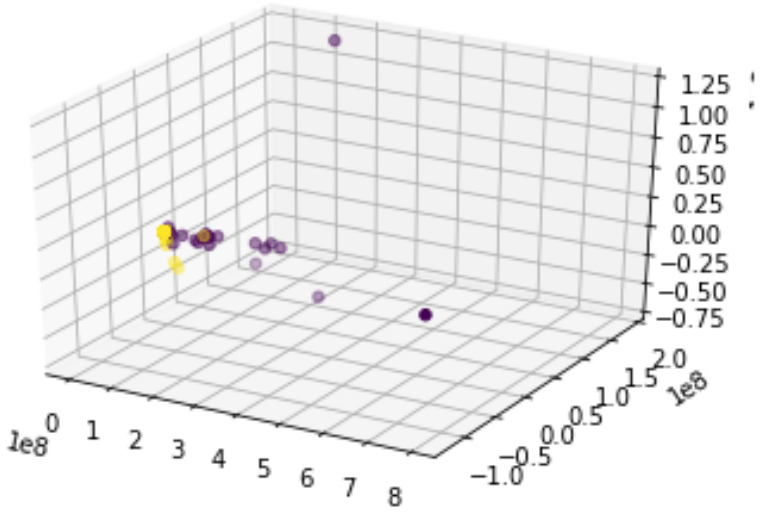


Figura 3.30: Representación de los tres componentes principales y anomalías del *Isolation Forest*

Una vez definidas las anomalías iniciales, se entrena el *Autoencoder* con los siguientes parámetros:

- *optimizer*: *adam* (se probó con *SGD* y *adadelta* con peores resultados).
- *loss*: error cuadratico médio (*MSE*).
- *epochs* (o épocas): el valor de 30 demostró ser el óptimo para los distintos *datasets*.
- *batch_size*: 256.

Una vez entrenado se vuelven a pasar todos los datos por la red y se generan los datos reconstruidos, con estos se calcula el error con los datos originales dando lugar a la siguiente gráfica:

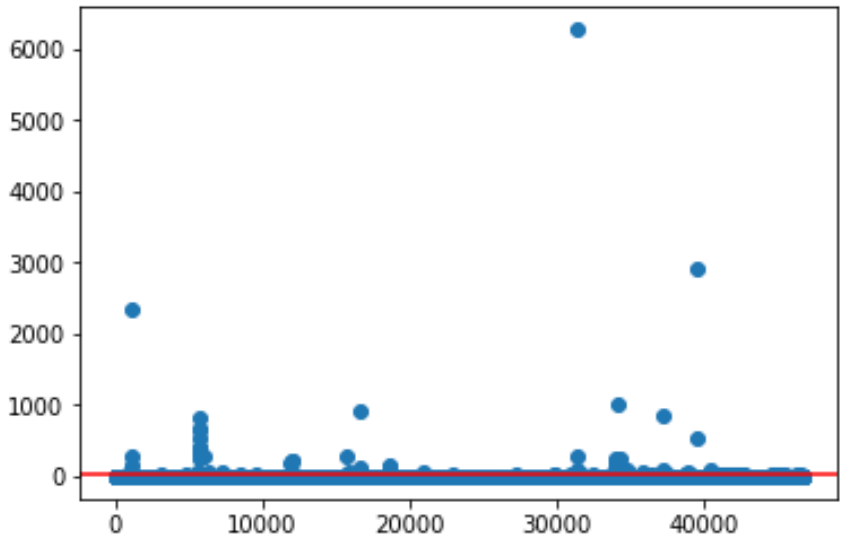


Figura 3.31: Error de reconstrucción

La línea roja muestra el valor medio del error de reconstrucción y muestra la gran densidad que existe en torno a este valor y como existen varias observaciones que se encuentran muy separadas de la esta zona. El umbral es el que permite clasificar que se considera como anomalía o no, pero en este caso no se utiliza el umbral si no las n que más error contengan. Como con el modelo anterior se observa la clasificación utilizando los tres componentes principales 3.32.

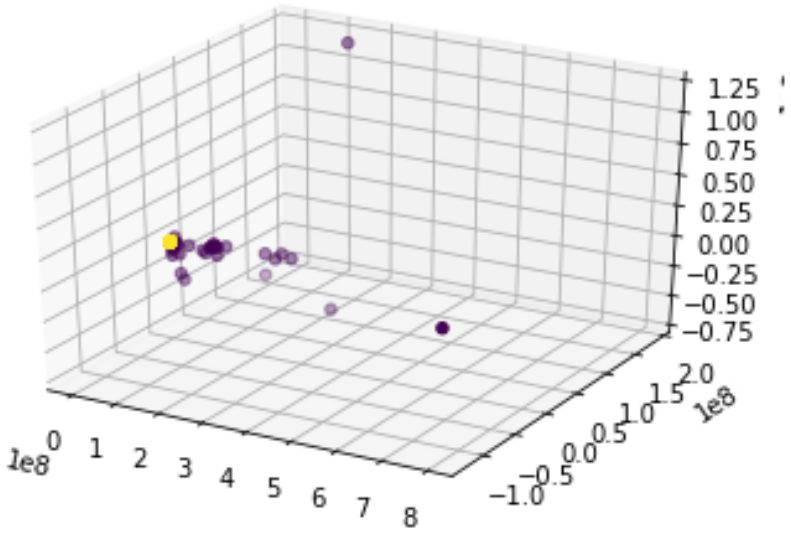


Figura 3.32: Representación de los tres componentes principales y anomalías del *Autoencoder*

Una vez definidas las distintas anomalías por los modelos se realiza un cruce y se seleccionan las anomalías que se hayan detectado en ambos, coincidiendo cerca de un 75 %.

Capítulo 4

Resultados

Una vez detectadas las anomalías se generan un fichero con solo las anomalías detectadas para cada ventana de tiempo. Sobre un a cantidad total de 46713 conexiones formadas tanto por conexiones de origen y destino, las anomalías detectadas por cada ventana son las siguientes:

	30 min	10 min	5 min
Columnas	1102	3202	6310
Anomalías	368	387	396

Cuadro 4.1: Anomalías por ventana de tiempo

Haciendo una comparación entre las distintas anomalías y las distintas ventanas se encuentra que:

- La cantidad de anomalías coincidentes entre las ventanas de 30 y 5 minutos es de 362
- La cantidad de anomalías coincidentes entre las ventanas de 10 y 5 minutos es de 383
- La cantidad coincidente entre todas las ventanas es de 360
- La evaluación principal se desarrollará sobre la ventana de 5 minutos, al contener la mayoría de anomalías

La evaluación y la comparación de resultados se dificulta al no tener un conjunto de datos etiquetados con el que comparar, por lo que se realiza un análisis sobre las variables comparando

los valores anómalos con un muestreo de un 20 % de los datos originales no anómalos, dividido en conexiones origen y destino. La comparación se realiza visualmente evaluando la evolución de las conexiones según la variable a lo largo del periodo, estas visualizaciones no se van a mostrar en este capítulo de la memoria debido a longitud y cantidad de visualizaciones, pero se encuentra disponible en el repositorio asociado al trabajo o en el anexo **??**. Para cada variable se analizan las agregaciones por separado. Un ejemplo de las visualizaciones utilizadas es la siguiente:

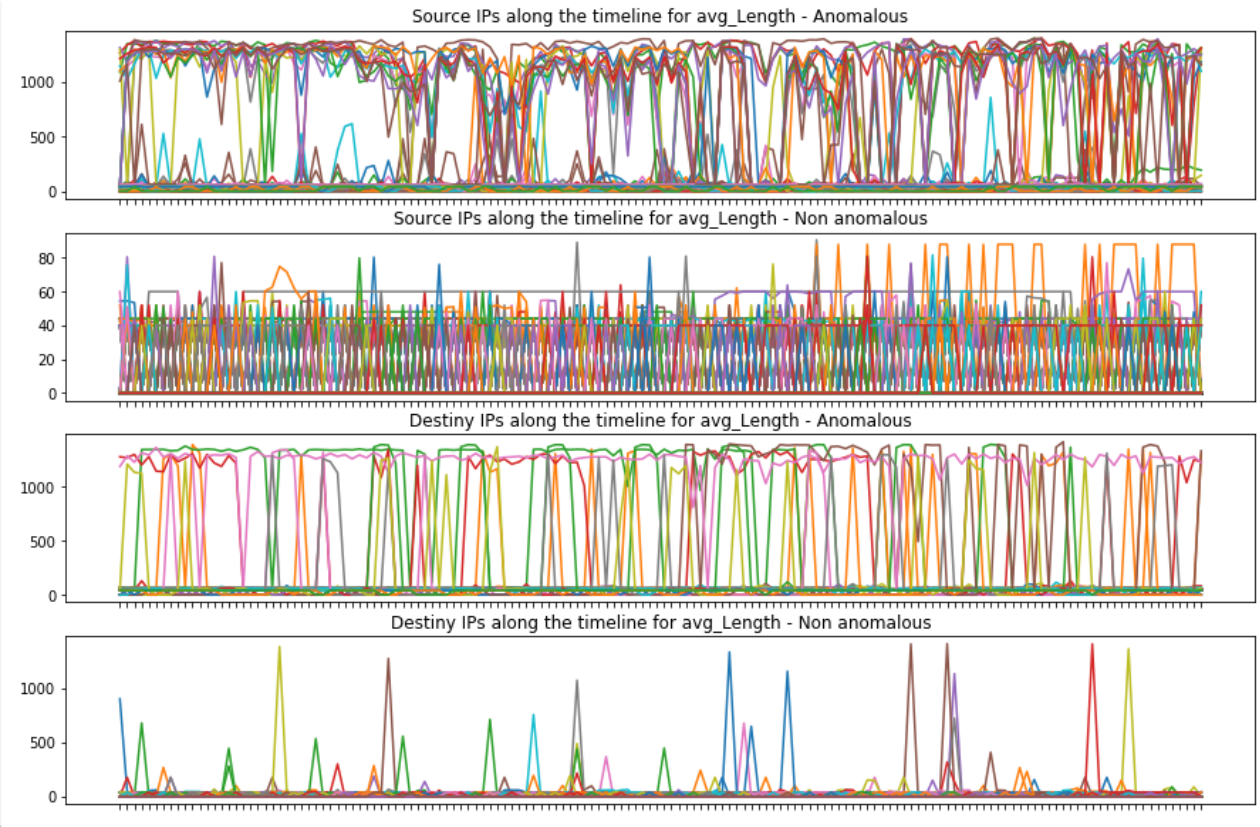


Figura 4.1: Visualización de evaluación para la media de la variable *Length*

Para la variable *Length* se puede observar lo siguiente:

- La media en las conexiones de origen tiene más variación que las no anómalas.
- Las no anómalas por lo general tienen valores medios muy pequeños.
- Para las conexiones de destino también existen conexiones con valores grandes, pero la mayoría tienen valores pequeños.

- Por lo general la media es mayor en las conexiones anómalas, puede tratarse de inyección de *malware* por ejemplo.
- La desviación estándar tiene valores más altos por lo general para los anómalos, al contrario de las no anómalas que tiene valores muy pequeños.
- El sumatorio de los valores es mucho más alto en anómalos y muy bajo en comparación con los no anómalos.
- Ciertas representaciones coinciden tanto para destino como origen, lo que puede significar conexiones asiduas entre ambas IPs.

Para la variable *IP1_IP2_time_diff_per_value* (la diferencia de tiempos entre conexiones únicas de origen y destino) se puede observar lo siguiente:

- Se presenta un patrón temporal para la media.
- Las conexiones anómalas tienen medias mayores.
- El patrón temporal también se aprecia en la desviación estándar.
- Los tiempos en las no anómalas en su mayor parte son muy pequeños.
- El sumatorio es mayor en anómalos.

Para la variable *IP1_time_diff_per_value* (la diferencia de tiempos entre conexiones únicas de origen) se observa un patrón temporal en las conexiones destino, pero no en las de origen para los valores medios. Para *IP2_time_diff_per_value* se observa el mismo patrón pero cambiado a las conexiones origen. Por otro lado, la distancia entre IPs, parece mostrar que las conexiones anómalas, tanto para origen como destino, pueden provenir de redes más cercanas.

En resumen para las variables de la cantidad de puertos en el último minuto se puede encontrar lo siguiente:

- Se tiene, de media, un mayor uso de puertos de destino en las conexiones anómalas, mucho más grande para las conexiones de destino que las de origen.
- Para las observaciones no anómalos por lo general la cantidad de puertos de origen es muy baja.

- Destaca mucho la cantidad de puertos de origen que se utilizan en las conexiones de destino.
- Para la cantidad de puertos de destino, es claramente mayor para conexiones origen en los valores anómalos, lo que significa que esas IPs pueden estar haciendo escaneos.
- La suma de puertos de destino para las conexiones de origen anómalas es muy grande en comparación al resto.

En las variables binarias se puede observar que la conexión tenga un valor positivo en alguna de ellas puede favorecer a considerarlo como anómalo. Tal y como se comentó con anterioridad, estos pueden ser comandos que se utilicen para infectar los dispositivos, por lo que puede tener sentido.

En resumen las anomalías presentan los siguientes rasgos:

- Mayores tiempos de diferencias entre conexiones (es decir, el tiempo que ha tardado en volver a conectarse), siguiendo cierto patrón.
- Mayor cantidad de puertos de origen y destino utilizados que las conexiones no anómalas.
- Mayor cantidad de información en los paquetes transmitida que los no anómalos.
- Cuentan con un pequeño porcentaje mayor de is_busybox e is_enable"que los no anómalos, pero muy pequeño.
- Las conexiones pueden tener cierta periodicidad en conexiones únicas de IP origen e IP destino, pero las origen siguen realizando conexiones a otros destinos.
- Una mayor cantidad de puertos destinos, puede indicar un escaneo.
- Mayor cantidad de información transmitida puede presentarse como inyección de *malware*.

Capítulo 5

Mejoras y Trabajos Futuros

Uno de los puntos más importantes a mejorar, dado que también se considera un proceso iterativo, es el preprocesado de los datos, este proceso es necesario para poder mejorar los resultados obtenidos y mejorar el funcionamiento de los modelos empleados. Esta mejora puede producirse con un mayor conocimiento dentro del área, de redes y de dispositivos IoT. De esta manera se pueden generar variables que aporten más información y puedan ser más útiles.

Como idea, en lugar de tratas las conexiones como un registro con variables de temporalidad y periodos comunes repetidos en el mismo eje, se podrían tratar como imágenes, donde cada variable a la que se ha aplicado a la ventana fuera una fila de la imagen y las columnas el periodo único. De esta forma se podría aplicar un *Convolutional Autoencoder* que pudiera obtener más información de las capas de convolución [33].

Por otro lado, también se podría mejorar la interpretación de los resultados con un mayor conocimiento del área y se podría realizar una análisis de interpretabilidad de los modelos, es decir, encontrar el porqué el modelo muestra los resultados obtenidos. Muchas de estas herramientas están orientadas a conjuntos de datos etiquetados (*LIME* [34]), por lo que podría ser complicado.

Otra mejora que se podría realizar es el análisis utilizando más datos, en este caso solo se están utilizando datos de un día en un periodo concreto. Con una cantidad mayor de datos se puede aprender la temporalidad de periodos más largos y detectar distintos tipos de anomalías. Con un día puede darse el caso de que lo que se ha detectado como anomalía es un patrón normal que se produce todos lo días, mientras que con los actuales al solo observar un periodo

pequeño de tiempo lo clasifique como anomalía.

La definición de que es y no es anomalía se podría mejorar utilizando un conjunto de datos etiquetado como *benchmark* para evaluar que tal ha funcionado el modelo. A causa de no tener un conjunto etiquetado no se puede definir un umbral que permita obtener el mejor ratio posible entre falsos positivos y valores reales. Con un conjunto etiquetado se podría utilizar la curva AUC - ROC (*Area Under the Curve - Receiver Operating Characteristics*) que utiliza ese ratio para definir si las anomalías se están detectando correctamente y también para mejorar el balance entre falsos positivos y reales, dado que los modelos no son perfectos y ciertas conexiones pueden ser solo valores atípicos pero no anómalos. En este caso no se ha utilizado ningún conjunto de libre uso, dado que no se han encontrado con las mismas características o los que se encuentran disponibles son muy antiguos y no son aplicables a nuestro problema.

Por último se podrían aplicar nuevas técnicas en el estado del arte, alguno de los ejemplos serían los siguientes:

- Uso de *Variational Autoencoders*.
- Uso de redes neuronales LSTM.
- Con un conjunto semi-etiquetado utilizar redes generativas antagónicas (o *Generative Adversarial Networks*).
- En el caso de disponer un conjunto de datos completamente etiquetado se podrían utilizar redes neuronales y *Gradient Boosting* como los algoritmos de *XGBoost, Catboost y LightGBM*.

Apéndice A

Evaluación de resultados

En este anexo se incluye el *Jupyter Notebook* utilizado en la evaluación de resultados con sus correspondientes gráficas y comentarios.

Libraries and paths

In [11]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

models = ['models_2019_06_01_21_21/',
          'models_2019_06_01_21_44/',
          'models_2019_06_01_22_11/']
```

Load data

In [12]:
```
data_30T = pd.read_csv(models[0] + 'detected_anomalies.csv',
                       sep='\t',
                       index_col=[0],)
data_10T = pd.read_csv(models[1] + 'detected_anomalies.csv',
                       sep='\t',
                       index_col=[0],)
data_5T = pd.read_csv(models[2] + 'detected_anomalies.csv',
                      sep='\t',
                      index_col=[0],)
print('Anomalies loaded!')
```

```
Anomalies loaded!
```

In [13]:
```
data_30T.head()
```

Out[13]:

	('avg_Length', 'avg_2019-02-01 09:30:00')	('avg_Length', 'avg_2019-02-01 10:00:00')	('avg_Length', 'avg_2019-02-01 10:30:00')	('avg_Length', 'avg_2019-02-01 11:00:00')	('avg_Length', 'avg_2019-02-01 11:30:00')	('avg_Length', 'avg_2019-02-01 12:00:00')	('avg_Le 'avg_20 12:30:0
104.192.1.15source	60.000000	59.978492	59.802326	60.000000	60.854239	60.613653	60
104.248.135.165source	61.504604	61.425604	61.297026	61.551904	61.367338	61.300701	61
104.248.37.237source	61.878968	61.898538	61.447473	59.957901	60.179832	59.987654	60
122.228.19.80source	44.000000	44.000000	44.000000	44.000000	44.000000	44.000000	44
133.34.156.192source	40.000000	40.031746	40.000000	108.638418	105.238342	40.000000	157

5 rows × 1102 columns

In [14]:
```
data_10T.head()
```

Out[14]:

	('avg_Length', 'avg_2019-02-01 09:30:00')	('avg_Length', 'avg_2019-02-01 09:40:00')	('avg_Length', 'avg_2019-02-01 09:50:00')	('avg_Length', 'avg_2019-02-01 10:00:00')	('avg_Length', 'avg_2019-02-01 10:10:00')	('avg_Length', 'avg_2019-02-01 10:20:00')	('avg_Le 'avg_20 10:30:0
104.192.1.15source	60.000000	60.000000	60.000000	60.181121	60.134423	59.200262	59
104.248.135.165source	60.000000	60.000000	61.549547	61.269025	61.802870	61.192314	61
104.248.37.237source	63.946527	61.741286	61.572278	61.074691	62.175765	62.102029	61
122.228.19.80source	44.000000	44.000000	44.000000	44.000000	44.000000	44.000000	0
133.34.156.192source	40.000000	40.000000	40.000000	40.090909	40.000000	40.000000	40

5 rows × 3202 columns

In [19]:
```
data_5T.head()
```

Out[19]:

	('avg_Length', 'avg_2019-02-01 09:35:00')	('avg_Length', 'avg_2019-02-01 09:40:00')	('avg_Length', 'avg_2019-02-01 09:45:00')	('avg_Length', 'avg_2019-02-01 09:50:00')	('avg_Length', 'avg_2019-02-01 09:55:00')	('avg_Length', 'avg_2019-02-01 10:00:00')	('avg_Le 'avg_20 10:05:0(
103.9.179.179source	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	(
104.192.1.15source	60.000000	60.000000	60.00000	60.000000	60.000000	60.000000	6(
104.248.135.165source	60.000000	60.000000	60.00000	61.557567	61.542310	61.398975	6(
104.248.228.27source	40.000000	40.000000	0.00000	50.500000	53.000000	44.000000	4(
104.248.37.237source	63.946527	61.981725	61.43956	61.983410	61.127975	61.215415	6(

5 rows × 6310 columns

In [20]:
```
print(f'Shape of data_30T is {data_30T.shape}')
print(f'Shape of data_10T is {data_10T.shape}')
print(f'Shape of data_5T is {data_5T.shape}')
```

```
Shape of data_30T is (368, 1102)
Shape of data_10T is (387, 3202)
Shape of data_5T is (396, 6310)
```

In [21]:
```
common_30T_5T = data_5T.index.isin(data_30T.index)
common_10T_5T = data_5T.index.isin(data_10T.index)
common_three = common_30T_5T & common_10T_5T
print(f'Number of common anomalies between 30T and 5T is {common_30T_5T.sum()}')
print(f'Number of common anomalies between 10T and 5T is {common_10T_5T.sum()}')
print(f'Number of common anomalies between all three is {common_three.sum()}')
```

```
Number of common anomalies between 30T and 5T is 362
Number of common anomalies between 10T and 5T is 383
Number of common anomalies between all three is 360
```

- Most anomalies are common between all three
- A hard look on 5T would give most of the insight for all three
- special cases might be found due to the window period
- Special cases must be looked at

Main evaluation will be performed on 5T dataset

In [22]:
```
data_5T_full = pd.read_csv('../data/final/final_dataset_5T.csv',
                           sep='\t',
                           index_col=[0])
data_5T = data_5T.drop(['host0','host1', 'host2',
                        'host3', 'host4', 'host5',
                        'host6', 'host7', 'is_src',
                        'is_anomaly'],
                       axis=1)
data_5T_full = data_5T_full.drop(['host0','host1', 'host2',
                        'host3', 'host4', 'host5',
                        'host6', 'host7', 'is_src'],
                       axis=1)
```

In [46]:
```
data_5T.columns = data_5T.columns.map(eval)
data_5T_full.columns = data_5T.columns.map(eval)
```

In [47]:
```
data_5T.head()
```

Out[47]:

	avg_Length						
	avg_2019-02-01 09:35:00	avg_2019-02-01 09:40:00	avg_2019-02-01 09:45:00	avg_2019-02-01 09:50:00	avg_2019-02-01 09:55:00	avg_2019-02-01 10:00:00	avg_2019-0 10:05:00
103.9.179.179source	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.00
104.192.1.15source	60.000000	60.000000	60.00000	60.000000	60.000000	60.000000	60.38
104.248.135.165source	60.000000	60.000000	60.00000	61.557567	61.542310	61.398975	61.07
104.248.228.27source	40.000000	40.000000	0.00000	50.500000	53.000000	44.000000	40.00
104.248.37.237source	63.946527	61.981725	61.43956	61.983410	61.127975	61.215415	60.94

5 rows × 6300 columns

In [138]:
```
data_5T_full.head()
```

Out[138]:

	avg_Length						
	avg_2019-02-01 09:35:00	avg_2019-02-01 09:40:00	avg_2019-02-01 09:45:00	avg_2019-02-01 09:50:00	avg_2019-02-01 09:55:00	avg_2019-02-01 10:00:00	avg_2019-02-01 10:05:00
1.0.171.245source	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.0.255.54source	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.1.158.248source	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.1.217.60source	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.1.230.15source	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 6300 columns

Get rid of the anomalies and generate a sample of 4000 (around 10%)

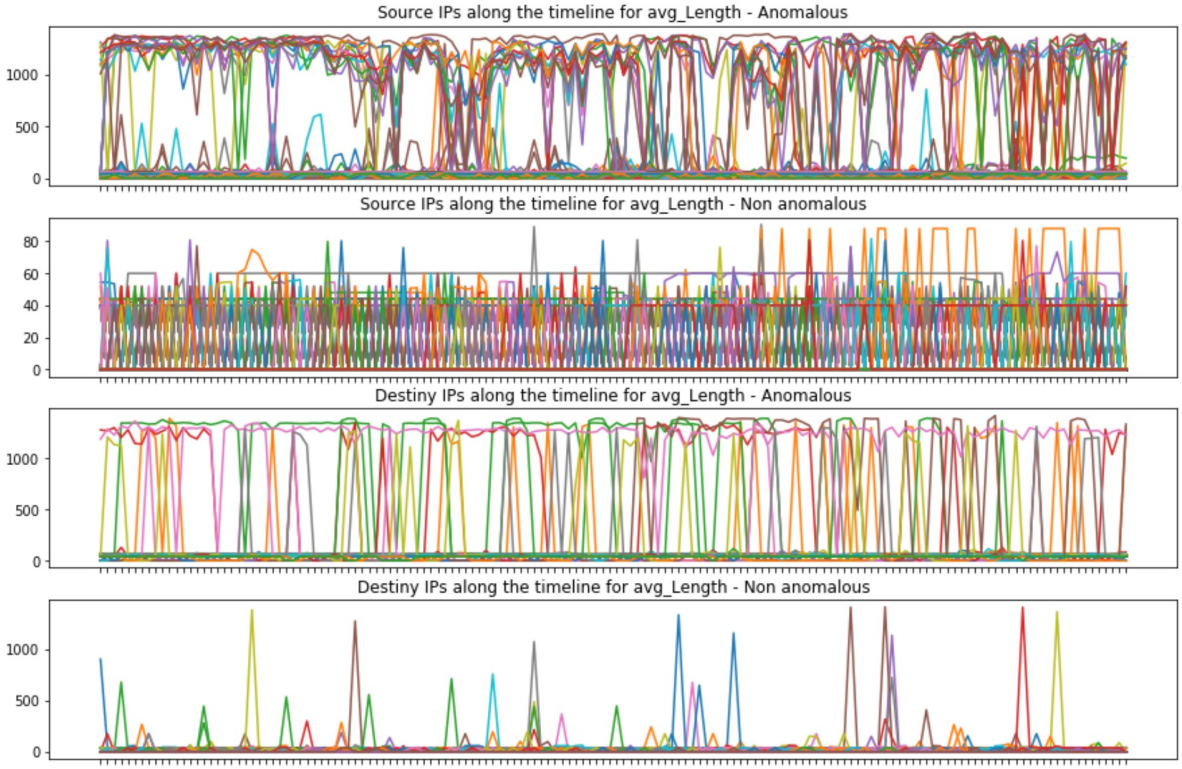
In [149]:
```
idx = ~data_5T_full.index.isin(data_5T.index)
data_5T_full = data_5T_full[idx]
data_5T_sample = data_5T_full.sample(4000)
```

In [150]:
```
def plot_all_ips(data, sample, var):
    fig, ax = plt.subplots(nrows=4, figsize=(15,10))
    source_ips = [c for c in data.index if 'source' in c]
    sample_source = [c for c in sample.index if 'source' in c]
    dst_ips = [c for c in data.index if 'dst' in c]
    sample_dst = [c for c in sample.index if 'dst' in c]
    for i in source_ips:
        ax[0].plot(data.loc[i, var])
    for i in sample_source:
        ax[1].plot(sample.loc[i, var])
    for j in dst_ips:
        ax[2].plot(data.loc[j, var])
    for j in sample_dst:
        ax[3].plot(sample.loc[j, var])

    ax[0].set_title(f'Source IPs along the timeline for {var} - Anomalous')
    ax[0].set_xticklabels([])
    ax[1].set_title(f'Source IPs along the timeline for {var} - Non anomalous')
    ax[1].set_xticklabels([])
    ax[2].set_title(f'Destiny IPs along the timeline for {var} - Anomalous')
    ax[2].set_xticklabels([])
    ax[3].set_title(f'Destiny IPs along the timeline for {var} - Non anomalous')
    ax[3].set_xticklabels([])
```

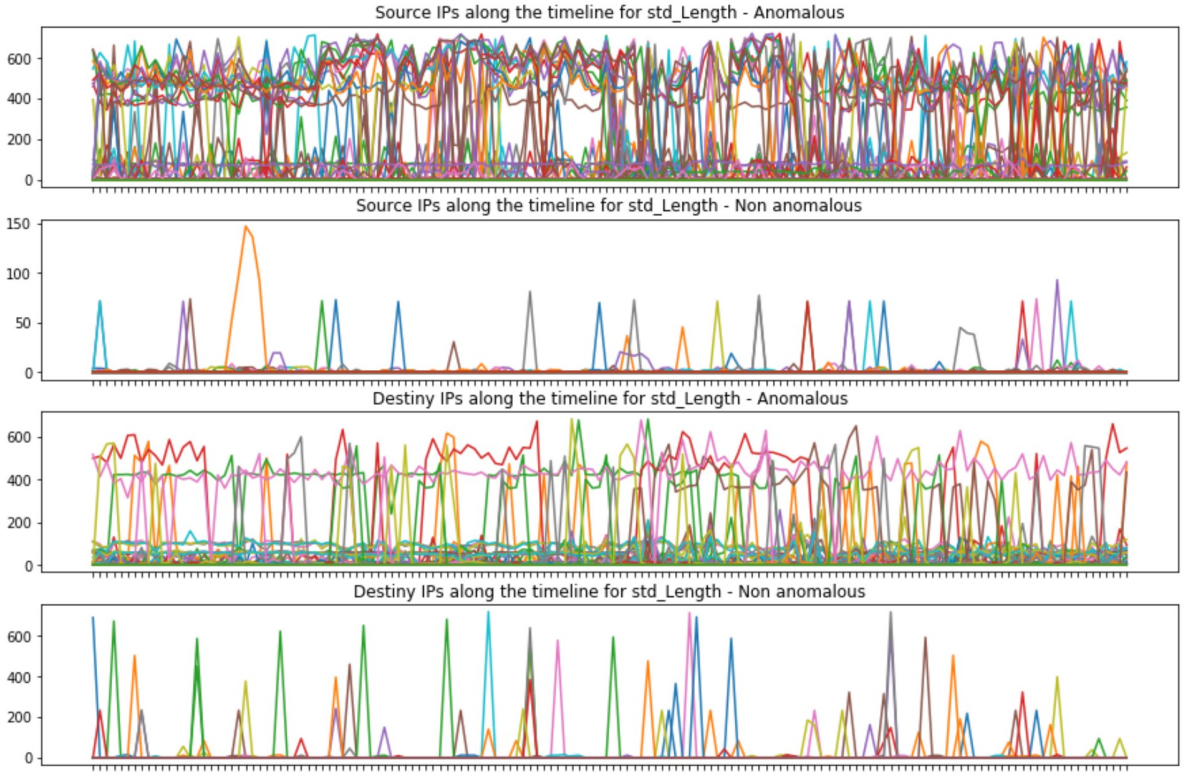
Length

In [152]:
```
plot_all_ips(data_5T, data_5T_sample, 'avg_Length')
```



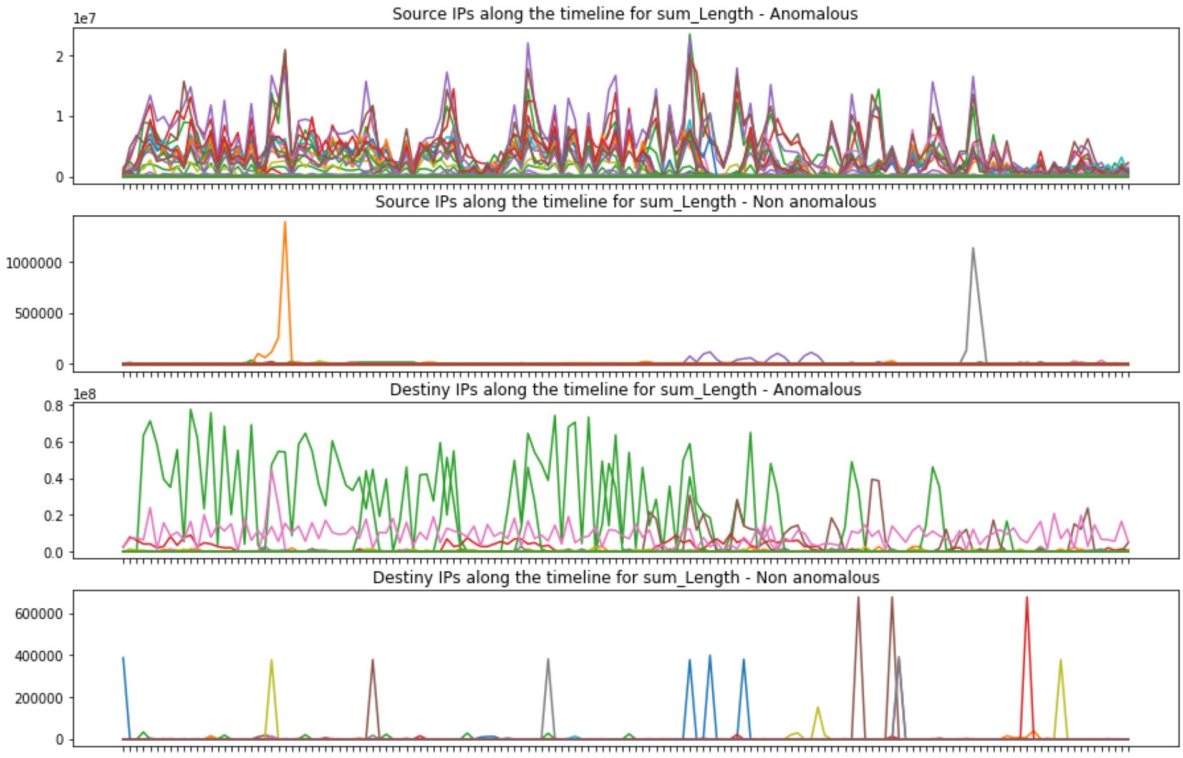
- Most anomalous source ips have a high variation of packet avg_length
- Destiny ips avg_length is lower than sources ips
- Average length is much higher for anomalous ips

In [153]:
```
plot_all_ips(data_5T, data_5T_sample, 'std_Length')
```



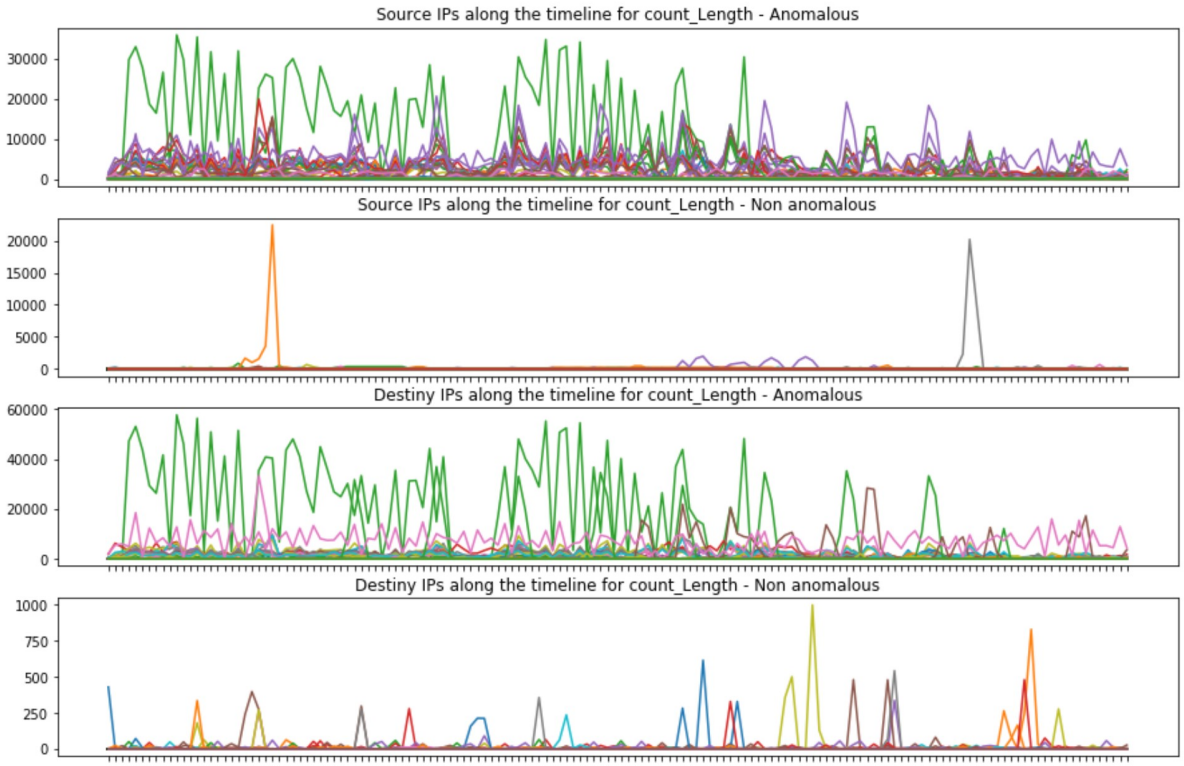
- Most source ips have either a great std or really low
- For destiny ips most ips tends towards a lower std, meaning that most destiny ips have lower packages lengths overall
- Even though most non anomalous ips have a really low average, there is still some std for destiny non anomalous ips

In [154]:
```
plot_all_ips(data_5T, data_5T_sample, 'sum_Length')
```



- Similar trend for mos source ips
- In destiny ips there is three above the rest, for the most part the sum is pretty low
- Sum of the packet lengths is much higher and variate in anomalous than non anomalous

```
In [155]: plot_all_ips(data_5T, data_5T_sample, 'count_Length')
```



- Both for source and destiny there are a couple of ips that stand out
- Those ips have the same form, so there is a connection between two ips (source and dst) that really stand out from both ends

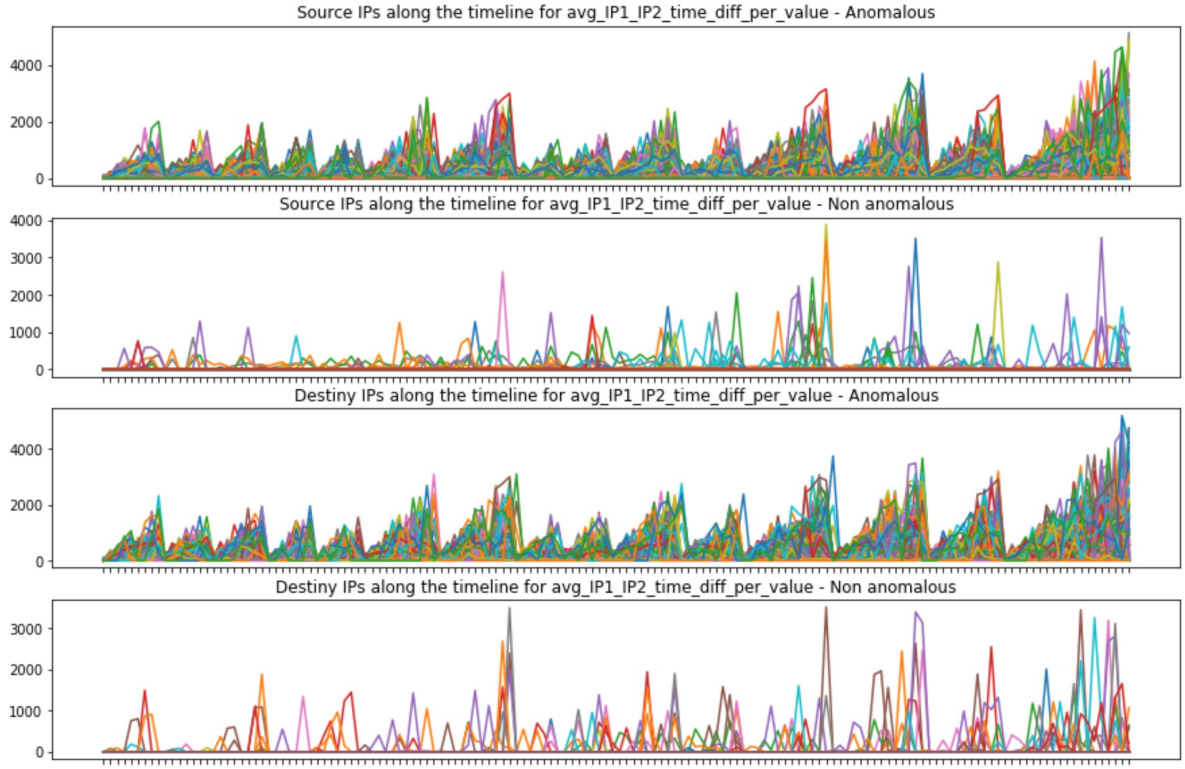
TODO: get those singular ips

Length summary

- Higher package length in anomalous
- Anomalous and non anomalous have similar std, whereas for source ips the std in non anomalous is pretty low
- Amount of information transmited (sum) is much higher in anomalous IPs
- Anomalies have higher counts, which means higher connections during the timeline
- The difference in package length could point to malware inyection

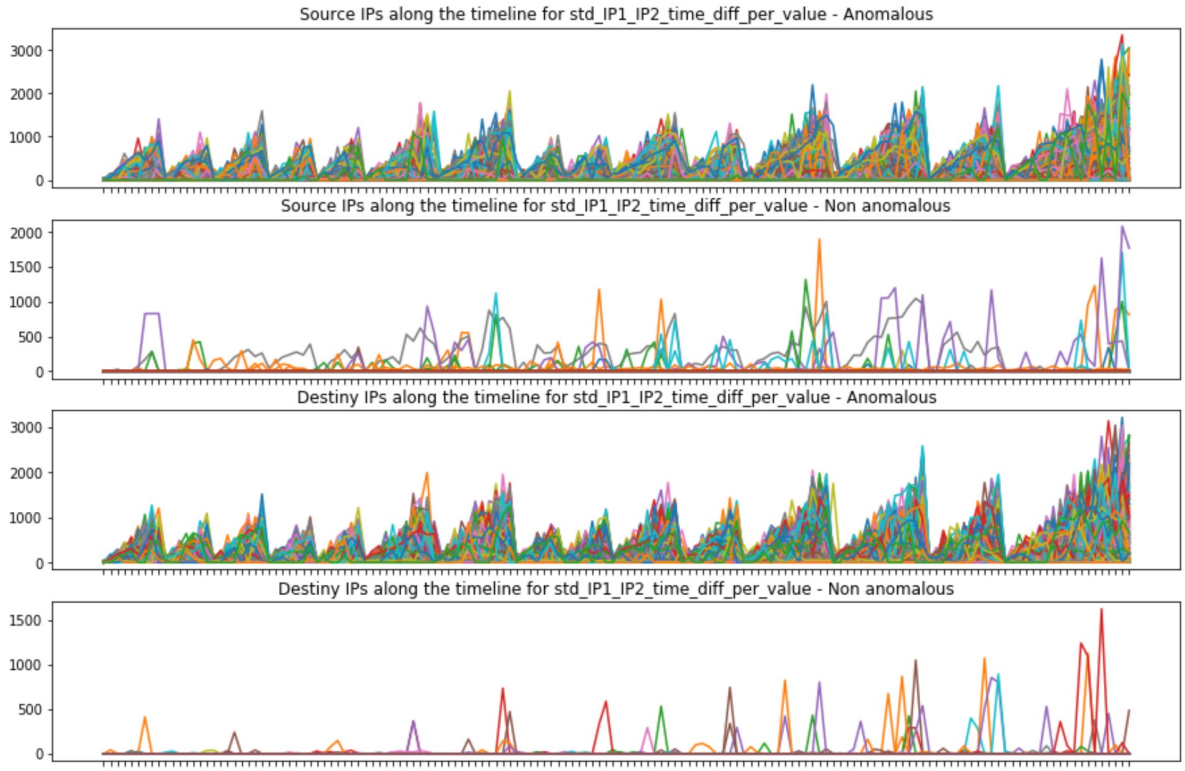
IP1_IP2_time_diff_per_value

In [156]:
```
plot_all_ips(data_5T, data_5T_sample, 'avg_IP1_IP2_time_diff_per_value')
```



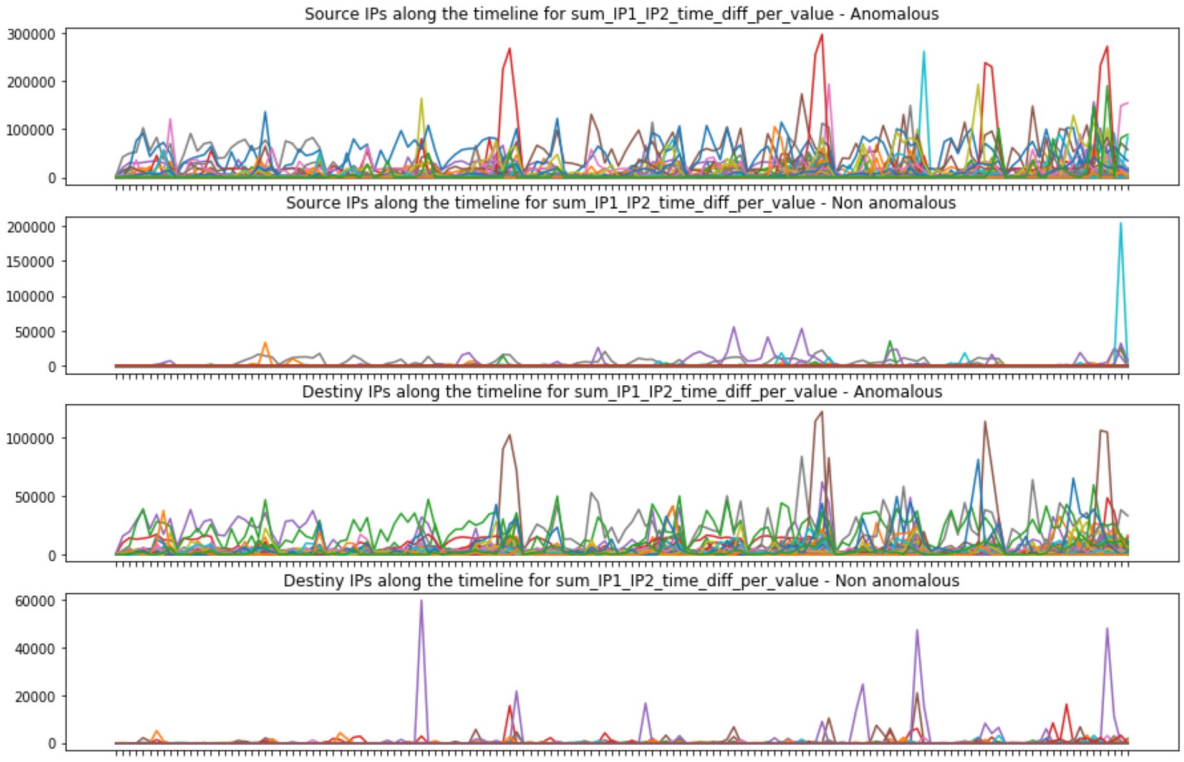
- source and destiny ips follow the same pattern on anomalous ips
- Time differences between unique ip connections are higher on anomalous ips, meaning that the same connection happens much more often on non anomalous ips

In [157]:
```
plot_all_ips(data_5T, data_5T_sample, 'std_IP1_IP2_time_diff_per_value')
```



- Similar pattern as in average
- Non anomalous have really low std, meanig that most are perodically done, whereas anomalous follow a connection pattern

In [190]:
```
plot_all_ips(data_5T, data_5T_sample, 'sum_IP1_IP2_time_diff_per_value')
```



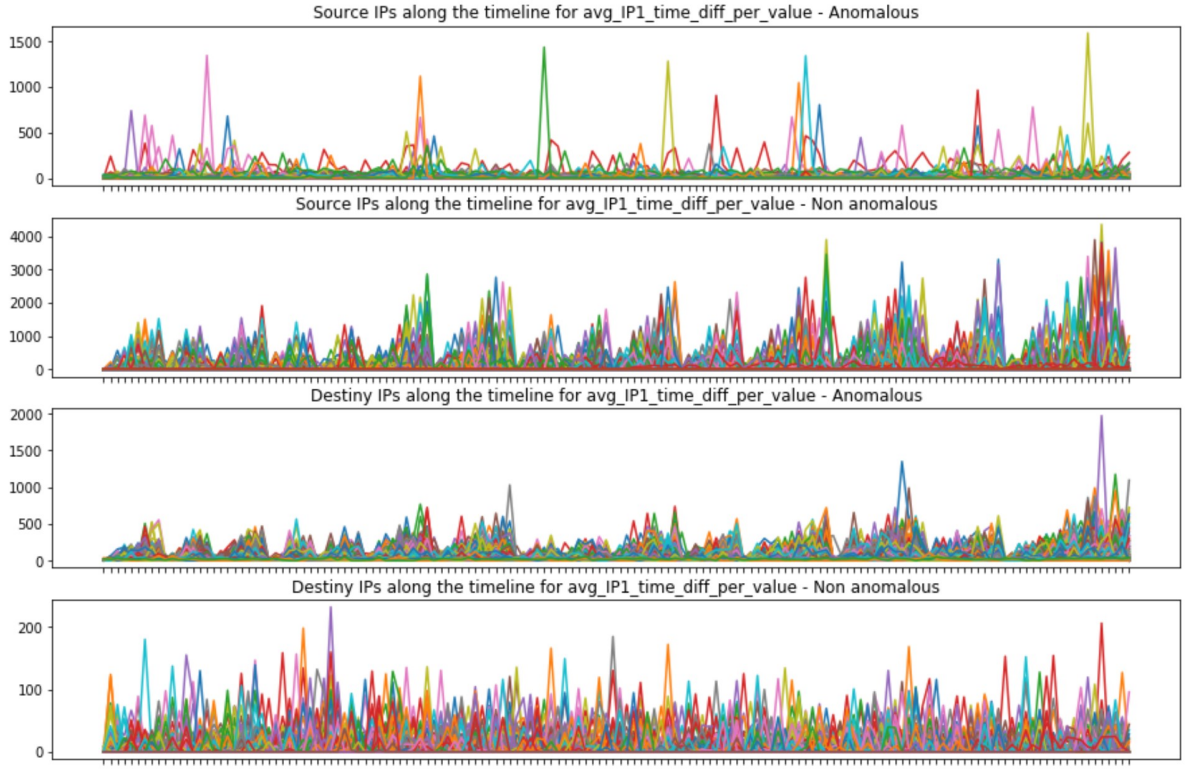
- Difference times are clearly higher in anomalous that follow the pattern

IP1_IP2_time_diff_per_value summary

- Clear connection pattern
- Higher difference times in anomalous
- The pattern could mean that the connections are programaticaly done for anomalous observations

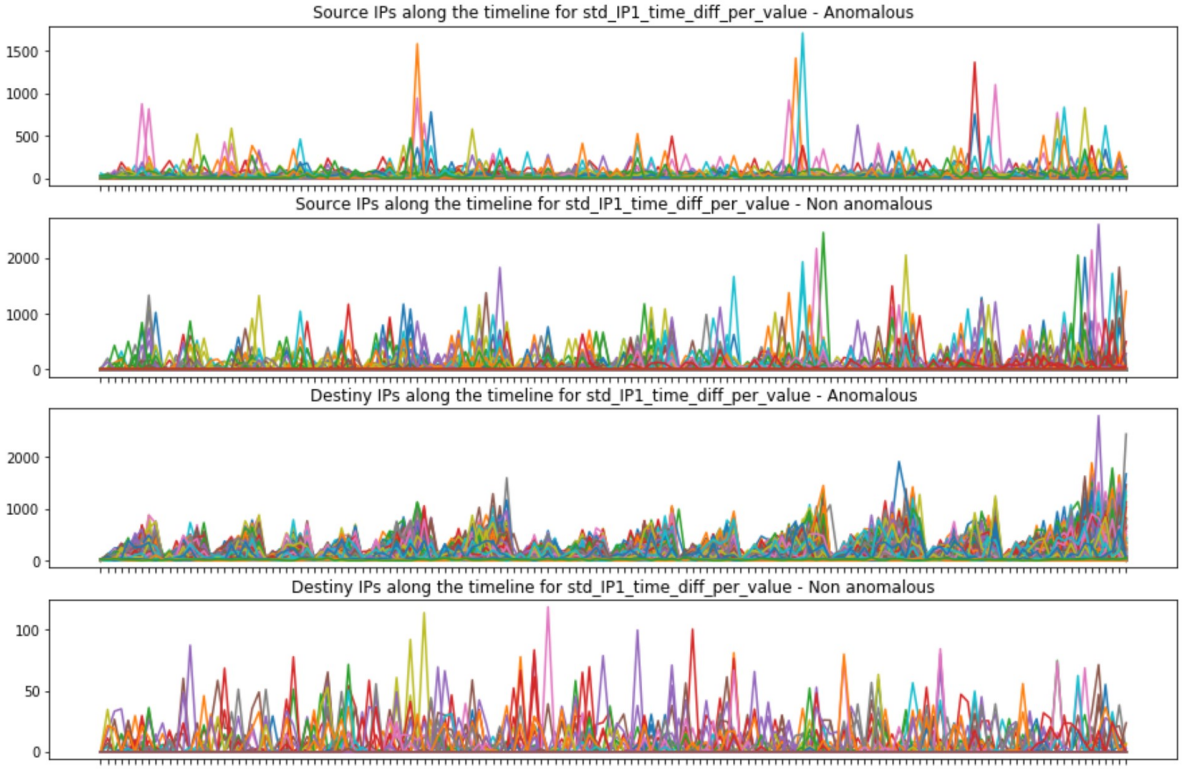
IP1_time_diff_per_value

In [191]:
```
plot_all_ips(data_5T, data_5T_sample, 'avg_IP1_time_diff_per_value')
```



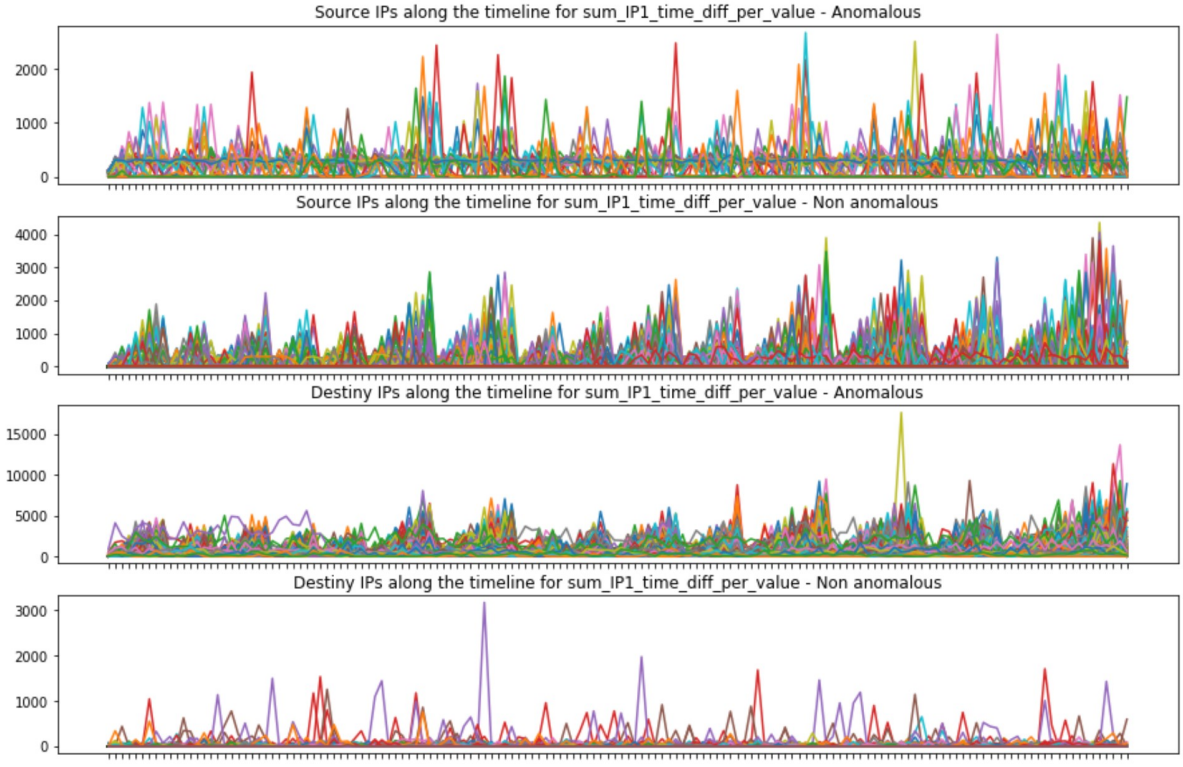
- No clear pattern on anomalous source ips, but a clear pattern on non anomalous
- Pattern on anomalous destiny ips, this could indicate that same source anomalous ip are connecting more times to different destiny ips (time difference is low), whereas the connections from source to the same destiny ip follows a pattern
- Non anomalous destiny are all over the place

In [161]:
```
plot_all_ips(data_5T, data_5T_sample, 'std_IP1_time_diff_per_value')
```



- Same conclusions as in average

In [192]:
```
plot_all_ips(data_5T, data_5T_sample, 'sum_IP1_time_diff_per_value')
```



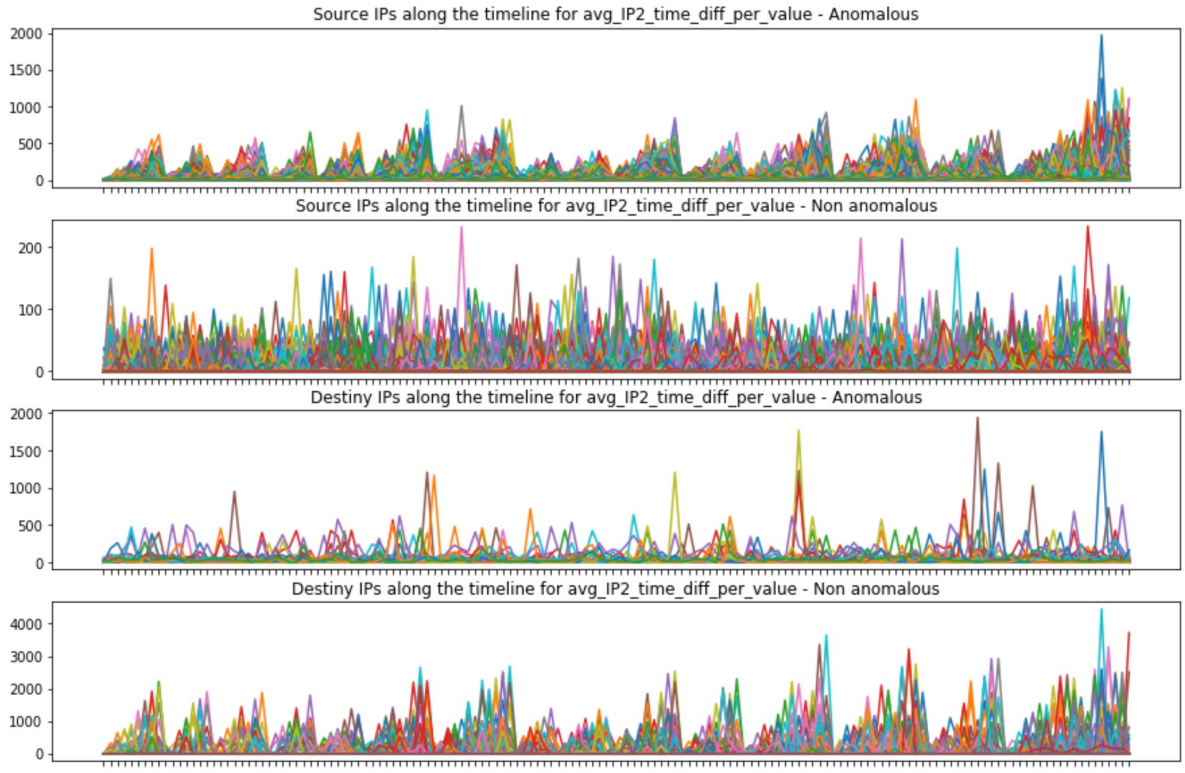
- Anomalous soures sum does not follow a clear pattern but most of the ocurrences seems to be pretty low endorsing the previous hypothesis
- Same as previous

IP1_time_diff_per_value summary

- Anomalous source ips have really low time differences with no clear pattern as anomalous
- Source anomalous ip coulb be used to establish connections to different destiny ips
- Avg-std and sum could endorse the hypothesis
- Anomalous destiny ips follow a connection pattern from the same source ip

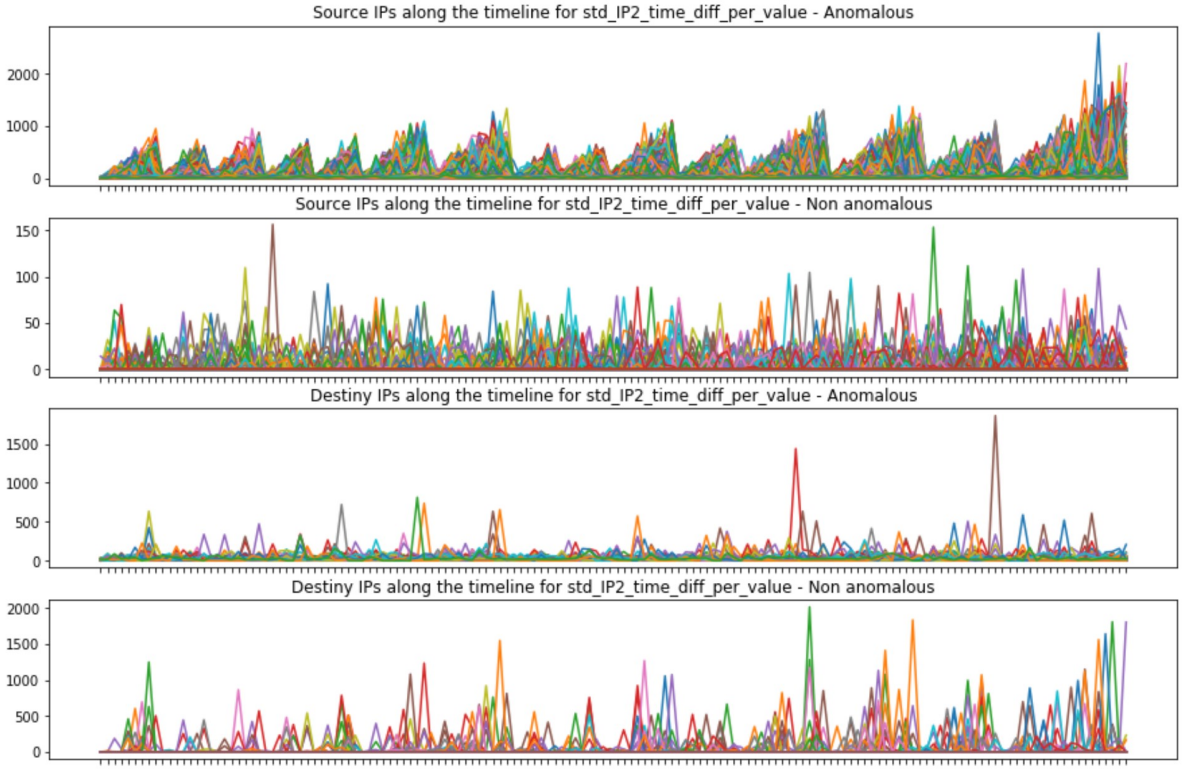
IP2_time_diff_per_value

```
In [164]: plot_all_ips(data_5T, data_5T_sample, 'avg_IP2_time_diff_per_value')
```



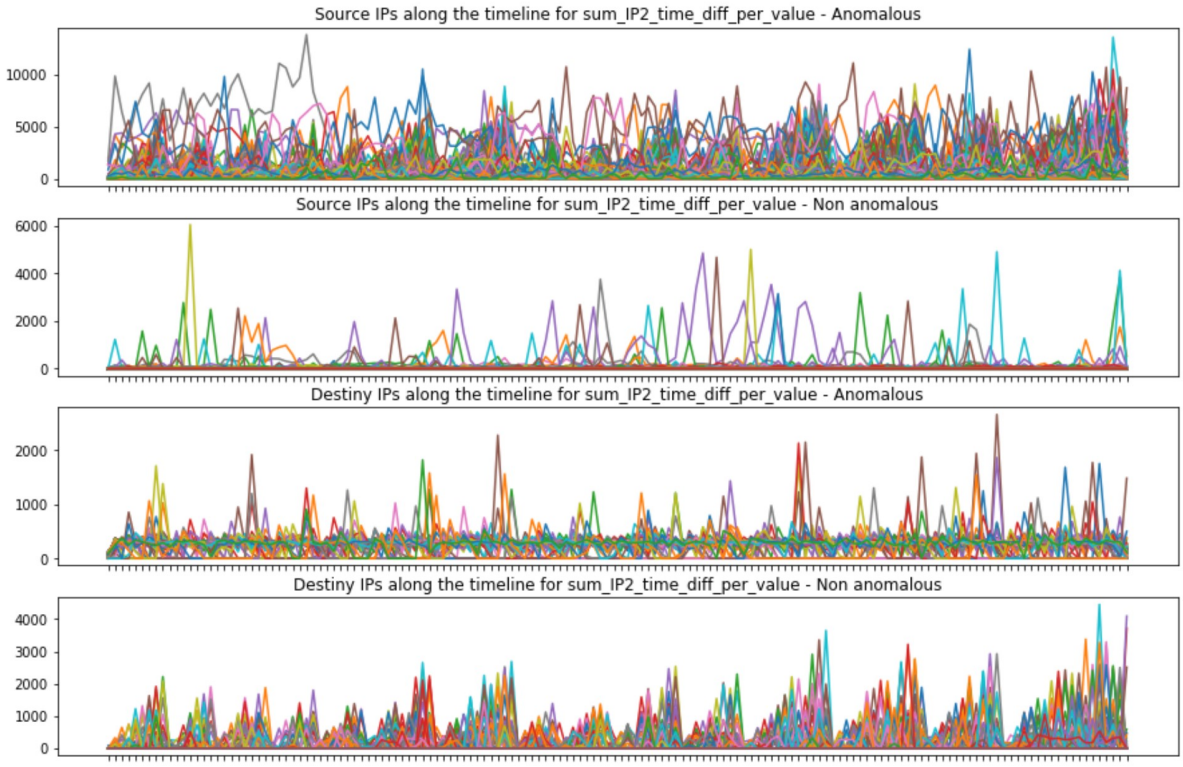
- Source anomalous ips follow a pattern similar as IP1 time diff, makes sense since the same source ips connect to different destiny ips
- Anomalous destiny ips have lower tiime differences
- Non anomalous sources are all over the place

In [165]:
```
plot_all_ips(data_5T, data_5T_sample, 'std_IP2_time_diff_per_value')
```



- Same as above and IP1 time diff

In [166]:
```
plot_all_ips(data_5T, data_5T_sample, 'sum_IP2_time_diff_per_value')
```

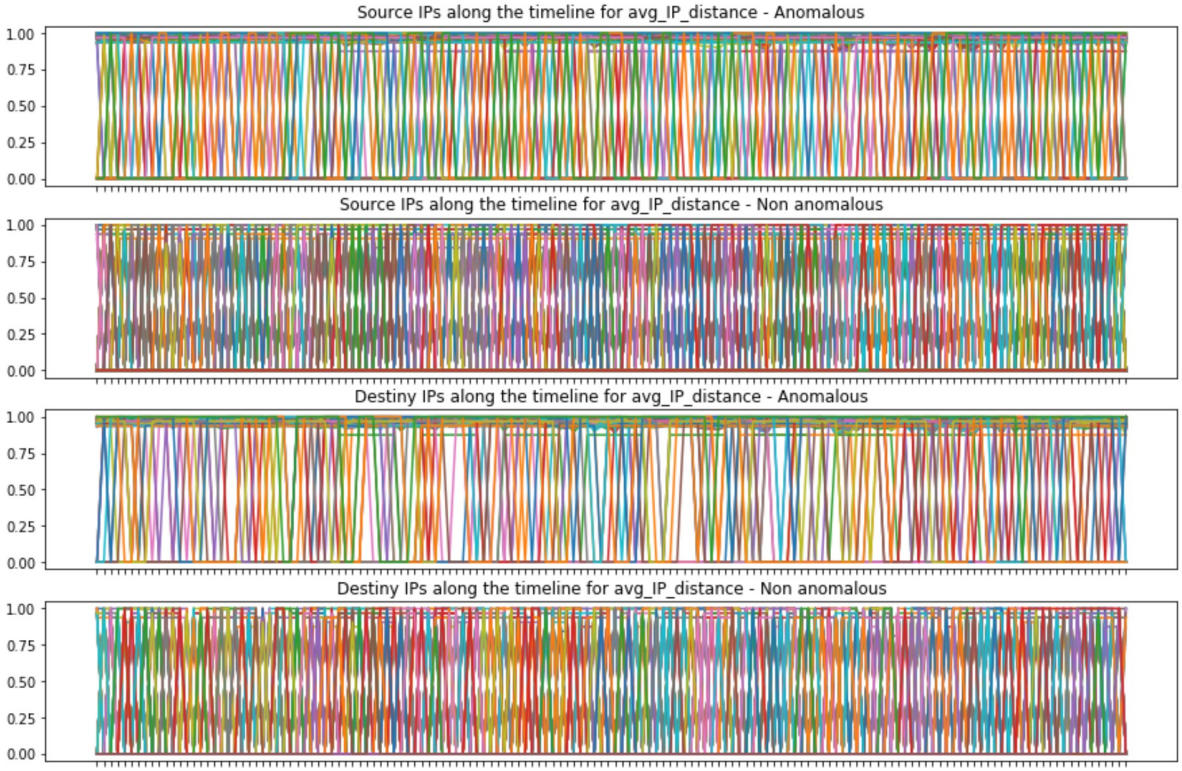


IP2_time_diff_per_value summary

- Same hypothesis as in IP1, meaning that anomalous sources check different destiny IPs and destiny IPS are checked by different sources
- Following IP1_IP2 time diff, there could be a pattern between hosts
- Patterns between sources and destinies have switched from IP1

IP_Distance

```
In [168]: plot_all_ips(data_5T, data_5T_sample, 'avg_IP_distance')
```



- No clear pattern

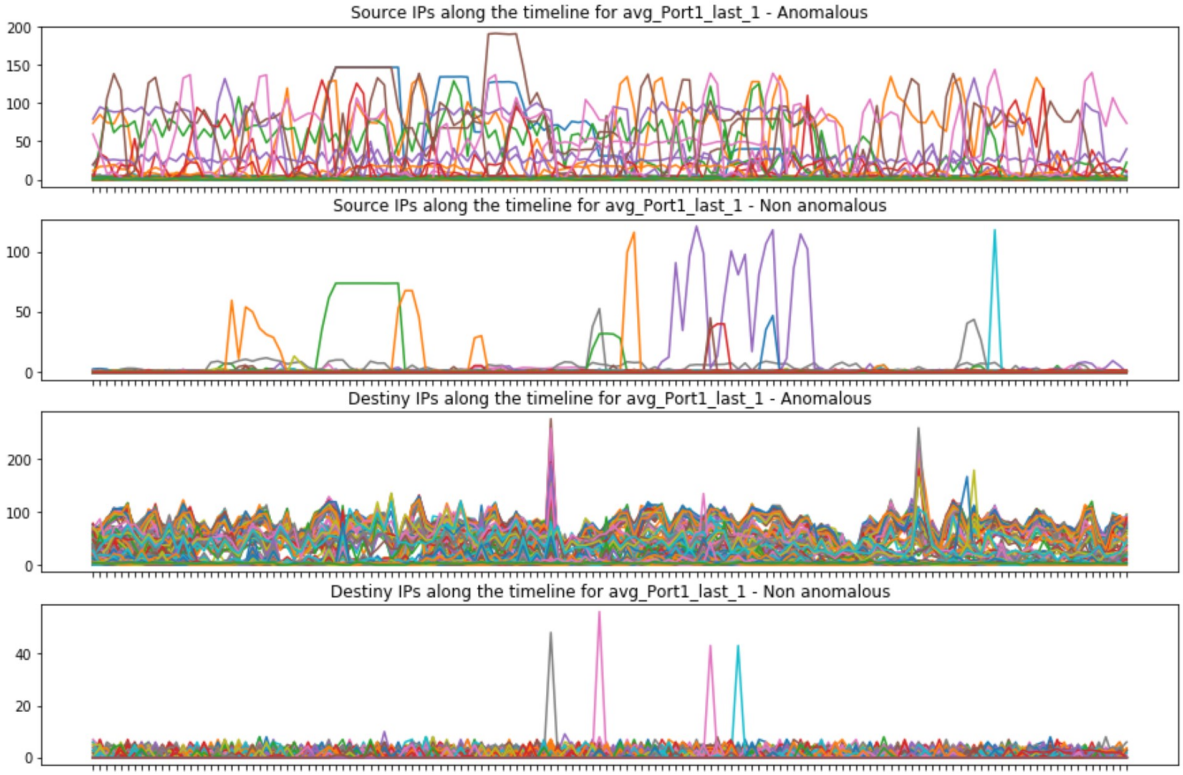
In [169]:
```
plot_all_ips(data_5T, data_5T_sample, 'std_IP_distance')
```

IP_distance summary

- Anomalous have a higher std, that could indicate that anomalous connections come from closer networks than non anomalous

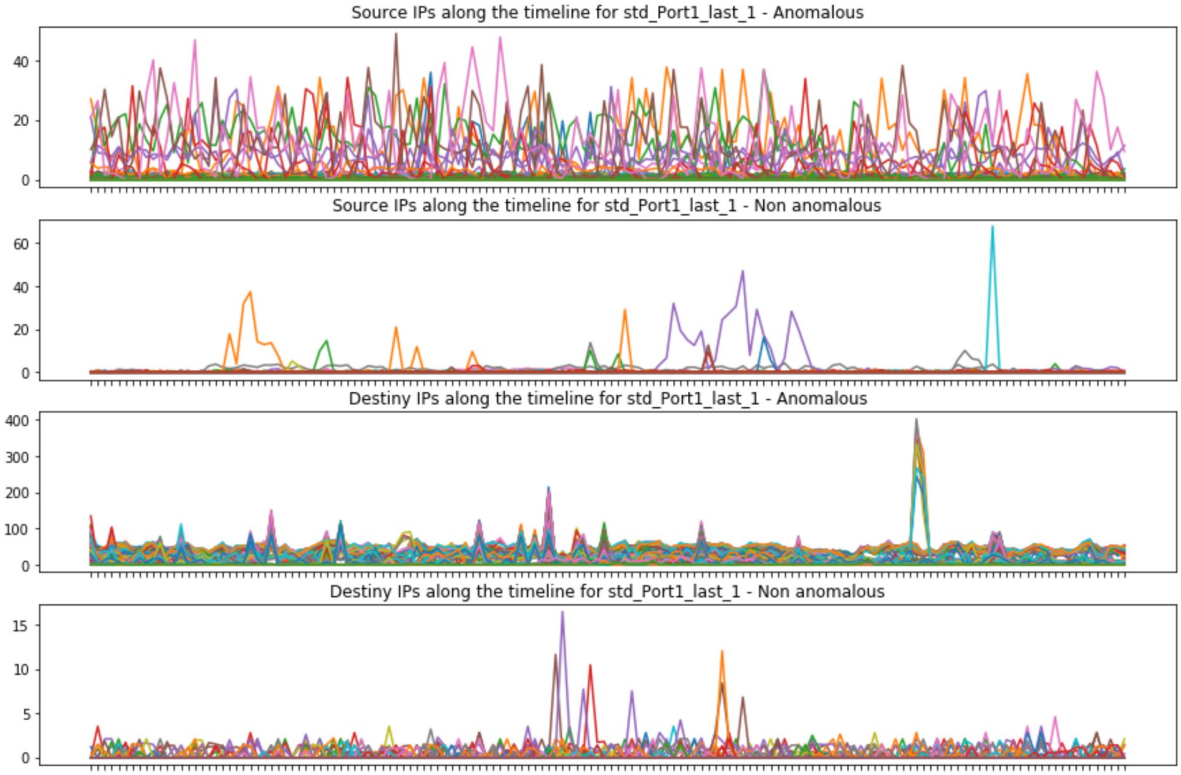
Port1_last_1

In [172]:
```
plot_all_ips(data_5T, data_5T_sample, 'avg_Port1_last_1')
```



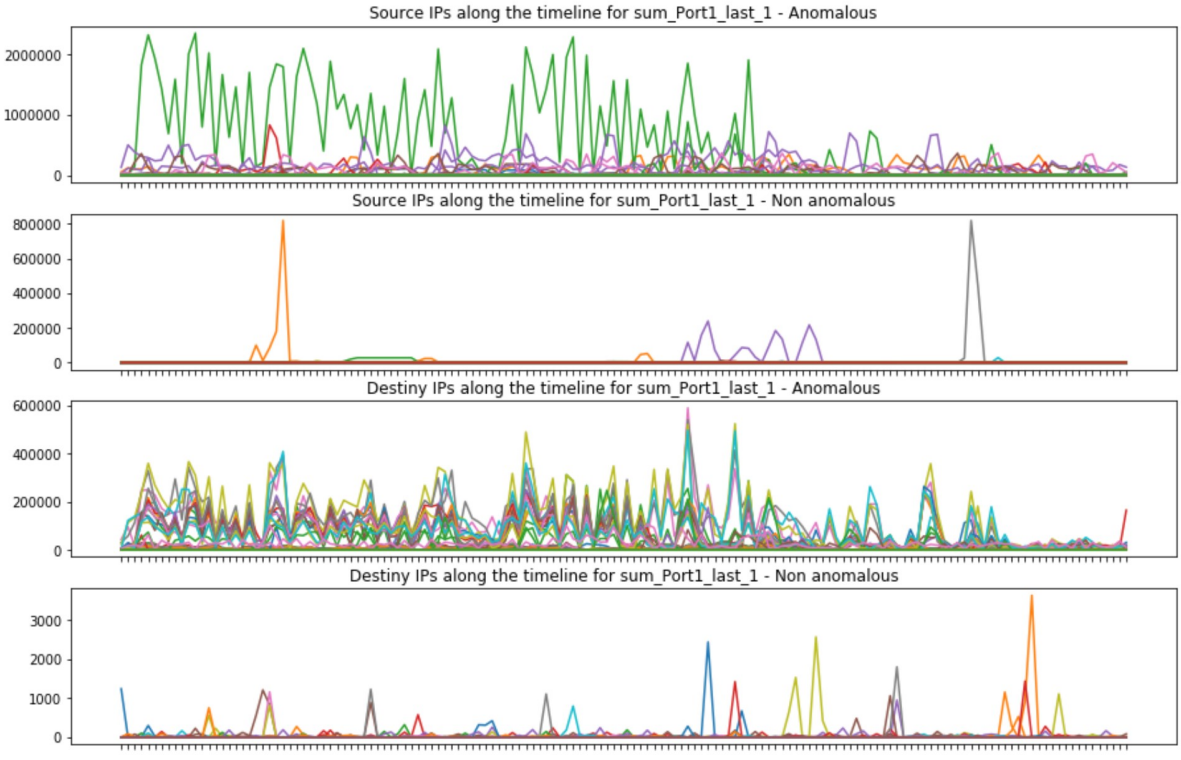
- Source anomalous have a higher use of ports thatn non anomalous, clearly
- Same four anomalous destinies, meaning that the ip that stacblished the connection to the destiny ip has used a larger amount of ports than the non anomalous counterparts

In [173]:
```
plot_all_ips(data_5T, data_5T_sample, 'std_Port1_last_1')
```



- Non anomalous tends to have lower std and lower average, meaning that the source ip uses a low number of ports during the 5 minute period

In [174]:
```
plot_all_ips(data_5T, data_5T_sample, 'sum_Port1_last_1')
```



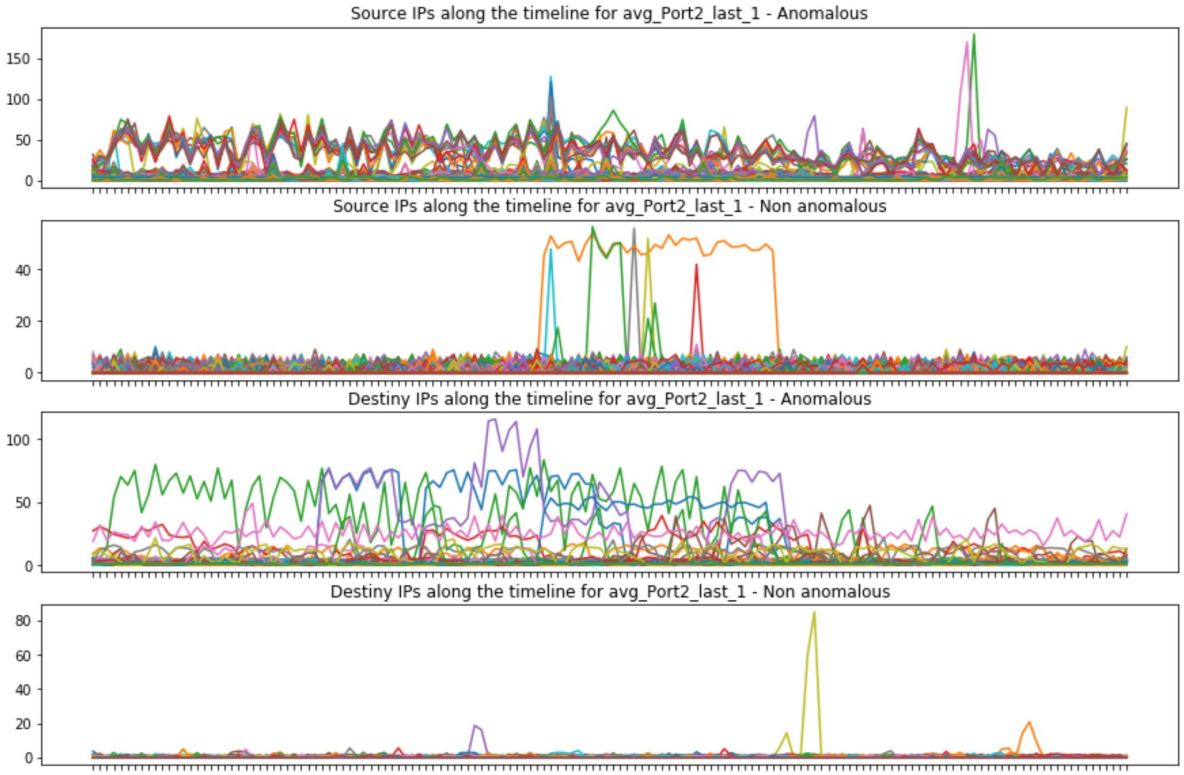
- Overall anomalous ips uses hicgher number of ports

Port1_last_1 summary

- Anomalous IPs have tend to be connected from ips with a high port usage in the last five minutes
- Non anomalous have very little usage of ports

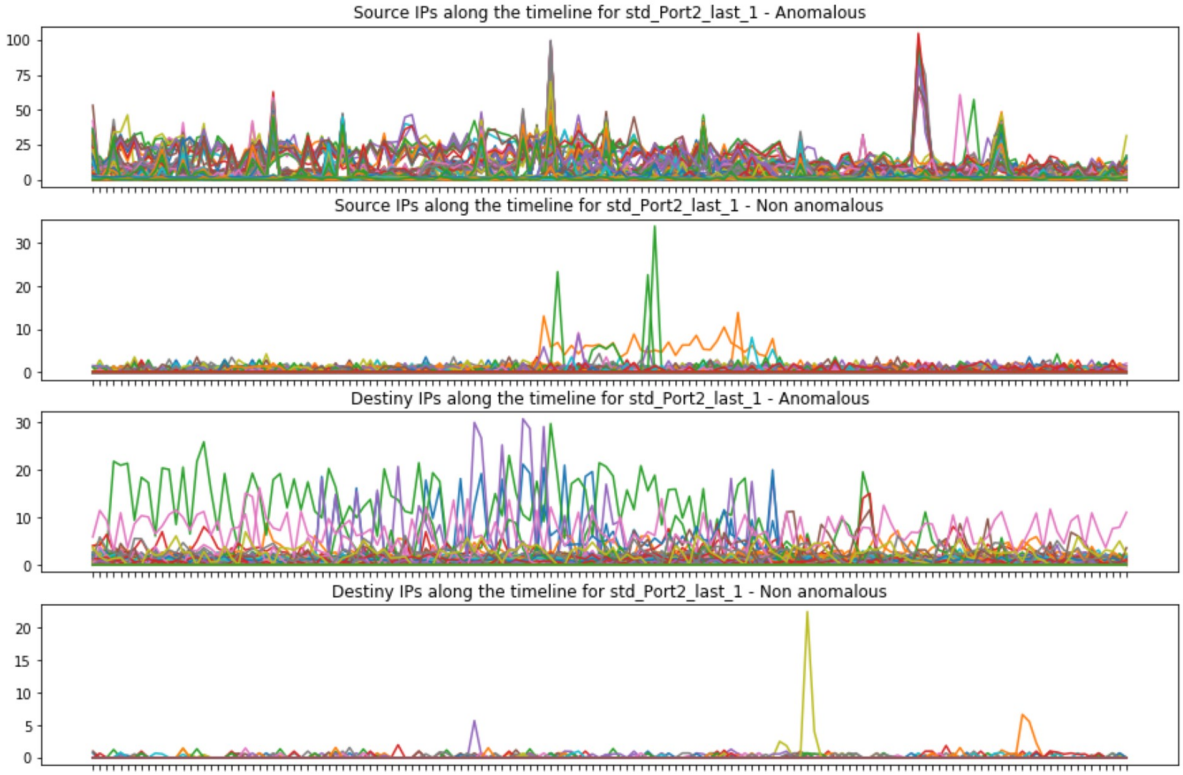
Port2_last_1

In [176]:
```
plot_all_ips(data_5T, data_5T_sample, 'avg_Port2_last_1')
```



- Anomalous source connections connects to more ports than non anomalous. Meaning that the source IPs connects to different destiny ports
- Destiny ip follow the same pattern, higher port connections from the same ip. Overall destiny anomalous ips have a larger amount of connections to different ports
- Most non anomaloues have little to no port2 values

In [177]:
```
plot_all_ips(data_5T, data_5T_sample, 'std_Port2_last_1')
```



- Non anomalous observarions have low deviation

In [178]:
```
plot_all_ips(data_5T, data_5T_sample, 'sum_Port2_last_1')
```

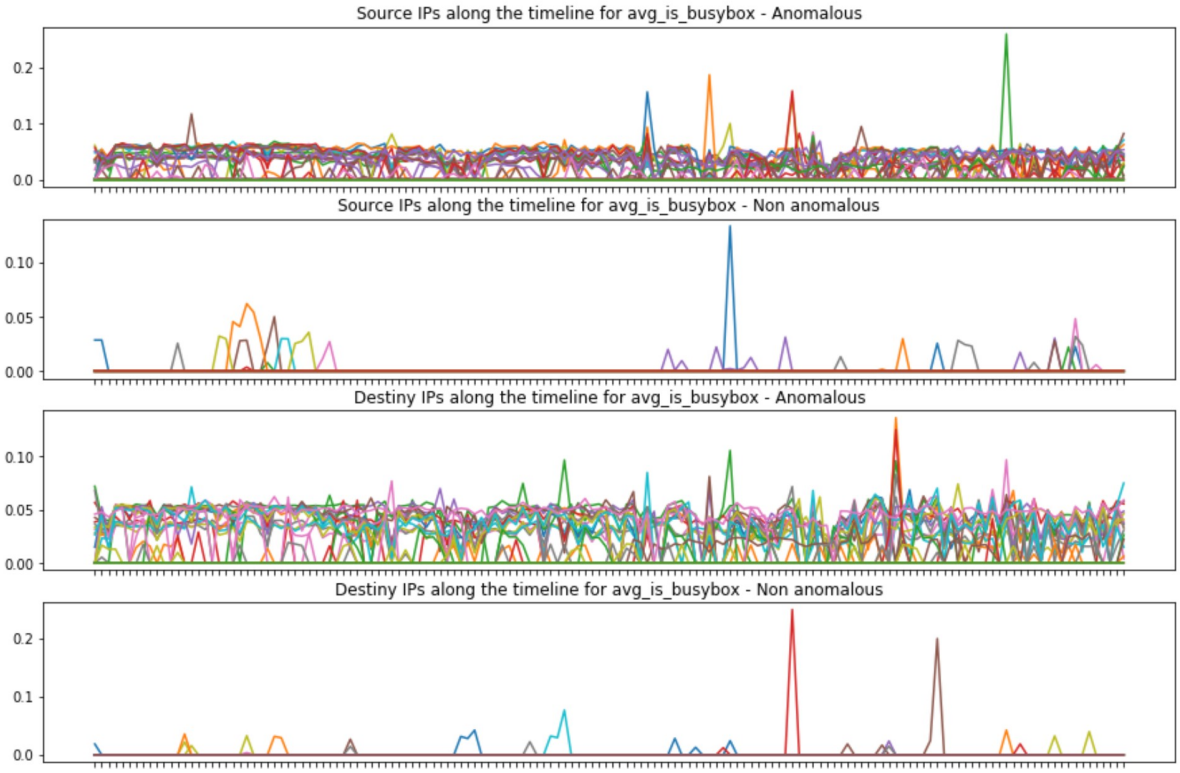
- Overall, anomalous have a higher destiny port usage than non anomalous

Port2_last_1

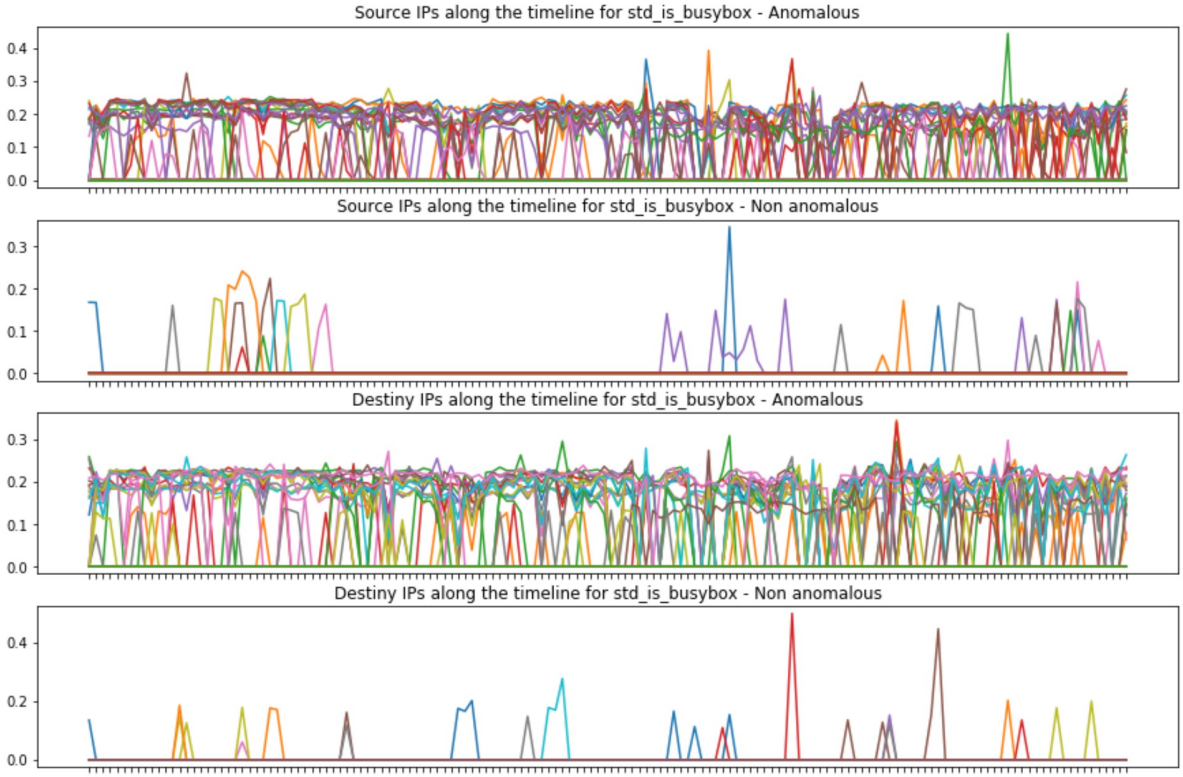
- Higher port usage on anomalous connections, this could indicate that a scanning is tacking place
- Higher port2 and port1 tends to be anomalous

is_busybox

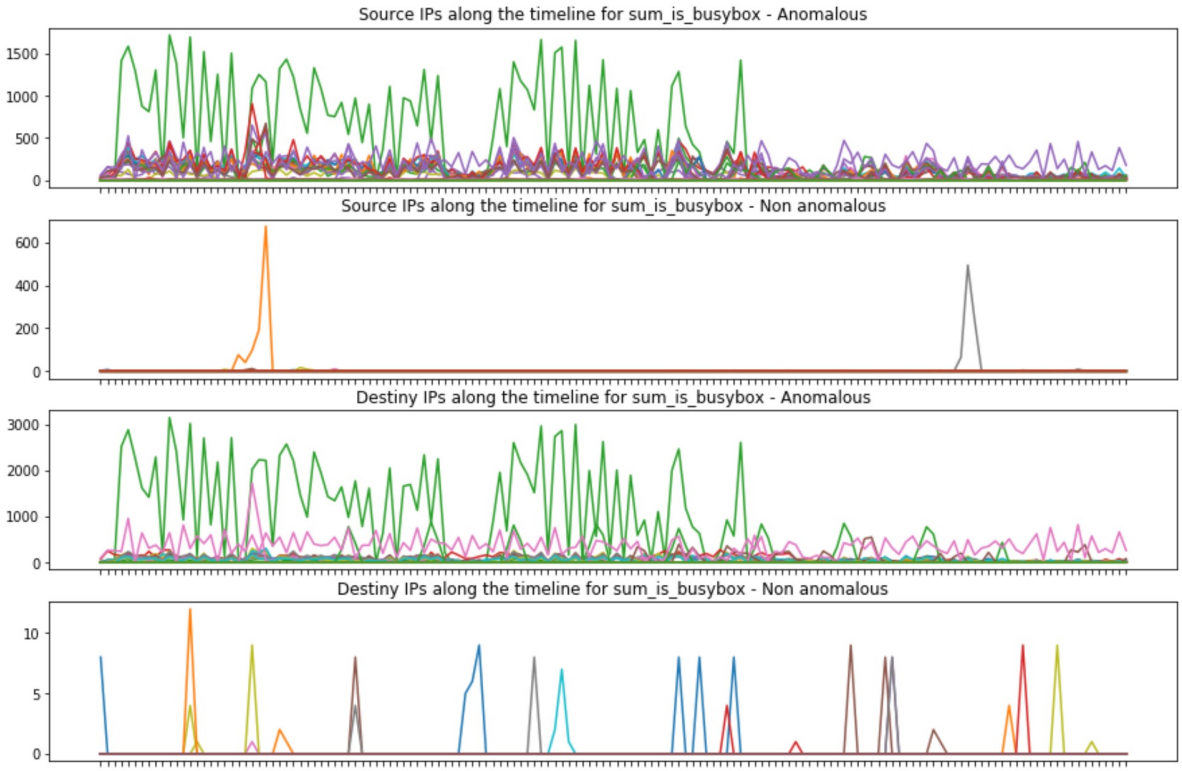
In [180]:
```
plot_all_ips(data_5T, data_5T_sample, 'avg_is_busybox')
```



In [181]:
```
plot_all_ips(data_5T, data_5T_sample, 'std_is_busybox')
```



In [182]:
```
plot_all_ips(data_5T, data_5T_sample, 'sum_is_busybox')
```

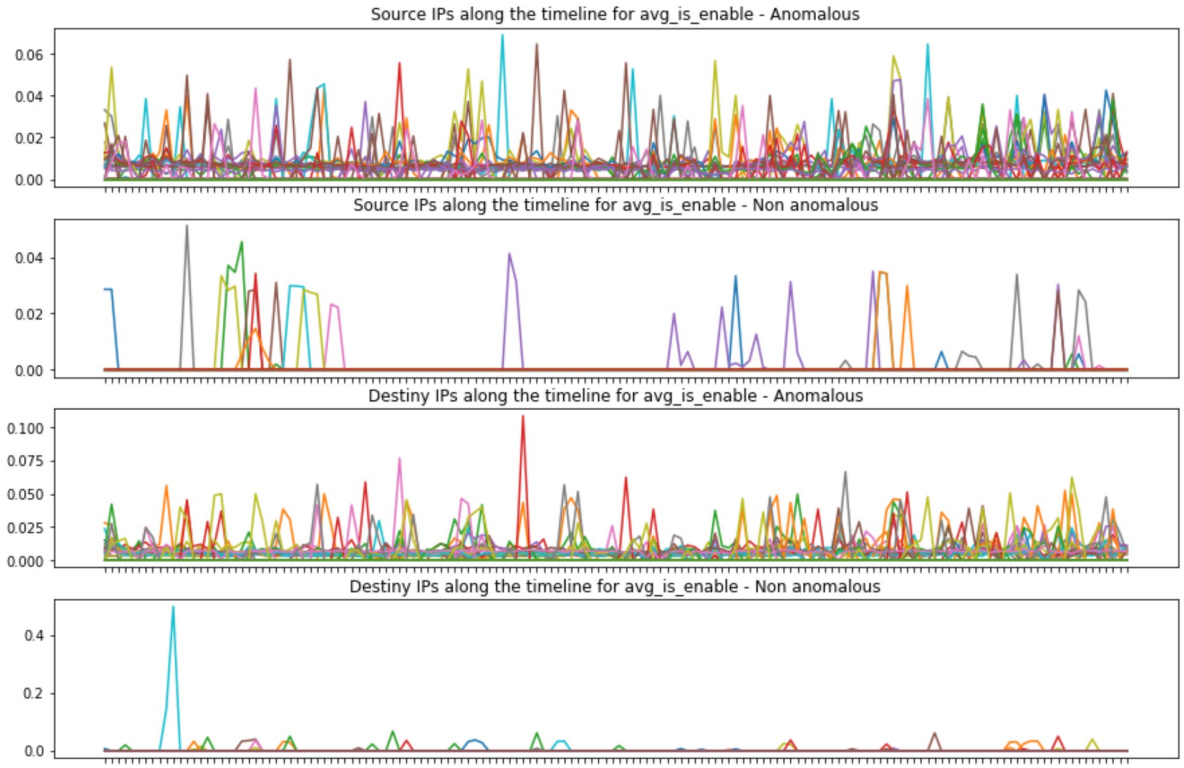


is_busybox summary

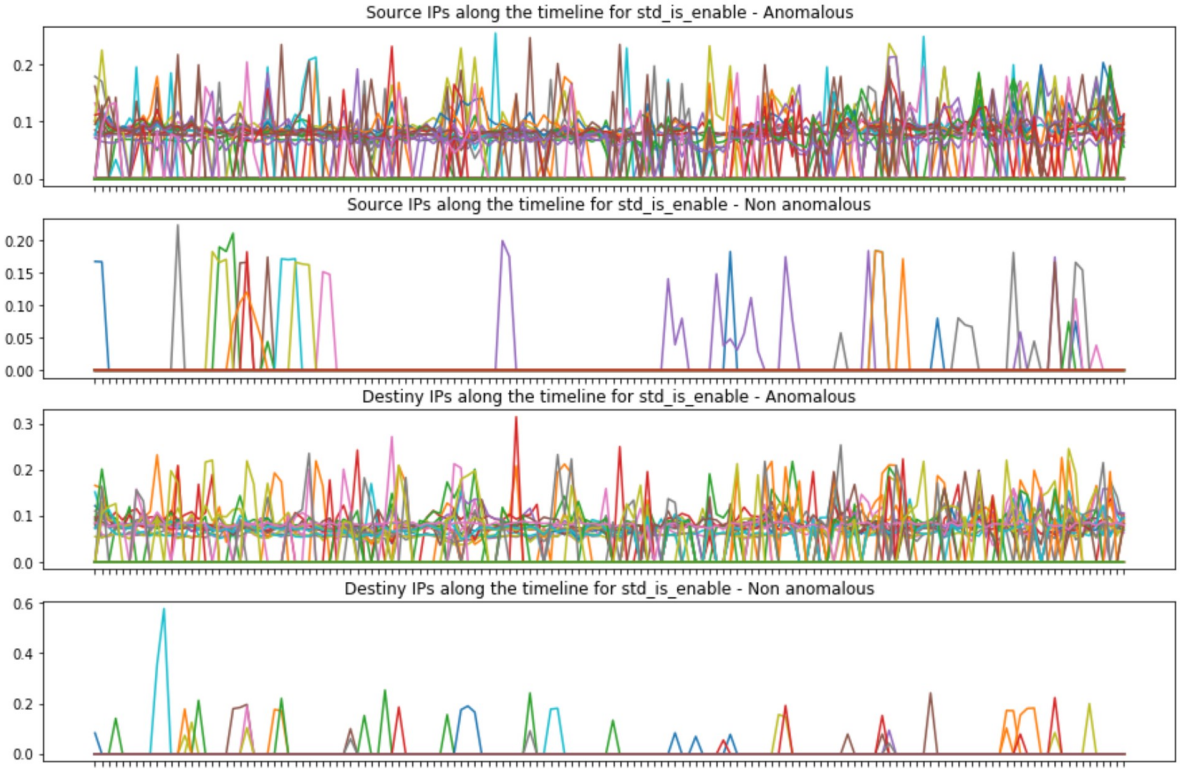
- Overall, most anomalous connections have a higher is_busybox value

is_enable

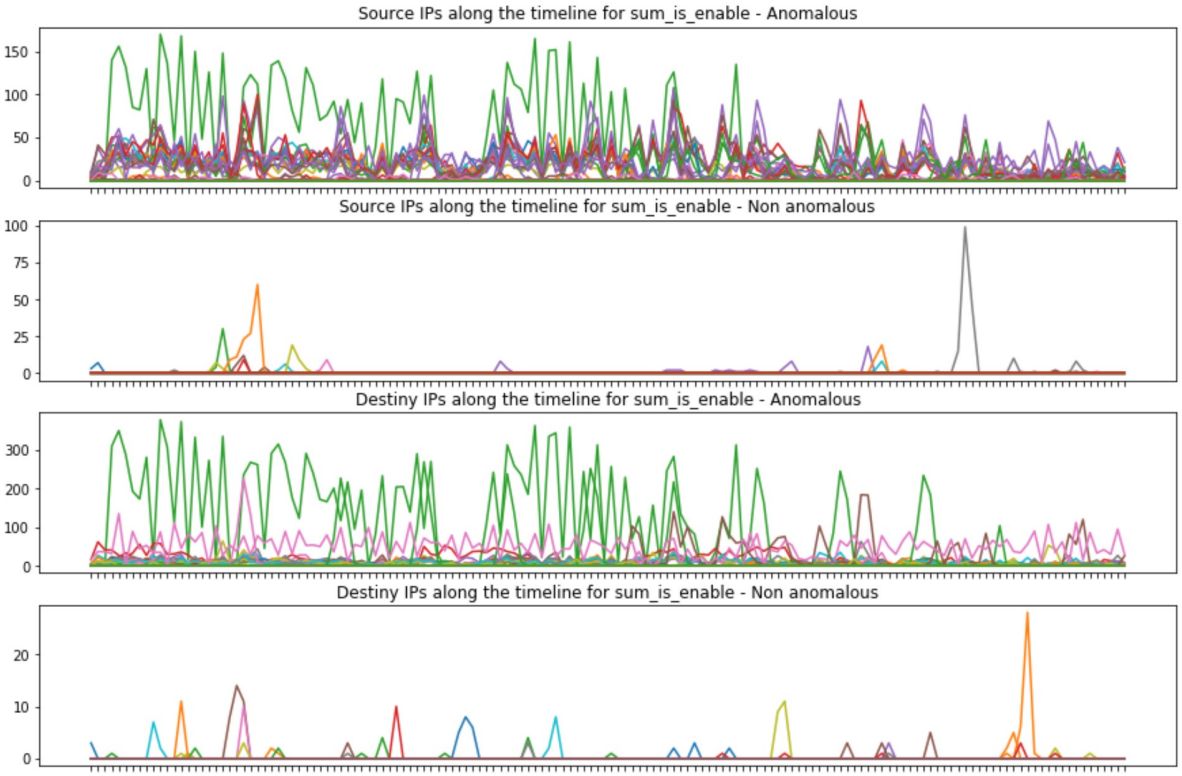
In [184]:
```
plot_all_ips(data_5T, data_5T_sample, 'avg_is_enable')
```



In [185]:
```
plot_all_ips(data_5T, data_5T_sample, 'std_is_enable')
```



In [186]:
```
plot_all_ips(data_5T, data_5T_sample, 'sum_is_enable')
```



is_enable Summary

- Might have a higher std than non anomalous
- Some connections have a higher summ tan non anomalous, specially in source connections

is_enable and is_busybox have much more sense on source connections (connections than try to infect) than in destiny connectios, due to the source being the one sending the payload

In [ ]:

Bibliografía

[1] Rebecca Bace and Peter Mell. Intrusion detection systems. *National Institute of Standards and Technology (NIST)*, 2001.

[2] Giovanni Vigna and Christopher Kruegel. Host-based intrusion detection. 2005.

[3] Stefan Axelsson. Research in intrusion-detection systems: a survey. *Department of Computer Engineering, Chalmers University of Technology*, 1998.

[4] Pedro Garcia-Teodoro, Jesus Diaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, 28(1-2):18–28, 2009.

[5] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 44(1.2):206–226, 2000.

[6] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[7] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[8] Simon Haykin. *Neural networks*, volume 2. Prentice hall New York, 1994.

[9] Qiong Liu and Ying Wu. *Supervised Learning*. Springer US, Boston, MA, 2012.

[10] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning – with Applications in R*, volume 103 of *Springer Texts in Statistics*. Springer, 2013.

[11] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[12] Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. 96:148–156, 1996.

[13] Ben Gorman. A kaggle master explains gradient boosting, 2017. Last accessed 27 March 2019.

[14] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. pages 785–794, 2016.

[15] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. pages 3146–3154, 2017.

[16] Home credit default risk, 2018. Last accessed 27 March 2019.

[17] Corporación favorita grocery sales forecasting, 2018. Last accessed 27 March 2019.

[18] Google analytics customer revenue prediction, 2019. Last accessed 27 March 2019.

[19] How to calculate principal component analysis (pca) from scratch in python, 2018. Last accessed 04 April 2019.

[20] Covariance matrix, 2019. Last accessed 04 April 2019.

[21] Shyam Boriah, Varun Chandola, and Vipin Kumar. Similarity measures for categorical data: A comparative evaluation. In *Proceedings of the 2008 SIAM international conference on data mining*, pages 243–254. SIAM, 2008.

[22] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, 1996.

[23] Anonymous. Inferring and characterizing internet-scale iot probing campaigns by leveraging a novel data dimensionality reduction technique. *ACSAC*, 2018.

[24] Andrew Ng et al. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011.

[25] Analizador de paquetes, 2019. Last accessed 03 June 2019.

[26] Jon Postel. Transmission control protocol. Technical report, 1981.

[27] Jon Postel. Internet protocol. Technical report, 1981.

[28] Feature engineering: What powers machine learning, 2019. Last accessed 04 June 2019.

[29] Tcpdump/libcap public repository, 2019. Last accessed 04 June 2019.

[30] Anatomy of an iot malware attack, 2017. Last accessed 04 June 2019.

[31] What is happening on 2323/tcp?, 2016. Last accessed 04 June 2019.

[32] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422. IEEE, 2008.

[33] Convolutional neural network on a structured bank customer data, 2018. Last accessed 08 June 2019.

[34] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. *CoRR*, abs/1602.04938, 2016.