Clasificación de registros desplazables no lineales mediante aprendizaje automático

Trabajo práctico para el curso de Doctorado en Ciencias informáticas de la Facultad de Informática de la Universidad Nacional de la Plata (UNLP):

ANÁLISIS INTELIGENTE DE DATOS EN ENTORNOS DE BIG DATA

Profesor: Dr. José Ángel Olivas Varela

Alumno: Lic. Ariel Maiorano

Mayo de 2018

Resumen. Se propone un experimento, utilizando diferentes técnicas de aprendizaje automático, para medir la eficacia de clasificar configuraciones de registros desplazables no lineales -o NLFSRs, por sus siglas en inglés: *Non Linear Feedback Shift Registers*-, de tamaños predeterminados, en dos grupos o clases; siendo el objetivo identificar aquellos con período de recursión máximo.

1 Introducción

Mediante la aplicación de diferentes técnicas de aprendizaje automático o *machine learning*, -entre otras, las que implementan redes de neuronales-, utilizando herramientas de software libre, se propone este experimento acotado como una primera prueba para determinar la eficacia de las técnicas implementadas en la clasificación de todas las configuraciones posibles de registros de desplazamiento no lineales, de tipos específico, y de tamaños predeterminados.

Se pretende constatar si los diferentes modelos a entrenar podrán diferenciar entre registros desplazables no lineales de período de recursión máximo.

Este trabajo práctico se desarrolla en cumplimiento de los requisitos del curso "Análisis Inteligente de Datos en Entornos de Big Data" dictado por el Dr. José Ángel Olivas Varela, y en complemento a la tesis doctoral, en preparación, del alumno, cuyo alcance incluye la implementación de un marco o *framework* para la aplicación de técnicas criptoanáliticas. Aunque en una versión inicial e incompleta, el trabajo adelantado a la fecha puede accederse desde: https://github.com/gicsi/mac. Por otra parte, los datos y scripts desarrollados para este trabajo, de ser autorizado, serán publicados en https://github.com/arielmaiorano/curso-analisis-datos-2018. Aunque el presente experimento trate casos muy específicos y acotados, el análisis del período de recursión de registros desplazables como componentes de algoritmos de cifrados simétricos, del tipo en flujo, o *stream ciphers*, está comprendido dentro de una de las líneas de investigación del alumno como participante de un grupo de investigación [1, 2].

1.1 *Machine learning* y criptología

Ya en el año 1991, Ronal Rivest –como suele decirse, la "R" en RSA-, publicó un artículo titulado *Cryptography and Machine Learning* [3]. Allí estudió la relación entre los campos de la criptografía y el aprendizaje automático, con énfasis en cómo estos campos se han aportado ideas y técnicas entre sí. También propuso nuevas ideas, para una, en sus palabras, futura fertilización cruzada.

Considerando específicamente lo relativo al criptoanálisis como rama de la criptología a la par de la criptografía, aunque en la literatura sea más sencillo encontrar trabajos relacionados a la aplicación de *SAT-solvers* -como por ejemplo en [6], donde se investigan seis esquemas de cifrado autenticado de la competencia CAESAR, apuntando a los ataques de recuperación del estado utilizando un SAT como herramienta-, se encuentran sin embargo ejemplos como [7], en el cual utilizan redes neuronales recurrentes (RNN) y demuestran que éstas pueden aprender algoritmos de descifrado -las asignaciones de texto-en-claro a texto-cifrado, para tres cifradores poli alfabéticos (Vigenere, Autokey y Enigma).

En este último trabajo demuestran además que una RNN con una memoria Long-Short-Term (LSTM) de 3000 unidades puede aprender la función de descifrado de la máquina Enigma, y argumentan que su modelo aprende representaciones internas eficientes de los cifrados mencionados.

En otro trabajo de publicación reciente [8], se estudia la utilización de redes neuronales recurrentes, también utilizando LSTM, para la implementación de las mismas funcionalidades que proveen los *SAT-solvers*.

En lo que a este trabajo respecta, es conveniente destacar que es deseable que el período o el límite inferior del período de una secuencia pseudoaleatoria para aplicaciones criptográficas sea tan elevado como sea posible. Por ejemplo, en un cifrado de flujo, o *stream cipher*, la longitud de una secuencia debe ser la misma que la longitud de un mensaje, y la secuencia nunca debe repetirse. Además, la complejidad lineal de una secuencia debe ser lo suficientemente alta, para que un adversario no pueda generar la totalidad secuencia a partir de un segmento parcialmente conocido.

1.2 Registros desplazables lineales y no lineales

Se pueden encontrar en la literatura numerosos algoritmos y técnicas criptográficas para generar secuencias pseudoaleatorias, éstas técnicas incluyen a los registros desplazables, o de desplazamiento retroalimentados (FSR por sus siglas en inglés, Feedback Shift Registers). La configuración de registros, o su configuración de retroalimentación, o también, su función de retroalimentación, se puede clasificar en dos categorías, a saber, registros de desplazamiento de realimentación lineal (LFSR, por sus siglas en inglés, Linear Feedback Shist Registers) y registros de desplazamiento de realimentación no lineal (o NLFSR, Non Linear Feedback Shift Registers) [5].

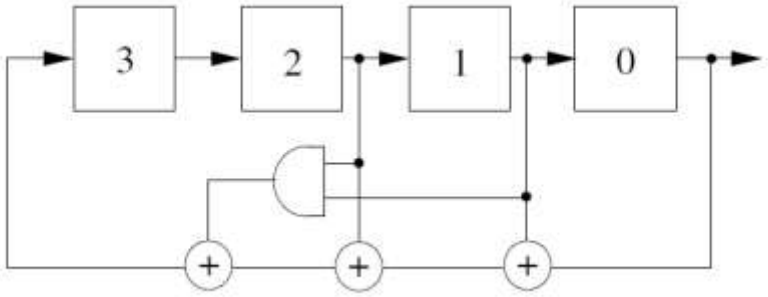

Figura 1. Ejemplo de un NLFSR de 4 bits.

Este trabajo se basa en los resultados obtenidos por Elena Dubrova [4], siguiendo lo allí planteado respecto a que el LFSR es uno de los mecanismos más utilizados para generar secuencias seudoaleatorias. Tiene numerosas aplicaciones, incluida la detección y corrección de errores, compresión de datos y criptografía. La investigación sobre los LFSRs comenzó a principios de los años 60 [5] y continuó activamente. Hoy los LFSR son un tema bien entendido. La mayoría de los problemas fundamentales relacionados con LFSRs están resueltos. Se sabe cómo sintetizar un LFSR con un máximo período -uno tiene que usar un polinomio generador primitivo-. Las propiedades estadísticas de secuencias generadas por LFSR se han caracterizado por los postulados de Golomb [5].

En cambio el NLFSR, que es una generalización del LFSR en donde el estado actual es una función no lineal del estado anterior, no cuenta con el mismo sustento teórico. Probablemente el problema abierto más importante es el de encontrar un procedimiento sistemático para construir NLFSRs con una longitud de período garantizado. El problema general es difícil porque parece no haber una teoría algebraica simple que lo apoye.

En el artículo mencionado [4] se presenta una lista completa, obtenida por búsqueda exhaustiva, de NLFSRs de n bits con el período $2^n - 1$ (aunque para no-lineales el período de recursión máximo sería 2^n, se considerarán ambos como máximos), para $n < 25$, para los siguientes tipos de funciones de retroalimentación de segundo grado algebraico:

1. $f(x_0,x_1,\ldots,x_{n-1}) = x_0 + x_a + x_b + x_c \cdot x_d$
2. $f(x_0,x_1,\ldots,x_{n-1}) = x_0 + x_a + x_b \cdot x_c + x_d \cdot x_e$
3. $f(x_0,x_1,\ldots,x_{n-1}) = x_0 + x_a + x_b + x_c + x_d + x_e \cdot x_h$

Donde a, b, c, d, e, h pertenecen a $\{1,2,\ldots,n-1\}$, "+" es el XOR (adición módulo 2) y "·" es la operación AND (multiplicación módulo 2).

2 Experimento

A partir de las configuraciones de NLFSRs obtenidas por Dubrova [4] para tipos específicos, se procedió a generar todas las combinaciones posibles para generar los dos grupos o clases que serán provistos como entrada para el entrenamiento de los clasificadores: aquellos de período de recursión máximo y el resto.

El objetivo del experimento será además determinar si, en caso de que el entrenamiento y validación de los clasificadores fuera aceptable, se podría extrapolar sus estimaciones para registros de longitudes mayores.

Considerando el estado del arte citado por ejemplo en [7, 10], donde se destaca su capacidad para aprender tareas algorítmicas, se utilizarán modelos de RNNs, por sus siglas en inglés, Recurrent Neural Networks, utilizando LSTM (Long-Short Term Memory) comparativamente a otros modelos de redes neuronales modernos.

En lenguaje de programación Python se implementaron, utilizando tensorflow [9] –librería que provee lo necesario para utilizar redes neuronales profundas, o más bien *Deep Learning* en general- a través del framework keras [10] –que facilita la utilización de la primera-, a su vez mediante Scikit-learn [11, 12] –que permite la implementación de otros tipos de modelos y diversos métodos de validación-; diferentes pruebas con los mismos conjuntos de datos.

Estos conjuntos fueron generados a partir de la información en [4], para los casos positvos, y mediante generación automática para los negativos (resto de configuraciones posibles de NLFSRs de período de recursión no máximo). Considerando registros de longitudes de 4 a 8 inclusive, resultan 121 positivos sobre un total de 4585. Se repartieron en conjuntos estratificados (de acuerdo al campo que determina el grupo o clase por supuesto) de 70% para el entrenamiento y el 30% restante para el *testing*.

Este experimento se limitó a lo indicado, no contemplado conjuntos de validación ni alternativas posibles de evaluación como validación cruzada o *Cross-validation*.

Los modelos seleccionados para experimento fueron el de "Vecinos más próximos", (o *Nearest Neighbors*), "Bayesiano ingenuo" (o *Naive Bayes*), "SVM Lineal" (o *Linear SVM*, por sus siglas en inglés de *Support Vector Machine*), "Árbol de decisión" (o *Decision Tree*), Bosques aleatorios (o *Random Forest*), una red neuronal simple, y, como se adelantó, una red neuronal recurrente, o *Recurrent Neural Network*, utilizando LSTM (*Long-Short Term Memory*).

Los parámetros no han sido optimizados, y fueron establecidos de acuerdo a los valores por defecto recomendados para este tipo de tareas. Además, las redes neuronales han sido entrenadas con un número limitado de iteraciones: en ambos casos, 500. La primera cuenta con dos etapas o *layers* ocultos, de 12 y 8 nodos o neuronas (al margen por supuesto de los 8 correspondientes a la etapa de entrada). La segunda de las redes neuronales cuenta con una única etapa de 100 nodos ocultos. En ambos casos la etapa de salida consta por supuesto de un nodo (que indica 0 para períodos no máximos y 1 para máximos) con función de activación *sigmoid*.

Los detalles de configuración y resultados obtenidos por cada uno de los modelos evaluados se incluyen en los materiales y código fuente a entregar junto a este trabajo y luego a publicar se correspondiera de acuerdo a lo adelantado en la introducción, y se presentan resumidos en la tabla siguiente:

Nombre de La técnica	**Precisión prom. total**	**Precisión de la clase**	***Recall* de la clase**	***f1-score* de la clase**
Vecinos más próximos	0.95	0.20	0.06	0.09
Bayesiano ingenuo	0.96	0.14	0.19	0.16
SVM lineal	0.95	0.00	0.00	0.00
Árbol de decisión	0.95	0.00	0.00	0.00
Bosques aleatorios	0.98	1.00	0.03	0.05
Red neuronal (NN)	0.97	0.56	0.14	0.22
NN Recurrente con LSTM	0.95	0.16	0.08	0.11

Tabla 1. Resultados obtenidos por tipo de modelo utilizado. Las métricas por clase se refieren a la que identifica a los períodos de recursión máximos.

3 Conclusiones

En este trabajo se presentaron los resultados de haber aplicado diferentes técnicas de aprendizaje automático con el objetivo de clasificar NLFSRs.

El desbalance del conjunto de datos crecerá exponencialmente en tanto crezca la longitud de los registros a evaluar, por lo que este problema no podría ser resuelto sólo con un *dataset* del mismo tipo pero con mayor cantidad de muestras. Este desbalance, es decir, que la cantidad de registros positivos represente menos del 5% del total de registros hace a la tarea de clasificación más compleja. Por esto, aunque el porcentaje promedio de precisión es muy alto, los específicos para el problema de identificar aquellos de período de recursión máximo son muy bajos.

Dado lo limitado de los experimentos realizados y sus resultados poco alentadores, y si bien en la literatura no se encuentran ejemplos de técnicas de aprendizaje automático aplicadas al criptoanálisis de algoritmos modernos –al margen de la aplicación a cifradores clásicos [7] o a las recientes publicaciones de redes neuronales que podrían utilizarse como *SAT-solvers* [8]-; la aplicación de estas técnicas para tareas específicas podría resultar de interés al menos como una herramienta complementaria de criptoanálisis, en caso de lograr una eficacia aceptable mediante la utilización de otros tipos de modelos y/o valiéndose de la posibilidad de manejar volúmenes mayores de datos en entornos de *big data*.

Referencias

1. Castro Lechtaler, A.; Cipriano, M.; García, E.; Liporace, J.; Maiorano, A.; Malvacio, E.; Tapia, N.; Dulio, N.; Pérez, P. Secuencias seudoaleatorias para criptología. (2016). XVIII Workshop de Investigadores en Ciencias de la Computación.
2. Castro Lechtaler, A.; Cipriano, M.; García, E.; Liporace, J.; Maiorano, A.; Malvacio, E. Análisis y estudio del período de recursión de los algoritmos Trivium y Trivium Toy. (2014). XVI Workshop de Investigadores en Ciencias de la Computación.
3. Rivest, R.L.: Cryptography and machine learning. In: Matsumoto, T., Imai, H., Rivest, R.L. (eds.) ASIACRYPT 1991. LNCS, vol. 739, pp. 427–439. Springer, Heidelberg (1993). https://people.csail.mit.edu/rivest/pubs/Riv91.pdf.
4. E. Dubrova. A List of Maximum Period NLFSRs", Report 2012/166, Cryptology ePrint Archive, 2012. http://eprint.iacr.org/2012/166.pdf.
5. S.W. Golomb, Shift Register Sequences, Aegean Park Press, Laguna Hills, CA, USA, 1981.
6. Dwiedi, Ashutosh, Klouček, Miloš, Morawiecki, Paweł, Nikolić, Ivica & Pieprzyk, Josef, Wójtowicz, Sebastian. (2017). SAT-based Cryptanalysis of Authenticated Ciphers from the CAESAR Competition. https://www.researchgate.net/publication/317590073_SAT-based_Cryptanalysis_of_Authenticated_Ciphers_from_the_CAESAR_Competition.
7. Greydanus, Sam. Learning the Enigma with Recurrent Neural Networks. (2017). arXiv:1708.07576. https://arxiv.org/abs/1708.07576.
8. Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, David L. Dill. Learning a SAT Solver from Single-Bit Supervision. (2018). arXiv:1802.03685 [cs.AI]. https://arxiv.org/abs/1802.03685.
9. Martín Abadi, *et. al*. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
10. François Chollet. Keras. 2015. Github repository. https://github.com/fchollet/keras.
11. Pedregosa *et al*.Scikit-learn: Machine Learning in Python. JMLR 12, pp. 2825-2830, 2011.
12. Buitinck *et al*. API design for machine learning software: experiences from the scikit-learn project. 2013. http://scikit-learn.org/.