Seguridad en contraseñas

Cota Martinez Guillermo Oswaldo

30 de enero de 2021

Resumen

La seguridad en las contraseñas es un tema de vital importancia al requerir que la información no sea accedida por cualquier entidad. Por ello se tomará en cuenta una base de datos de alrededor de $684,500$ contraseñas para realizar un análisis de distintas contraseñas y su seguridad con respecto a la frecuencia histórica de su uso.

Finalmente, se propondrá un algoritmo que reciba como entrada una contraseña y que determine su seguridad con una métrica preestablecida.

1. Objetivo

El objetivo del proyecto es poder clasificar cadenas alfanuméricas según su seguridad criptográfica, por medio de distintos métodos como $k-$vecinos, redes neuronales simples asi como redes neuronales recurrentes.
La clasificación predice una etiqueta y los problemas incluyen problemas binarios de "sí o no", así como problemas de clasificación múltiple como "si esto es bueno, malo o promedio". Con la clasificación, las respuestas correctas deben etiquetarse para que su algoritmo pueda aprender de ellas.
Por tanto, por medio de los métodos ya mencionados, se darán ciertas etiquetas que mencionarán su seguridad criptográfica.

2. Marco Teórico

2.1. Las redes neuronales artificiales.

Las redes neuronales artificiales son un modelo computacional inspirado en el comportamiento observado en su homólogo biológico. Consiste en un conjunto de unidades, llamadas neuronas artificiales, conectadas entre sí para transmitirse señales. La información de entrada atraviesa la red neuronal (donde se somete a diversas operaciones) produciendo unos valores de salida.
Cada neurona está conectada con otras a través de unos enlaces. En estos enlaces el valor de salida de la neurona anterior es multiplicado por un valor de peso. Estos pesos en los enlaces pueden incrementar o inhibir el estado de activación de las neuronas adyacentes. Del mismo modo, a la salida de la neurona, puede existir una función limitadora o umbral, que modifica el valor resultado o impone un límite que no se debe sobrepasar antes de propagarse a otra neurona. Esta función se conoce como función de activación.

2.2. $K-$vecinos

La idea básica sobre la que se fundamenta este paradigma es que un nuevo caso se va a clasificar en la clase más frecuente a la que pertenecen sus K vecinos más cercanos. El paradigma se fundamenta por tanto en una idea muy simple e intuitiva, lo que unido a su fácil implementación ace que sea un paradigma clasificatorio muy extendido. Este algoritmo es supervisado y esta basado en instancias en la que el algoritmo memoriza instancias de entrenamiento que son usadas como 'base de conocimiento' para la fase de predicción.

Es un sistema de clasificación basado en la comparación de instancias nuevas con instancias presentes en el juego de datos de entrenamiento. La construcción de grupos se realiza a partir del propio juego de datos de entrenamiento, tomando como referencia el parámetro k indicado por el investigador. El algoritmo de los K vecinos más cercanos es uno de los algoritmos más simples que existen que muestran la esencia del aprendizaje basado en instancias. Este algoritmo asume que todas las instancias corresponden a puntos que se encuentran en un espacio de dimensión n. El vecino más cercano de una instancia es definido en términos de la distancia Euclidiana estándar o con alguna otra métrica.

3. Nuestros datos

Los datos que usaremos son una recopilación de contraseñas, disponible en internet, de contraseñas que han sido filtradas de la red en passwords leaks.
Nuestra base de datos consiste en 677,568 renglones $\times$ 3 columnas, contraseña, hash de la contraseña y la cantidad de veces que ha aparecido.
La colección de los csv, tenia dos entradas con datos nulos, así que los eliminamos. Existían valores duplicados en cada documento, que son el conteo de

contraseñas segmentado, por lo cual decidimos sumar las cuentas de todos los duplicados para obtener el número de veces que cada contraseña fue usada, esto nos deja con 677,566 registros.

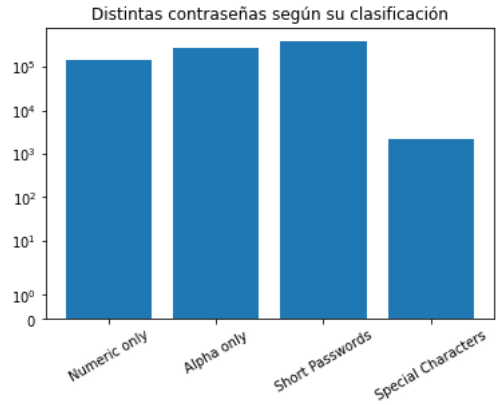
3.1. Exploración de los datos

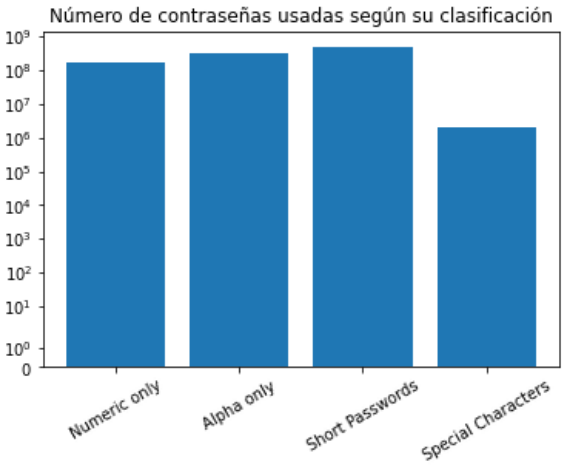
En la exploración de datos, se decidió contar el número de contraseñas distintas y el número total que estas fueron usadas en las contraseñas. La elección de las siguientes características fue elegida de acuerdo a las contraseñas que normalmente son vulneradas, así como la última categoría que corresponde a aquellas contraseñas que tienen carácteres especiales, es decir carácteres no alfanuméricos que son recomendadas para aumentar la seguridad en las contraseñas.

- Contraseñas formadas únicamente por números: 142,340
- Suma del uso de contraseñas formadas por números: 171 260,703
- Contraseñas formadas por letras: 278,189
- Suma del uso de contraseñas formadas por letras: 335 503,274
- Contraseñas de menos de 8 caracteres: 382,452
- Suma del uso de contraseñas cortas: 477 964,360
- Contraseñas con caracteres especiales:2,147
- Suma del uso de contraseñas con caracteres especiales:1 977,061

Las contraseñas mas utilizadas son:

Contraseña	count
123456789	7 016,669
qwerty	3 599,486
111111	2 900,049
12345678	2 680,521
abc123	2 670,319
12345	2 088,998
1234567890	2 075,018
123123	2 048,411
00000	1 832,944
iloveyou	1 462,146
1234	1 143,408
1q2w3e4r5t	1 109,333
qwertyuiop	1 028,295





4. Seguridad en Contraseñas

4.1. Entropía en contraseñas

La entropía de la contraseña predice qué tan difícil sería descifrar una contraseña determinada mediante adivinanzas, descifrado por fuerza bruta, ataques de diccionario u otros métodos comunes. La entropía básicamente mide cuántas conjeturas tendrá que hacer un atacante para adivinar su contraseña. A medida que aumenta la potencia de cálculo, la cantidad de tiempo necesario para adivinar grandes cantidades de contraseñas disminuye significativamente, por lo que es útil hacer ciertas suposiciones en el momento de un cálculo dado en cuanto al número de conjeturas por segundo que puede hacer una computadora.
La cantidad de conjeturas que se necesitan para adivinar al 100 % definitivamente una contraseña o frase de contraseña (es decir, la cantidad de posibles combinaciones de contraseña o frase de contraseña) generalmente tiende a ser una función del tamaño del "grupo de símbolos.ª la potencia de la cantidad de símbolos utilizados.
La entropía de una contraseña estará dada por:

L = Longitud de la contraseña; Número de símbolos en la contraseña
S = Tamaño del grupo de posibles símbolos únicos (a-z,A-Z,0-9,símbolos).
Número de posibles combinaciones = S^L

Entropía = $log_2(S^L)$

Sin embargo, es importante notar que estadísticamente, un ataque de fuerza bruta no requerirá adivinar TODAS las combinaciones posibles para eventualmente alcanzar la permutación correcta. Por lo tanto, tendemos a mirar el número esperado de conjeturas requeridas, que se puede reformular como cuántas conjeturas se necesitan para tener un 50 % de posibilidades de adivinar la contraseña.

Intentos esperados (para 50 %) = $2^{Entropy-1}$

Lamentablemente, esta fórmula solo se aplicaría a los casos más simples. Muchos medidores de contraseñas y formularios de registro en línea complican las cosas al imponer varias restricciones arbitrarias (y desafortunadamente no aleatorias) sobre los patrones permitidos que pueden existir en una contraseña. Los ejemplos incluyen .ªl menos un carácter en mayúscula", .ªl menos un símbolo", etc.

4.2. BruteForce

Un ataque de fuerza bruta es un intento de descifrar una contraseña o nombre de usuario, que consiste en aplicar el método de prueba y error con la esperanza de dar con la combinación correcta finalmente. En función de la longitud y complejidad de la contraseña, descifrarla puede llevar desde unos segundos hasta varios años.
Los diccionarios de contraseñas son la herramienta más básica. Se pueden utilizar diccionarios íntegros y ampliados, con ayuda de caracteres especiales y números, o bien utilizan diccionarios especiales.
En un ataque estándar, se elige un destino y combina posibles contraseñas con el nombre de usuario seleccionado. A este tipo de ataques se les denomina ataques de diccionario.
Como su nombre implica, un ataque de fuerza bruta inverso consiste en invertir la estrategia de ataque, comenzando con una contraseña conocida (como las contraseñas filtradas disponibles en línea) y buscando millones de usuarios hasta que se encuentra una coincidencia.
Muchas herramientas automatizadas pueden descifrar una contraseña consistente en una única palabra que pertenezca a un diccionario en un segundo. Veamos un ejemplo, proveído de howsecureismypassword.net

1. La columna 0 es la cantidad de caracteres de la contraseña
2. La columna 1 es sólo números
3. La columna 2 es sólo minúsculas
4. La columna 3 es sólo minúsculas y mayúsculas
5. La columna 4 es números, mayúsculas y números.
6. La columna 5 es números, mayúsculas, números y símbolos.

0	1	2	3	4	5
4	1 seg	1 seg	1 seg	1 seg	1 seg
5	1 seg	1 seg	1 seg	1 seg	1 seg
6	1 seg	1 seg	1 seg	1 seg	5 seg
7	1 seg	1 seg	25 seg	1 min	6 min
8	1 seg	5 seg	22 min	1 hr	8 hrs
9	1 seg	2 min	19 hrs	3 dias	3 smnas
10	1 seg	58 min	1 mes	7 meses	5años
11	2 seg	1 dia	5años	41años	400años
12	25 seg	3 semanas	300años	2k	34k
13	4 min	1año	16k	100k	2m
14	41 min	51años	800k	9m	200m

5. Metodología

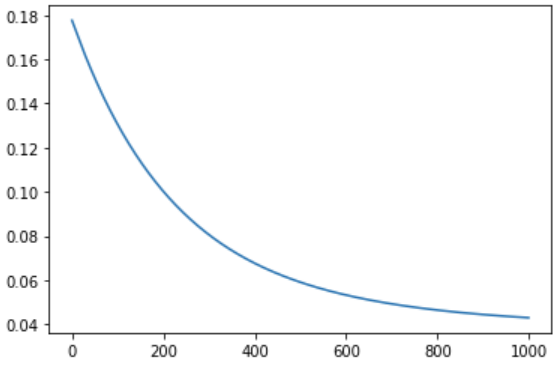
Para realizar nuestro estimador de seguridad, realizamos 3 modelos diferentes para comparar la eficiencia de los mismos: una red neuronal simple, una red neuronal recurrente y K-vecinos.

5.1. Red Neuronal Multicapa

Para la Red Neuronal Multicapa estandarizamos las entradas, las cuales tienen 27 entradas, para hacer esto convertimos las letras de cada contraseña a algún número, esto lo hacemos primero asignando cada carácter de la contraseña en ASCII y después estandarizando los tamaños, es decir dado que la contraseña que tiene mayor número de caracteres es 27, todas las contraseñas las codificaremos a un vector de tamaño 27 de manera que también le asignaremos un carácter que representa 'el espacio vacío' de manera que una vez codificados los distintos caracteres de una contraseña, las dimensiones restantes para sumar 27, corresponderán a un valor -1 que representa que la contraseña no tiene caracteres en esa posición.
Posteriormente realizamos un escalamiento para estas contraseñas con MinMaxScaler. Y asignaremos a cada contraseña un score de seguridad, dividiendo en intervalos uniformes el conjunto según su ordenamiento ya hecho.
Para esto, se asigna una puntuación a las contraseñas en función a su posicionamiento en el escalamiento, siendo aquellos que se encuentran en la cola inferior, los que tienen peor seguridad, mientras los que estén en la cola superior, tendrán un score de 30, la máxima seguridad.

Resultados: Exactitud del modelo: tensor(0.0331)
A pesar de haber hecho iteraciones de entrenamiento y ver una reducción del error, se observó que el error disminuyó debido a la activación de una única neurona, predecir esto, minimiza el error, pues asume que la mejor estrategia es predecir un elemento y la proporción de veces que acierta, es proporcional al tamaño de etiquetas totales. Por lo visto, no es tan útil este modelo pues se inclina a predecir la seguridad en un valor fijo que se encuentra en 12, esto es de esperarse pues no identifica la utilidad de las secuencias y minimiza el error estando en el centroide.

Figura 1. Evolución del error de la red a través de las iteraciones de entrenamiento



5.2. Red Neural Recurrente

Para los algoritmos de la Red Neuronal Recurrente, nos encontramos con una problemática adicional; la complejidad de trabajar con un dataset tan grande y trabajar con la seguridad del password es bastante elevada, nos importa el orden en que estén los caracteres, así que tienen que reconocer qué carácter está en cada posición y qué carácter es.

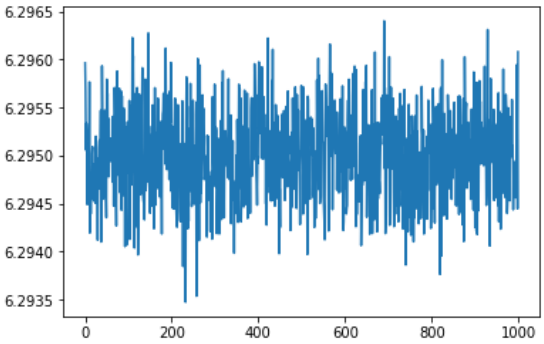
Para eso, hicimos uso de One-Hot Encoding para dimensiones muy grandes.
Resulta que podemos identificar cada letra como un carácter del One-Hot Encoding, es decir, identificamos a=[1,0,0,....] b=[0,1,0,0,..] y así sucesivamente para los 86 caracteres diferentes. Cada letra de un password estará dado por un vector de dimensión 8.
En total la contraseña más larga tenía 27 caracteres, entonces para estandarizar el tamaño que cada password representa, estamos en dimensión 86*27. Así logramos que al modelo le importara el orden de las contraseñas. Dentro de esos 86 caracteres, 1 representa el espacio vacío, que usaremos para estandarizar el tamaño, es decir, como no todas las contraseñas tienen la misma longitud, pues hay un carácter que representa que ya no se escribió nada después de cierta longitud.

Esto puede volverse un problema ya que tenemos 27*86*600,000 contraseñas, las cuales tenemos que pasar a One-Hot Encoding, hacerlo para todas y almacenar los valor involucra una cantidad excesiva de RAM así que creamos una función que recibe una contraseña y la convierte en One-Hot Encoding, aunque esto hace que aumentará la complejidad en el tiempo.

Para recibir un input en nuestro modelo recurrente, pasaremos el input por una función que convierte el password a tensor. Ya que convertimos nuestra contraseña 'abc' al tensor correspondiente, es importante recordar que sera un tensor de tres vectores, puedes ser la entrada de nuestra red.
Nuestra RRN recibira un input a la vez, es decir primero recibe la 'a' después la 'b' y por último la 'c',

Para ahorrar un poco de poder de cómputo, redujimos el espacio de seguridad, antes habiendo 30 seguridades posibles y ahora hay 10.
Resultados: Con el modelo planteado en esta sección, se hicieron dos pruebas: La primera fue una red neuronal recurrente con 128 neuronas en la capa oculta y 128 neuronas en el tensor de estados que iba de un estado a otro usando el gradiente estocásitco . Con esta se obtuvo un 9.6 % de precisión. Para el segundo experimento se agregó una capa oculta adicional antes de llegar a la capa que conecta con el estado oculto, adicionalmente se eligió agrupar los datos de entrenamiento en minibatches para ajustar el gradiente, a pesar de esto, el comportamiento de la red no cambió, pues el error en las dos redes no se vió tan afectado y mostraron comportamientos sin alguna afinidad al decremento, como se muestra en la figura.

Figura 2. Entrenamiento a lo largo del tiempo de una red neuronal recurrente definida para este problema usando función de error cuadrático



5.3. K-vecinos

El tercer experimento fue usar la metodología conocida como K-vecinos, un algoritmo de clasificación el cuál tiene como objetivo la clasificación de vectores de acuerdo a su distancia respecto a los vectores de entrenamiento, siendo los $k-$más cercanos, aquellos que el algoritmo tomará en cuenta, enseguida, la mayoría de etiquetas de los $k-$vecinos más cercanos, será la elegida para el vector a clasificar. Este es un método que depende principalmente de dos parámetros: El primero es el número k sobre el cuál se tomarán en cuenta los vecinos. El segundo es la métrica definida para la clasificación del vector. Recordemos que distintas métricas, inducen distintas vecindades, es por ello que en el problema de clasificación usamos distintos parámetros para comparar su rendimiento.
Para la clasificación, los datos requirieron un tratamiento similar al del modelo de la red neuronal multicapa y consistió en una codificación de cada contraseña en un vector de dimensión 27 y cada carácter que componía la contraseña fue codificado a su valor en ASCII, representando al espacio vacío con un valor -67, el valor no tiene algún significado y solo fue usado para expresar que es un valor distinto que

no representa algún carácter. Así fue un algoritmo de $k-$vecinos en 27 dimensiones y las etiquetas, nuevamente corrieron del 0-10.

El primero de los modelos usó 5 vecinos con medida euclideana.
El segundo experimento requirió de 50 vecinos usando la medida de Hamming, esto pues esta métrica es más efectiva al verificar distancias entre vectores que representan valores cualitativos. Así, esta métrica es más adecuada para el caso de estudio.
El último experimento consistió en el mismo expreimento pasado pero con métrica euclidea para comprobar diferencias.

5.4. Resultados

La exactitud de los modelos fueron las siguientes:

Modelo	Vecinos	Métrica	Precisión
1	5	Euclideana	23.39 %
2	50	Hamming	21.02 %
3	50	Euclideana	23.69 %

Enseguida se calculó el promedio del error cuadrático entre las predicciones hechas por cada modelo y las etiquetas correctas, resultando el menor error en promedio al del tercer modelo, siendo esta un error de 2,25.

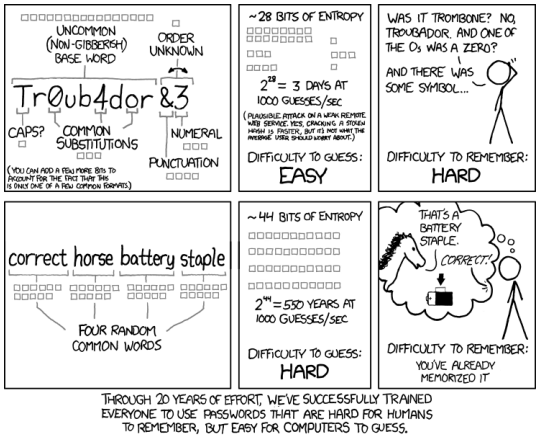
6. Conclusiones

La clasificación de contraseñas con el uso de esta base de datos es retador y por observar de acuerdo a los modelos propuestos, no es una tarea trivial el usar los datos para el problema. Se probaron tres modelos distintos, dos usando redes neuronales y el último usando el modelo de clasififcación mediante $k-$vecinos. A pesar de haber pensado en un inicio que las primeras dos serían la opción viable para la clasificación, resultó contradictorio que hayan resultado tan poco efectivos los métodos, prediciendo en el mejor de los casos hasta un 10 % del total de las muestras. Mientras que con $k-$vecinos, resultó más efectiva la tarea aunque con resultados llegando hasta el 24 % de la precisión deseada. Otros método son recomendados para el seguimiento de esta problemática, aunque quizá la estrategia de estudiar este caso con una base de datos, no sea la adecuada, se propone el entrenamiento de otros modelos usando otras técnicas.

7. Apéndice: Generar contraseñas Seguras Passphrase y Diceware

Uno de los objetivos de este documento, es igual proponer un método para la generación de contraseñas seguras. Este método opta por hacer uso de lo que se conoce como la entropía de la información para generar contraseñas formadas por palabras que tengan alta entropía entre sí y por ende poca susceptibilidad a los patrones o repeticiones de carácteres. Esto lo hacemos mediante la propuesta *Diceware* que usa la aleatoriedad junto a una lista de palabras ordenadas según su 'similitud', esto es que aquellas palabras que tengan baja entropía entre sí, están juntas de modo que al elegir números aleatorios, se tenga menor probabilidad de elegir contraseñas con ínidices cercanos entre sí. Más información acerca de éste método y la lista de palabras utilizadas pueden ser encontradas en: https://theworld.com/ reinhold/diceware.html.
Una vez finalizado el modelo, se decidió probar el rendimiento de este con la seguridad planteada con *Diceware* y se realizó el siguiente experimento:
Se inicializa una contraseña y el modelo elige si tiene la máxima seguridad o no, en caso de que no, se rechaza y se vuelve a correr.
En este experimento, en promedio le tardó al programa, generar 1.104 contraseñas hasta que alguna diera una seguridad máxima, por lo que podemos notar que es un buen modelo para predecir si las contraseñas son seguras.

Figura 3. Descripción de la seguridad de las contraseñas.
Fuente: https://xkcd.com/936/



Referencias

[1] Wikipedia(Red neuronal artificial) https://es.wikipedia.org/wiki/Red_neuronal_artificial

[2] Web(How to calculate entropy) https://generatepasswords.org/how-to-calculate-entropy/

[3] Documentation(Pytorch Docs) https://pytorch.org/docs/stable/index.html

[4] Documentation(Scikit Docs) https://scikit-learn.org/

[5] Book(Michael Nielsen. Neural Networks and Deep Learning) http://neuralnetworksanddeeplearning.com/ 2019.

[6] Book(Goodfellow, Ian, Yoshua Bengio y Aaron Courville (2016). «Deep Learning». En: MIT Press. Cap. Sequence Modeling: Recurrentand Recursive Nets, págs. 367-415) https://www.deeplearningbook.org/contents/rnn.html.