Analisis de Datos para Ciberseguridad: Trabajo práctico 1

Ph. D. Saúl Calderón Ramírez
Instituto Tecnológico de Costa Rica,
Escuela de Ingeniería en Computación, Programa de Ciencias de Datos,
PAttern Recongition and MAchine Learning Group (PARMA-Group)

28 de julio de 2025

Fecha de entrega: Domingo 17 de Agosto.
Entrega: Un archivo .zip con el código fuente LaTeX o Lyx, el pdf, y un notebook Jupyter, debidamente documentado. A través del TEC-digital.
Modo de trabajo: Grupos de 3 personas.

Resumen

En el presente trabajo práctico se introducir los arboles de decision por medio de su implementación para resolver un problema práctico en ciberseguridad.

En el presente trabajo practico se utilizara el *dataset KDD99* `https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`. El conjunto de datos KDD 1999 es uno de los más utilizados para entrenar y evaluar sistemas de detección de intrusos (IDS). Está basado en simulaciones de tráfico de red en un entorno militar y contiene conexiones etiquetadas como normales o como uno de varios tipos de ataques.

1. Categorías de características (features): Las 41 características del dataset se dividen en cuatro grupos principales:
   a) Características básicas (Basic features): Información general de la conexión como duración, protocolo, servicio, estado de la conexión. Ejemplos: duration, protocol_type, service, flag.
   b) Características de contenido (Content features): Analizan el contenido de la conexión para detectar intentos de acceso no autorizado. Ejemplos: num_failed_logins, logged_in, root_shell, num_compromised.
   c) Características de tráfico basado en tiempo (Time-based traffic features): Estadísticas de conexiones en una ventana de tiempo (por ejemplo, en los últimos 2 segundos). Ejemplos: count, srv_count, serror_rate, srv_serror_rate.

d) Características de tráfico basado en host (Host-based traffic features): Estadísticas de conexiones hacia el mismo host en una ventana más amplia.- Ejemplos: dst_host_count, dst_host_srv_count, dst_host_same_srv_rate.

2. Las conexiones se etiquetan como normales o como uno de los siguientes tipos de ataque:

   a) DoS (Denial of Service): Saturar el sistema para que no responda a usuarios legítimos. Ej: smurf, neptune.

   b) Probe: Recolección de información sobre la red. Ej: satan, portsweep.

   c) R2L (Remote to Local): Acceso no autorizado desde una máquina remota. Ej: guess_passwd, warezclient.

   d) U2R (User to Root): Escalada de privilegios desde una cuenta local. Ej: buffer_overflow, rootkit.

Para limitar el alcance del trabajo practico, se analizara y creara un modelo de las conexiones normales y de tipo *backdoor*, tal como se fija en el codigo base.

1. (30 puntos) Análisis descriptivo de las características en el conjunto

1. **(15 puntos)** Por cada característica del espacio de entrada, compute los siguientes momentos estadísticos (usando la implementación de las funciones correspondientes en pytorch, o la implementacion realizada por su equipo): media, desviación estándar, inclinación y kurtosis. Presente tales momentos de forma compacta y analice los resultados.

2. **(15 puntos)** Por cada característica, grafique el histograma para la categoría ataque (*backdoor*) y normal, y compute la distancia de Jensen-Shannon entre ambas aproximaciones de las densidades. Realice los gráficos para facilitar la comparación, y analice los resultados.

2. (40 puntos) Implementación de la clasificación multiclase con árboles de decisión (60 puntos)

1. Para realizar la clasificación entre los ataques *backdoor* y las conexiones normales, su equipo implementará los árboles de decisión. El código provisto define las clases *CART* y Node_*CART*, las cuales permiten construir un CART binario. Cada nodo del arbol tiene atributos como el *feature,* el umbral, y el coeficiente de *gini* (o la entropía) de la partición definida en tal nodo. Además define el atributo *dominant_class* para el nodo, el cual es el resultado de calcular la clase con mayor cantidad de apariciones en

la partición que define al nodo. Finalmente el código incluye la funcionalidad para generar un archivo *xml* (el cual se puede abrir en cualquier navegador web), para representar fácilmente el árbol.

a) **(10 puntos)** Implemente el método *calculate_gini(data_partition_torch, num_classes = 2),* el cual calcule el coeficiente de gini para el conjunto de datos recibido en un tensor de *pytorch.* Para ello utilice la definición inidicada en el material del curso. Realice una implementacion matricial (prescindiendo de estructuras de repeticion al maximo). Comente la implementacion, detallando cada funcion utilizada en la documentacion externa.

$$E_{\text{gini},\rho}\left(\tau_d\right) = 1 - \sum_{k=1}^{K} a_k^2.$$

1) Diseñe e implemente al menos 2 pruebas unitarias para esta función.

b) **(20 puntos)** Implemente los metodos *select_best_feature_and_thresh(data_torch, num_classes = 2) y create_with_children,* de la clase *Node_CART.* Este método recibe como parámetros el conjunto de datos en un tensor tipo torch a analizar. El método debe probar de forma extensiva todos los posibles features y sus correspondientes umbrales en los datos recibidos, hasta dar con el menor coeficiente ponderado de gini (o la minima entropía, dependiendo de la función de error a utilizar). **Utilice indexación lógica para evitar al máximo el uso de estructuras de repetición tipo *for***. Solamente puede usar estructuras de repetición para iterar por los *features* y posibles umbrales dentro del conjunto de datos. Recuerde que para evaluar una posible partición, es necesario calcular el coeficiente de gini ponderado sugerido para decidir el feature y umbral óptimos es:

$$\overline{E}_{\text{gini}}\left(\tau_d, d\right) = \frac{n_i}{n} E_{\text{gini}}\left(D_i\right) + \frac{n_d}{n} E_{\text{gini}}\left(D_d\right).$$

Comente la implementacion, detallando cada funcion utilizada en la documentacion externa.

1) Diseñe e implemente al menos 2 pruebas unitarias para esta función.

c) **(10 puntos)** Implemente la función *test_CART* la cual evalúe un CART previamente entrenado para un conjunto de datos D representado en un tensor. Calcule la tasa de aciertos (*accuracy*), definida como:

$$a = \frac{c}{n}$$

donde c corresponde a las estimaciones correctas, para tal conjunto de datos y retornela. Comente la implementacion, detallando cada funcion utilizada en la documentacion externa.

1) Diseñe e implemente al menos 2 pruebas unitarias para esta función.

3. (30 puntos) Evaluación del CART

1. **(10 puntos)** Evalúe el CART implementado con el conjunto de datos provisto usándolo como conjunto de datos de entrenamiento y prueba. Reporte la tasa de aciertos y el F1-score promedio de todas las clases, obtenida e incluya el código de la evaluación. Pruebe con una profundidad máxima de 3 y 4 nodos, siempre con mínimo 2 observaciones por hoja.

2. **(20 puntos)** Para una profundidad máxima de 2 y 3 nodos: evalúe el CART implementado usando 10 particiones aleatorias del conjunto de datos, con un 70 % del conjunto de datos como conjunto de datos de entrenamiento, y el restante 30 % como conjunto de datos de prueba. Reporte una tabla con la tasa de aciertos, F1-score y tiempo de ejecución (para el entrenamiento y la evaluacion, por separado), con el promedio y desviación estándar para las 10 corridas.

   a) Grafique el arbol obtenido para la corrida con mejor F1-score. Analice la estructura de ese arbol usando lo observado en la primer sección. Como se podria hacer mas eficiente la construccion del arbol y su evaluacion, usando la distancia de Jensen-Shannon previamente implementada?