
Ataques al aprendizaje automático

PID_00278570

Gerard Riera Solà

Gerard Riera Solà

Graduado en Matemáticas por la Universidad Politécnica de Cataluña y máster en Matemática Avanzada e Ingeniería Matemática por la misma universidad. Ha desarrollado su actividad académica y profesional fundamentalmente en el campo de la inteligencia artificial. En el ámbito académico, ha aplicado redes neuronales por la detección de arritmias en señales electrocardiográficas. En el ámbito profesional, trabaja a la Fundación Eurecat en el campo de la ciberseguridad. Además, sus competencias se centran en la aplicación del aprendizaje automático para sistemas de autenticación y verificación de la identidad basados en información biométrica (señales biológicas, comportamiento), en su aplicación en sistemas de detección de intrusos y ataques y en la generación de inteligencia sobre amenazas.

El encargo y la creación de este recurso de aprendizaje UOC han sido coordinados por la profesora: Cristina Pérez Solà

Primera edición: febrero 2021

© de esta edición, Fundació Universitat Oberta de Catalunya (FUOC)

Av. Tibidabo, 39-43, 08035 Barcelona

Autoría: Gerard Riera Solà

Producción: FUOC

Todos los derechos reservados

Ninguna parte de esta publicación, incluido el diseño general y la cubierta, puede ser copiada, reproducida, almacenada o transmitida de ninguna forma, ni por ningún medio, sea este eléctrico, mecánico, óptico, grabación, fotocopia, o cualquier otro, sin la previa autorización escrita del titular de los derechos.

Índice

Introducción	5
Objetivos	6
1 Ataques al aprendizaje automático	7
1.1 Descripción de los ataques al aprendizaje automático	7
1.2 Categorización de los ataques al aprendizaje automático	8
1.2.1 El momento del ataque	9
1.2.2 Objetivos del atacante	10
1.2.3 Información del atacante	11
2 Envenenamiento	12
2.1 Modelización de los ataques de envenenamiento	12
2.2 Ataques al aprendizaje supervisado	14
2.2.1 Modificación de etiquetas	14
2.2.2 Ataques de inserción	16
2.3 Ataques al aprendizaje no supervisado	16
2.4 Técnicas de defensa	17
2.4.1 Submuestreo	18
2.4.2 Eliminación de valores extremos	18
2.4.3 Optimización recortada	19
3 Ataques en tiempo de decisión	21
3.1 Modelización de los ataques en tiempo de decisión	21
3.1.1 Ataques a clasificadores binarios	22
3.1.2 Ataques a clasificadores multiclase	23
3.2 Técnicas de defensa	24
4 Otros tipos de ataques	27
Resumen	28
Ejercicios de autoevaluación	29
Solucionario	30
Glosario	31
Bibliografía	32

Introducción

Actualmente, con el rápido aumento de la capacidad de computación y la disminución de su coste en los últimos años, podemos encontrar implementaciones de algoritmos de aprendizaje automático en ámbitos muy diversos y en todo tipo de dispositivos, como nuestros teléfonos móviles. La adopción de estas tecnologías a gran escala abre un gran abanico de posibilidades en cuanto a la generación de modelos predictivos o la visión artificial, entre otras cosas, y aun así también comporta retos en cuanto a seguridad, puesto que estos sistemas son susceptibles de ser atacados con métodos no convencionales, de forma que se abre una puerta a fallos o a la extracción de información confidencial, entre otros.

Esto es relevante especialmente en un contexto de seguridad, donde tenemos un agente que intenta protegerse de la intrusión de agentes maliciosos. Si la detección de ataques se hace con un sistema de aprendizaje automático, los intrusos pueden usar técnicas diseñadas para inducir el mal funcionamiento de estos sistemas de detección.

En este módulo didáctico trataremos de manera introductoria los ataques y las técnicas de defensa respectivas en el contexto del aprendizaje automático. Primero definiremos el concepto de *ataque* y clasificaremos los tipos de ataque existentes según los criterios de utilidad y metodológicos. Posteriormente, nos centraremos en los ataques de envenenamiento (*poisoning*) y en los ataques en tiempo de decisión, presentando modelos de ataque genéricos, técnicas específicas de defensa y ejemplos relevantes. Finalmente, comentaremos otros tipos de ataque que se pueden llevar a cabo.

Objetivos

Los objetivos que se pretenden lograr con este módulo didáctico son los siguientes:

- 1.** Conocer los tipos de ataques que se pueden llevar a cabo contra un sistema de aprendizaje automático.
- 2.** Aprender a modelizar matemáticamente el funcionamiento de ciertos ataques.
- 3.** Conocer diferentes técnicas de protección contra los ataques más comunes.
- 4.** Saber evaluar el efecto de ataques y de métodos de predicción en el rendimiento de los algoritmos.

1. Ataques al aprendizaje automático

En este apartado presentaremos las características fundamentales de los ataques a los sistemas de aprendizaje automático y los clasificaremos basándonos en sus características más relevantes.

1.1 Descripción de los ataques al aprendizaje automático

Para hablar de un ataque a un sistema de aprendizaje automático, necesitamos cuatro ingredientes fundamentales:

- 1) **Conjunto de entrenamiento.** Se trata de un conjunto de datos, expresados normalmente en forma vectorial, que son una muestra representativa de la información que procesará el modelo.
- 2) **Modelo.** Se trata de un modelo algorítmico creado mediante algún proceso de aprendizaje automático con el conjunto de entrenamiento.
- 3) **Propietario del modelo.** Tiene información completa sobre los datos de entrenamiento y el mismo modelo, y asumimos que tiene objetivos lícitos. Hablamos de compañías que diseñan antivirus, administradores de redes corporativas, usuarios privados de ordenadores, etc.
- 4) **Atacante.** Es un agente que tiene como objetivo inducir un mal funcionamiento del modelo en algún sentido para conseguir un objetivo malicioso. Estos objetivos pueden ser, por ejemplo, introducir software malicioso en un ordenador, enviar correos de *phishing* o evadir la detección de acciones criminales por cámaras de seguridad.

Así, pues, definimos el ataque a un modelo de aprendizaje automático como la manipulación por parte de un atacante, o bien del conjunto de entrenamiento o bien de los datos que el modelo recibe durante su funcionamiento, para conseguir un objetivo ilícito.

Podemos ver muy bien representados los ataques a los modelos de aprendizaje automático en el contexto de la seguridad, puesto que por definición estudia un problema adversarial en el cual hay un agente que ha creado un modelo y que se quiere proteger de un agente *malicioso*.

En cuanto a la ciberseguridad, podríamos considerar el escenario de una empresa que tiene un sistema de detección de amenazas desplegado, como por ejemplo un IDS o antivirus, basado en inteligencia artificial para proteger su red corporativa de ataques de *hackers* denegando un servicio o software de secuestro. Los atacantes, por lo tanto, estarán interesados en manipular su software malicioso de tal manera que pueda evadir los sistemas de detección, mientras que la empresa tendrá que diseñar estrategias de defensa contra estas técnicas de manipulación.

En una escala más cotidiana, podemos considerar el problema de un gestor de correo electrónico que tenga incorporado un sistema de detección de correo basura basado en aprendizaje automático.

En el ámbito de la seguridad física podemos considerar, por ejemplo, el caso siguiente:

Una empresa de seguridad tiene cámaras instaladas que vigilan la caja fuerte de un banco y quiere construir un modelo de aprendizaje automático que sea capaz de detectar personas en el video grabado por las cámaras. Durante el entrenamiento del modelo mediante un conjunto de datos de personas, un grupo criminal obtiene acceso al conjunto de entrenamiento y manipula ciertas entradas para hacer que el modelo obtenga un mal rendimiento, sin que la empresa de seguridad lo detecte. De este modo, el grupo podrá intentar llevar a cabo un robo sin ser detectado por las cámaras.

También podemos encontrar ejemplos de ataques a sistemas de aprendizaje automático fuera del ámbito de la seguridad:

- Un ladrón podría utilizar una técnica de clonación de voz para suplantar la identidad de alguien y solicitar al banco una transferencia de su dinero a la cuenta del atacante.
- Hay sistemas de salud, como por ejemplo marcapasos, que utilizan técnicas de aprendizaje automático. Un atacante podría introducir artefactos en el conjunto de datos de entrenamiento de uno de estos sistemas para hacer que funcionara mal posteriormente y crear problemas de salud en las personas que los usaran.

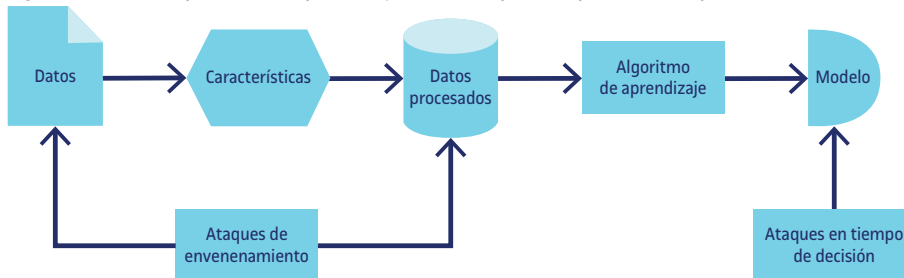
1.2 Categorización de los ataques al aprendizaje automático

Podemos clasificar las características de los ataques a los sistemas de aprendizaje automático basándonos principalmente en tres factores independientes: el momento del ataque, la intención del atacante y el conocimiento que tiene sobre el algoritmo.

1.2.1 El momento del ataque

Un algoritmo de aprendizaje automático requiere un proceso de entrenamiento antes de poder ser utilizado para hacer inferencia. Los ataques que se hacen antes o durante el entrenamiento se denominan *de envenenamiento*, mientras que los que se hacen durante la inferencia se denominan *ataques en tiempo de decisión*.

Figura 1. Fases del proceso de aprendizaje donde se pueden producir ataques.



Los **ataques de envenenamiento** se centran en manipular los datos usados durante el entrenamiento para interferir en él.

Esta manipulación provoca que el algoritmo tenga un funcionamiento diferente del esperado durante la inferencia, idealmente favorable a los intereses del atacante.

Ejemplo de un ataque de envenenamiento

Supongamos que queremos detectar correos basura. Para hacerlo, entrenamos a un clasificador con un conjunto de datos que contiene correos maliciosos y correos legítimos, cada uno etiquetado como tal. Un posible atacante que quisiera hacernos llegar correos de este tipo sin ser detectados podría cambiar las etiquetas de ciertos correos que estaban etiquetados previamente como legítimos. De este modo, el modelo integraría ciertos patrones maliciosos como patrones legítimos, lo que limitaría su capacidad de detección.

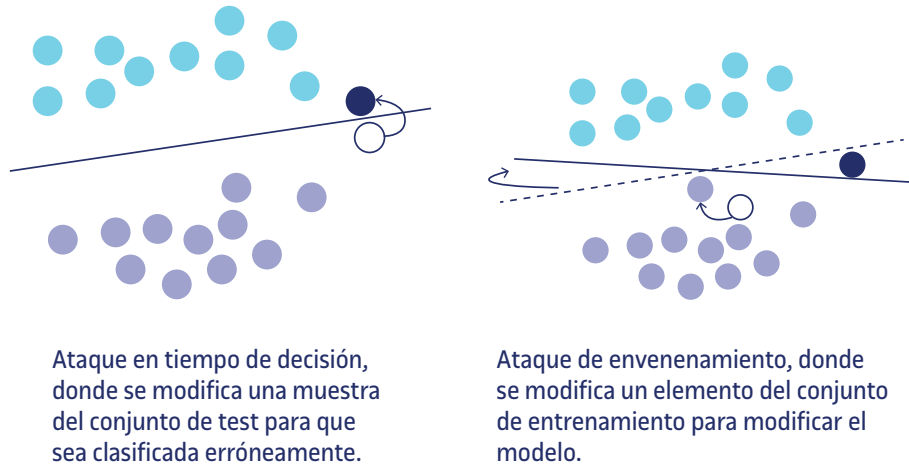
Los **ataques en tiempo de decisión** parten del supuesto de que el modelo ya está entrenado y se centran en modificar las entradas de entrada al modelo de forma que las predicciones se vean afectadas.

El caso más frecuente es la evasión de clasificadores, en la que, dada una entrada x con clase y , se intenta crear una modificación $\bar{x}$ lo más pequeña posible de forma que el clasificador f prediga una clase $f(\bar{x}) \neq y$.

Ejemplo de un ataque en tiempo de decisión

Volviendo al ejemplo del detector de correo basura, un ataque en tiempo de decisión podría consistir en intentar hacer pasar el correo basura como legítimo modificando su contenido ligeramente, como por ejemplo sustituyendo ciertas palabras por sinónimos menos sospechosos o introduciendo ciertas palabras clave relacionadas con el receptor de los correos.

Figura 2. Comparación del momento del ataque.



1.2.2 Objetivos del atacante

Los motivos del atacante pueden ser diversos, pero los podemos dividir en tres categorías: ataques dirigidos, ataques no dirigidos y ataques de extracción de información.

Un **ataque dirigido** tiene como objetivo disminuir o aumentar la clasificación de un tipo de ejemplos concreto.

Un **ataque no dirigido** tiene como objetivo degradar el rendimiento de un modelo sin tener en cuenta cómo se efectúa esta degradación.

Un **ataque de extracción de información** tiene como objetivo obtener información confidencial, ya sea sobre los datos de entrenamiento o sobre el modelo en sí mismo.

1.2.3 Información del atacante

Uno de los factores más importantes que hay que considerar es la información que el atacante tiene sobre el sistema que ataca. Diferenciamos dos categorías: los ataques de caja blanca, en los cuales el atacante tiene información perfecta, y los ataques de caja negra, en los cuales se dispone solo de información parcial.

En el contexto que nos ocupa, tener información perfecta sobre el sistema implica conocer los parámetros del modelo y los datos de entrenamiento. En este módulo consideraremos principalmente este tipo de ataques, puesto que, asumiendo que el atacante tiene información perfecta, consideramos el escenario más negativo desde el punto de vista del defensor y, diseñando estrategias teniendo en cuenta, nos preparamos para todos los supuestos. Ahora bien, si el atacante tuviera una falta de información, podría optar por hacer cierto tipo de extracción de información (comentaremos algún ejemplo de ello al final del módulo) para llegar a la información perfecta.

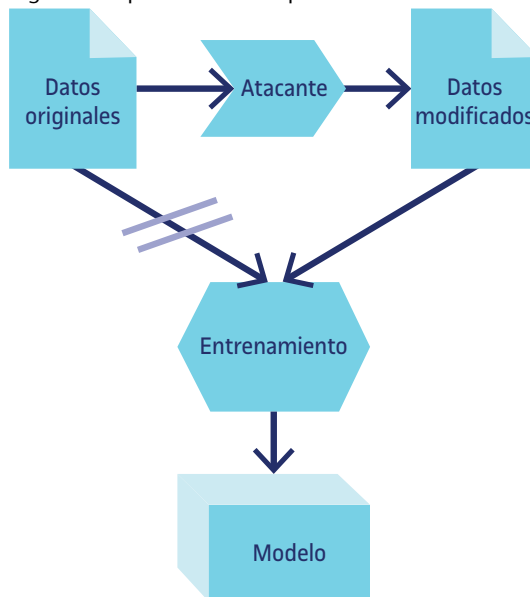
2. Envenenamiento

En este apartado trataremos los ataques de envenenamiento, que consisten en modificar los datos del conjunto de entrenamiento antes o durante el proceso de entrenamiento de los modelos de aprendizaje para degradar su funcionamiento. Primero analizaremos la anatomía del ataque y posteriormente trataremos las técnicas de defensa apropiadas.

2.1 Modelización de los ataques de envenenamiento

Supongamos que tenemos un conjunto de datos de entrenamiento $\mathcal{D}_0$. La metodología general del atacante A será transformarlo a otro conjunto de datos $\mathcal{D}$ antes de que suceda el entrenamiento y sin ser descubierto. Por lo tanto, se presenta una necesidad de compromiso entre maximizar el objetivo malicioso, medido basándose en las predicciones posteriores del modelo sobre un conjunto de instancias S , y minimizar la alteración de los datos.

Figura 3. Esquema de un ataque de envenenamiento.



Definimos, pues, $\mathcal{U}_A(\mathcal{D}, S)$ la función de utilidad para el atacante, que depende tanto de los nuevos datos de entrenamiento $\mathcal{D}$ como del conjunto de instancias en las que se evalúa el resultado del ataque S , además de los parámetros ω aprendidos por el modelo, que dejamos implícitos. Esta cantidad medirá la consecución del objetivo malicioso, sea cuál sea, y por lo tanto el atacante buscará maximizarla.

Ejemplos de funciones de utilidad

Algunos ejemplos de posibles funciones de utilidad que puede tener en cuenta el atacante son:

- El error total del modelo en el conjunto de entrenamiento $\mathcal{D}_0$ (ataque no dirigido).
- Si el modelo es un clasificador, el error total del modelo en el subconjunto de datos, que son de una clase concreta y_0 (ataque dirigido).

Por otro lado, sea $c(\mathcal{D}, \mathcal{D}_0)$ la función de coste del ataque, que cuantifica la desviación del conjunto de datos modificado respecto del conjunto original. Como atacantes, queremos minimizar esta cantidad, puesto que, cuanto más grandes sean las modificaciones que introducimos, más probable será que sean detectadas por el defensor.

Ejemplos de funciones de coste

Algunos ejemplos de posibles funciones de coste del ataque son:

- Para datos continuos, la distancia euclídea entre los datos del conjunto original $\mathcal{D}_0$ y las modificaciones respectivas a $\mathcal{D}$.
- Para datos discretos, para cada dato modificado calculamos el número de componentes que han cambiado y calculamos la media.

Así, pues, el problema se puede representar matemáticamente como un problema de maximización de una función objetivo (la utilidad para el atacante) teniendo en cuenta siempre el coste c . A continuación presentamos dos posibles formulaciones para el problema.

En la primera formulación incorporamos la función de coste como una penalización a la función que hay que maximizar, mientras que en la segunda imponemos simplemente una cota superior para el coste. Las dos son equivalentes:

$$\max_{\mathcal{D}} \mathcal{U}_A(\mathcal{D}, \mathcal{S}) - \lambda c(\mathcal{D}, \mathcal{D}_0) \quad (2.1)$$

o bien

$$\begin{aligned} \max_{\mathcal{D}} \mathcal{U}_A(\mathcal{D}, \mathcal{S}) \\ \text{sujeta a: } c(\mathcal{D}, \mathcal{D}_0) \leq C \end{aligned} \quad (2.2)$$

siendo C una cota superior para la desviación permitida, $C, \lambda > 0$. Utilizaremos una u otra formulación según lo que nos convenga: la primera tiene la ventaja de no constar de ninguna restricción, pero la función que hay que maximizar es más compleja. En cambio, en la segunda formulación se tiene que maximizar una función más simple, pero solo podremos usar métodos de optimización con restricciones.

Vectores de características discretos y continuos

Decimos que un vector de características es discreto cuando sus componentes solo pueden tomar valores discretos (un número finito o infinito pero numerable). En caso contrario, decimos que es continuo (toma valores reales o complejos).

$\max_{\mathcal{D}}$

Con $\max_{\mathcal{D}} f$ nos referimos a encontrar el máximo de $f(x)$ para todo elemento x del conjunto $\mathcal{D}$.

2.2 Ataques al aprendizaje supervisado

Empezaremos estudiando los ataques a modelos de aprendizaje supervisado. Si separamos este caso del aprendizaje no supervisado es porque aquí trabajamos con etiquetas, y, por lo tanto, las técnicas de ataque se basan generalmente en manipularlas de alguna manera. En particular, nos restringiremos a estudiar el caso de los clasificadores binarios, pero todo se extiende de manera natural a clasificadores generales.

2.2.1 Modificación de etiquetas

La **modificación de etiquetas** consiste en cambiar la etiqueta o clase de ciertos datos del conjunto de entrenamiento sin modificar los datos por sí mismos.

Actualmente, con el manejo de grandes volúmenes de datos por parte de los algoritmos de aprendizaje, a menudo extraídos en masa, la dificultad primordial para el aprendizaje supervisado es disponer de etiquetas fiables para estos datos. Es una práctica común recurrir a sistemas auxiliares como el abastecimiento participativo (*crowdsourcing*) para conseguir esta información, un hecho que abre la puerta a introducciones de información maliciosas. También observamos este problema en entornos de aprendizaje distribuido o federado, donde los datos provienen simultáneamente de muchos orígenes, no todos verificados.

Consideraremos el problema para el caso de un clasificador lineal f . Denotaremos $f(x; \mathcal{D})$ en la predicción de f sobre el ejemplo x cuando f ha sido entrenado sobre el conjunto de datos $\mathcal{D}$.

Sea $\mathcal{D}_0 = \{(x_i, y_i)\}_{i=1 \dots n}$ con $y_i \in \{0, 1\}$, y suponemos una cota superior C para el coste de edición de las etiquetas y_i . Sea $z = \{z_i\}_{i=1}^n$ tal que $z_i = 1$ si decidimos modificar y_i , $z_i = 0$, y el c_i coste de invertir la etiqueta. Supongamos también, para simplificar el problema, que S es el conjunto total de entrenamiento.

El objetivo del atacante es introducir cambios en el conjunto de datos de forma que el clasificador sea entrenado en este conjunto modificado y obtenga un alto rendimiento (para no levantar sospechas) y obtenga a la vez el peor rendimiento posible en el conjunto no modificado. De este modo se sabotea el rendimiento del modelo en los datos que encontrará en tiempo de predicción. Podemos expresar la función de utilidad de la manera siguiente:

$$\mathcal{U}_A(\mathcal{D}) = - \sum_{(x_i, y_i) \in \mathcal{D}} l(y_i, f(x_i; \mathcal{D}_0)) - l(y_i, f(x_i; \mathcal{D}(z)))$$

Aprendizaje supervisado

El objetivo del aprendizaje supervisado es crear una función capaz de predecir el valor correspondiente a cualquier objeto de entrada válido después de haber visto una serie de ejemplos, los datos de entrenamiento. Los datos de entrenamiento consisten en pares de objetos, a menudo vectores, en los cuales un componente son los datos de entrada y el otro corresponde a los resultados deseados o etiquetas.

Abastecimiento participativo

Consiste en obtener datos para entrenar un modelo de manera distribuida y colaborativa haciendo que un gran número de voluntarios aporte datos, generalmente de manera anónima.

para cierta función de pérdida l y siendo $\mathcal{D}(z)$ el conjunto de datos original modificado según z .

Ejemplos de funciones de pérdida

- **Error cuadrático medio.** Sea $\bar{x}$ una aproximación de x . El error cuadrático medio se expresa:

$$\sum_{i=1}^n \|x_i - \bar{x}_i\|^2. \quad (2.3)$$

Es muy utilizado debido a su conveniencia desde el punto de vista matemático, puesto que es una función convexa y representa la varianza en el caso de un estimador no sesgado, pero tiene la desventaja de acentuar el peso de los valores extremos (*outliers*) debido a los términos cuadráticos.

- **Entropía cruzada categórica.** Sea p_i la probabilidad real de la clase i y $\bar{p}_i$ la probabilidad de que el modelo prediga para la clase i . La función de entropía cruzada se expresa:

$$\sum_{i=1}^n p_i \log \left(\frac{1}{\bar{p}_i} \right). \quad (2.4)$$

Mide la diferencia entre la distribución de probabilidad real y la predicha. Minimizar la entropía cruzada es equivalente a maximizar la log-verosimilitud.

Reformulando la ecuación (2.2) con las hipótesis anteriores, obtenemos:

$$\begin{aligned} \max_z \quad & \sum_{(x_i, y_i) \in \mathcal{D}_I} l(y_i, f(x_i; \mathcal{D}_0)) - l(y_i, f(x_i; \mathcal{D}(z))) \\ \text{sujeta a:} \quad & \end{aligned} \quad (2.5)$$

$$\sum_{i=1}^n z_i c_i \leq C \quad z_i \in \{0, 1\}$$

Para eliminar la dependencia de la elección de las etiquetas z del conjunto de datos $\mathcal{D}(z)$, podemos aplicar un truco que consiste en tomar un conjunto de datos $\mathcal{D}$ de $2n$ elementos donde los primeros n son una copia del conjunto original $\mathcal{D}_0$ y donde la otra mitad es $\mathcal{D}_0$ pero con todas las etiquetas invertidas. Haciendo esto, el problema se transforma en uno equivalente pero más tratable matemáticamente:

$$\begin{aligned} \max_q \quad & \sum_{i=0}^{2n} q_i [l(y_i, f(x_i; \mathcal{D}_0)) - l(y_i, f(x_i; \mathcal{D}))] \\ \text{tal que:} \quad & \end{aligned} \quad (2.6)$$

$$q_i + q_{i+n} = 1 \quad i = 1 \dots n$$

$$\sum_{i=1}^{2n} q_i c_i \leq C \quad q_i \in \{0, 1\}$$

En el problema 4 del *notebook* podréis encontrar una posible implementación del algoritmo anterior para *Support Vector Machines* (SVM).

2.2.2 Ataques de inserción

En un **ataque de inserción** el atacante busca introducir un cierto número de datos adicionales al conjunto de datos de entrenamiento para distorsionar el entrenamiento lo máximo posible.

Consideremos el caso primario en que el atacante introduce un solo dato (x_a, y_a) e intenta inducir un error máximo sobre un punto objetivo del conjunto de entrenamiento (x_T, y_T) . Matemáticamente, el adversario intentará maximizar la función de utilidad:

$$\max_{x_a} \mathcal{U}_A(x_a) = l(y_T, f_{x_a}(x_T)). \quad (2.7)$$

No se impone ninguna restricción adicional respecto al coste de inserción, puesto que el hecho de limitar la inserción a un solo elemento ya comporta una restricción.

Ejemplo de un ataque de inserción

En *Poison frogs! Targeted clean-label poisoning attacks on neural networks* (NeurIPS 2018), los autores aplican ataques de inserción basados en optimización para engañar clasificadores basados en reconocimiento de imagen sobre el conjunto de datos CIFAR. Consiguen, por ejemplo, que imágenes de perros sean clasificadas como pájaros.

2.3 Ataques al aprendizaje no supervisado

Consideremos ahora un escenario de aprendizaje no supervisado, es decir, que nuestro conjunto de datos original está formado por vectores no etiquetados $\{x_i\}_{i=1}^n$, que expresaremos también como $\mathbf{X}$, la matriz asociada al conjunto de datos. Trataremos el caso prototípico de un algoritmo de clusterización, en el que queremos aprender una función $f(\mathcal{D})$ tal que asigna cada punto del conjunto de datos a un clúster. De forma más general, podemos representar esta función como una matriz $\mathbf{Y}$ tal que y_{ij} es la probabilidad de que x_i pertenezca al clúster j . Ahora bien, como la numeración de los clústeres es irrelevante, lo que realmente nos importa es la probabilidad de que dos vectores cualesquiera pertenezcan al mismo clúster. Para cuantificar esto, definimos la matriz (simétrica) $\mathbf{Z} = \mathbf{Y}\mathbf{Y}^T$, de forma que z_{ij} es esta probabilidad para los vectores x_i y x_j .

En este punto diferenciaremos entre ataques dirigidos y ataques no dirigidos. En un **ataque dirigido** un atacante definiría una asignación de clústeres ob-

jetivo $\mathbf{Z}^*$ e intentaría modificar los datos de forma que $\mathbf{Z}$ se aproximara lo máximo posible al objetivo. La función de utilidad sería, pues:

$$\mathcal{U}_A = -\|\mathbf{Z} - \mathbf{Z}^*\|_F, \quad (2.8)$$

donde $\|\cdot\|_F = \sqrt{\sum_i \sum_j |a_{ij}|^2}$.

En cambio, en un **ataque no dirigido** el atacante buscaría simplemente obtener una matriz $\mathbf{Z}$ lo más alejada posible de la matriz $\mathbf{Z}_0$, que resultaría de entrenar con los datos no adulterados. La función de utilidad toma la forma:

$$\mathcal{U}_A = \|\mathbf{Z} - \mathbf{Z}_0\|_F \quad (2.9)$$

En ambos casos la función de coste es

$$c(\mathcal{D}, \mathcal{D}_0) = c(\mathbf{X}, \mathbf{X}_0) = \|\mathbf{X} - \mathbf{X}_0\|_F. \quad (2.10)$$

Usando las ecuaciones 2.1 y 2.2, obtenemos las dos formulaciones siguientes equivalentes para el problema de optimización:

$$\max_{\mathbf{X}} \mathcal{U}_A(\mathbf{Z}_0, \mathbf{Z}(\mathbf{X})) - \lambda c(\mathbf{X}, \mathbf{X}_0), \quad (2.11)$$

o bien

$$\begin{aligned} \max_{\mathbf{X}} \mathcal{U}_A(\mathbf{Z}_0, \mathbf{Z}(\mathbf{X})) \\ \text{tal que :} \\ c(\mathbf{X}, \mathbf{X}_0) \leq C. \end{aligned} \quad (2.12)$$

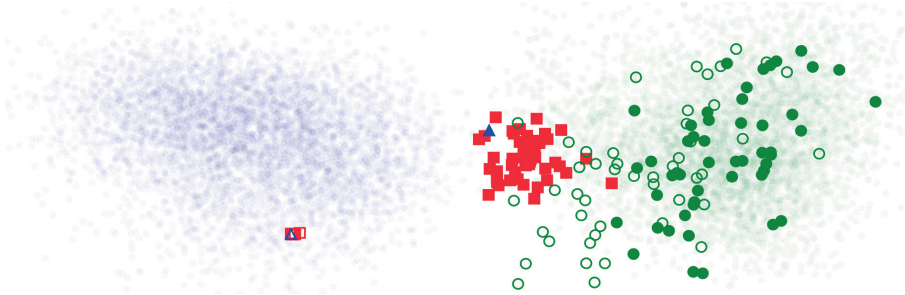
Ejemplo de ataque a un modelo de clusterización

Supongamos que un modelo separa el conjunto de datos $\mathcal{D}$ en k clústeres. Podemos usar este método heurístico para evitar el problema de optimización, que puede ser costoso computacionalmente: para cada uno, encontramos su centroide y calculamos las distancias respecto de los otros $k-1$ centroides. Seguidamente, insertamos $k-1$ datos sintéticos situados en el punto medio de estas distancias. De este modo, maximizamos, para cada par de clústeres, la probabilidad de que se fusionen. Haremos un ataque no dirigido, puesto que distorsionamos lo máximo posible la asignación de clústeres del modelo.

2.4 Técnicas de defensa

El diseño de técnicas de defensa contra ataques de envenenamiento se basa esencialmente en construir algoritmos de aprendizaje robustos, es decir, que sean capaces de aprender un modelo suficientemente similar al modelo óptimo a pesar de que los datos de entrenamiento hayan sido alterados.

Figura 4. Ejemplo de inserción de datos maliciosos (en rojo) para juntar dos clústeres (en verde y azul).



En general, distinguimos tres técnicas principales en cuanto al aprendizaje robusto, que introduciremos a continuación: el submuestreo, la eliminación de valores extremos y la optimización recortada.

2.4.1 Submuestreo

La idea general de la técnica del submuestreo se basa en extraer muchos subconjuntos de medida m del conjunto de datos de entrenamiento $\mathcal{D}$ y entrenar un modelo diferente para cada uno. Posteriormente, se selecciona el modelo que ha obtenido el mejor rendimiento según la métrica definida por el problema, como la log-verosimilitud, el error cuadrático medio, etc.

Este enfoque es especialmente efectivo si el porcentaje del conjunto de datos que ha sido modificado es pequeño, porque entonces, si m es pequeño comparado con la medida de $\mathcal{D}$, habrá una alta probabilidad de que alguno de los subconjuntos generados no contenga elementos maliciosos.

Algoritmo 2.1 Sub-muestreo

```

1:  $\mathcal{D} = \mathcal{D}_1 \sqcup \dots \sqcup \mathcal{D}_s$ 
2: for  $i = 1, \dots, s$  do
3:    $f_i \leftarrow$  modelo entrenado en  $\mathcal{D}_i$ 
4:    $e_i \leftarrow l(f_i, \mathcal{D}_i)$ 
5:  $f^* \leftarrow f_i$  tal que  $e_i$  es mínimo
6: devolver  $f^*$ 

```

Unión disjunta

Con $\mathcal{D}_1 \sqcup \dots \sqcup \mathcal{D}_s$ nos referimos a la unión disjunta de los subconjuntos, puesto que hemos hecho una partición de $\mathcal{D}$, y por lo tanto los conjuntos no tienen intersección.

Medida del conjunto de entrenamiento

El método de submuestreo es recomendable que lo usemos solo cuando dispongamos de un conjunto amplio de datos, puesto que si no es así los subconjuntos serían demasiado pequeños y habría riesgo de sobreajuste, puesto que cada uno de estos subconjuntos no contendría una muestra suficientemente representativa.

2.4.2 Eliminación de valores extremos

La técnica de eliminación de valores extremos pretende identificar y eliminar elementos sospechosos de haber sido introducidos de manera artificial en nuestro conjunto de datos para intentar obtener de nuevo un conjunto inalterado.

Existen varias estrategias para eliminar valores extremos. Una de estas estrategias se basa en el análisis de componentes principales (PCA).

Tomamos la matriz de vectores del conjunto de datos (posiblemente modificado) $\mathcal{D}$, encontramos iterativamente el componente de más varianza (a lo largo del vector propio de mayor valor propio) y eliminamos los vectores que tienen mayores valores una vez proyectados sobre esta dimensión. Repetimos este proceso hasta que no podamos eliminar más elementos. Para clasificadores, se puede probar que es posible obtener una cota para el error máximo que depende solo del porcentaje de alteración y de la medida del conjunto de datos.

Algoritmo 2.2 Eliminación de valores propios con PCA

```

 $\mathcal{D}^* \leftarrow \mathcal{D}$ 
repetir
   $\mathbf{A} \leftarrow \sum_{x \in \mathcal{D}^*} x x^T$ 
   $v \leftarrow$  vector propio de mayor valor propio de  $\mathbf{A}$ 
   $S := \{x \in \mathcal{D}^* \text{ t.q. } (v^T x)^2 \geq \frac{10 \log n}{m}\}$ 
   $\mathcal{D}^* \leftarrow \mathcal{D}^* - S$ 
hasta que  $S = \emptyset$ 
devolver  $\mathcal{D}^*$ 
  
```

Vectores y valores propios

Dada una matriz A , diremos que el vector v es un vector propio de A de valor propio λ si $Av = \lambda v$.

Otra idea, aplicable a clasificadores, se basa en generar los llamados *micromodelos*:

Particionamos $\mathcal{D}$ en n subconjuntos $\mathcal{D}_i$ y entrenamos un clasificador f_i en cada uno de estos conjuntos. Posteriormente, utilizamos estos micromodelos para limpiar el conjunto de datos de la manera siguiente:

Dado $x \in \mathcal{D}$, lo eliminaremos de $\mathcal{D}$ si se cumple que

$$\sum_i \omega_i f_i(x) \geq L$$

Para un umbral L prefijado y unos ciertos pesos ω_i preestablecidos, podemos tomar $1/n$ si consideramos simplemente la media sin valorar ningún modelo en especial. La idea de este método es introducir robustez en la hora de decidir qué datos son anómalos: si hemos entrenado muchos modelos y la suma ponderada de la predicción de todos ellos sobre un dato en concreto es mayor que un cierto umbral fijado, tendremos mucha más certeza para clasificarlo como anómalo que si lo hacemos con un solo modelo. Los valores de los pesos ω_i nos pueden permitir ajustar la predicción valorando más la opinión de ciertos micromodelos que hayamos determinado que producen predicciones más fiables.

2.4.3 Optimización recortada

Supongamos que el conjunto de datos de entrenamiento tiene n elementos y un atacante introduce m adicionales. Nuestro objetivo para protegernos del

ataque es aprender un modelo sobre este conjunto de datos modificado que sea lo más parecido posible al modelo entrenado con datos no corrompidos. La esencia de la técnica de la optimización recortada consiste en asumir que el atacante tiene información completa sobre nuestro modelo y en observar que su objetivo es introducir datos que maximicen nuestra función de pérdida l . Por lo tanto, esperaremos que, de todos los datos $x \in \mathcal{D}$, los que aporten una pérdida mayor serán las maliciosas.

Partiendo de aquí, es natural definir el problema de optimización para el defensor así:

$$\min_{\omega} \sum_{j=1}^n \{l(y_i, f_{\omega}(x_i)), i = 1 \dots n+m\}_{(j)}, \quad (2.13)$$

donde $\{\}_{(j)}$ simboliza el i -ésimo elemento más pequeño de la sucesión. Es decir, seleccionamos solo los n valores mayores de la función de pérdida a la hora de hacer la optimización, puesto que hemos hecho la hipótesis de que estos valores son los que corresponden a los datos maliciosos.

Para que el problema sea tratable computacionalmente, es conveniente hacer una modificación:

$$\begin{aligned} \min_{\omega, \{\tau_i\}_{i=1, \dots, n+m}} \sum_{i=1}^{n+m} \tau_i l(y_i, f_{\omega}(x_i)) \\ \text{t.q. } 0 \leq \tau_i \leq 1 \quad i = 1 \dots n+m, \quad \sum_i \tau_i = n. \end{aligned} \quad (2.14)$$

A continuación aplicamos el algoritmo 2.3 para hacer un proceso de optimización dual, en el cual optimizamos alternativamente las τ_i y los parámetros ω del modelo.

Algoritmo 2.3 Optimización recortada

- 1: $\tau_i \leftarrow \{0,1\} i = 1 \dots n+m$ aleatoriamente tal que $\sum_i \tau_i = n$.
 - 2: $\omega \leftarrow \arg \min_{\omega} \sum_i \tau_i l(y_i, f_{\omega}(x_i))$.
 - 3: Ordenar ascendentemente $\{l(y_i, f_{\omega}(x_i))\}_i$ y obtener la ordenación $\{o_i\}_i$.
 - 4: $\tau_i \leftarrow 1$ si $o_i \leq n$, $\tau_i \leftarrow 0$ de lo contrario.
 - 5: Volver a 2 si se ha producido algún cambio en las τ_i ; de lo contrario continuar.
 - 6: Devolver ω .
-

3. Ataques en tiempo de decisión

En un ataque en tiempo de decisión el adversario pretende generar una decisión incorrecta de un modelo ya entrenado modificando los datos de entrada. En este apartado nos centraremos en entender los ataques en tiempo de decisión para proporcionar técnicas efectivas de defensa ante ellos. Presentaremos modelizaciones para clasificadores binarios y multiclase, puesto que veremos que con estos dos casos podemos generalizar fácilmente otros tipos de modelos, como los de clusterización.

3.1 Modelización de los ataques en tiempo de decisión

Partimos del hecho de que disponemos de los elementos siguientes:

- Un clasificador $f(x) = \text{sgn}(g(x))$ para alguna función $g(x)$ que toma valores reales. Supondremos que g toma valores positivos para los elementos maliciosos y negativos para los elementos benignos.
- Un vector adversarial x^A . Se trata de un dato sobre el cual el atacante quiere inducir una clasificación errónea.

Signo

Con $\text{sgn}(a)$ nos referimos al signo de a como número real.

Ejemplos de vectores adversariales

- Una imagen de una persona que el atacante quiere que sea clasificada como otra cosa para engañar a un detector de intrusos de un banco basado en el reconocimiento de imágenes.
- Una IP maliciosa que el atacante quiere que sea clasificada como benigna por un sistema de seguridad de una red corporativa.
- Una función de coste $c(x, x^A)$ que asigna un coste, desde el punto de vista del atacante, a modificar el vector adversarial x^A y convertirlo en x . Este coste corresponderá generalmente a una degradación de funcionalidad maliciosa.

La expresión de la función de coste c puede tener diferentes formas. Las más comunes son las basadas en alguna norma $\mathcal{L}_p$:

$$c(x, x^A) = \|x - x^A\|_p. \quad (3.1)$$

También podemos considerar la norma ponderada:

$$c(x, x^A) = \left(\sum_i \alpha_i |x_i - x_i^A|^p \right)^{1/p}, \quad (3.2)$$

que tiene sentido cuando las características comportan costes de modificación variables.

Aun así, las formulaciones anteriores son poco útiles cuando tratamos con modelos en los que ciertas características de ataque pueden ser sustituidas por otros incurriendo un riesgo cero. Un ejemplo de esto lo encontramos en el ámbito lingüístico, donde el atacante podría sustituir una palabra maliciosa por un sinónimo menos común para evadir un detector. En estos escenarios tiene sentido considerar una función de coste del tipo

$$c(x, x^A) = \left(\sum_i \alpha_i \min_{j \notin C_i} |x_i - x_i^A|^p \right)^{1/p}, \quad (3.3)$$

donde C_i es la clase de equivalencia de x_i^A que contiene las características que pueden ser sustituidas por x_i^A sin ningún coste. Es decir, hacemos la minimización solo sobre los elementos que aportan un coste de sustitución.

3.1.1 Ataques a clasificadores binarios

Una vez fijada la función de coste para el atacante, consideramos el problema de optimización para encontrar el valor óptimo de la modificación x :

$$\min_x \{ \min(g(x), 0) + \lambda c(x, x^A) \}, \quad (3.4)$$

donde λ es un parámetro que regula la importancia relativa del coste del ataque. En esta formulación el hecho de que x sea clasificado como malicioso ($g(x) > 0$) aporta un beneficio nulo, mientras que se bonifican valores de x que obtengan valores más bajos de $g(x)$. Existen dos relajaciones del problema para hacerlo convexo:

Por un lado, podemos sustituir el término $\min\{g(x), 0\}$ por $g(x)$, y obtenemos:

$$\min_x \{ g(x) + \lambda c(x, x^A) \}, \quad (3.5)$$

Por otro lado, podemos ignorar el valor concreto de g , solo valorando que sea negativo (o que $f = -1$), y obtenemos:

$$\min_x c(x, x^A) \quad (3.6)$$

$$\text{t.q.: } f(x) = -1.$$

Finalmente, también podemos optar para minimizar solo g , imponiendo una cota superior para el coste C :

Función convexa

Decimos que f es una función convexa si para todo par de puntos x_1, x_2 de su dominio se tiene que $f(\lambda x_1 + (1 - \lambda) x_2) \leq \lambda f(x_1) + (1 - \lambda) f(x_2)$ para todo $\lambda \in [0, 1]$. Un ejemplo típico es la parábola $f(x) = x^2$.

$$\begin{aligned} \min_x g(x) \\ \text{t.q.: } c(x, x^A) \leq C. \end{aligned} \quad (3.7)$$

La mayoría de formulaciones que hemos propuesto son convexas, y por tanto pueden ser resueltas aplicando técnicas estándares de optimización convexa. Sin embargo, en general, no lo son necesariamente. En estos casos se puede aplicar un descenso de gradiente para encontrar óptimos locales. Ilustraremos el procedimiento para su formulación (3.5):

Algoritmo 3.1 Descenso de gradiente para ataques en tiempo de decisión

```

1:  $x^* \leftarrow x^A$ 
2: for  $t = 1 \dots N_{iter}$  do
3:    $G(x) \leftarrow \nabla_x g(x) + \lambda \nabla_x c(x, x^A)$ 
4:    $x^* \leftarrow x^* - \beta_t G(x)$ 
return  $x^*$ 
  
```

donde β_t es la tasa de aprendizaje, que podemos ir variando según un ciclo establecido (es común hacerla decrecer como una exponencial negativa).

Cuando las variables son discretas, las estrategias que se suelen seguir son las llamadas *coordinate greedy*, que consisten en ir fijando coordenadas y modificar el valor entre un número finito de opciones de manera iterativa hasta que su valor no pueda mejorar.

3.1.2 Ataques a clasificadores multiclase

Si en los clasificadores binarios teníamos una función de decisión del tipo $f(x) = \text{sgn}(g(x))$, en un clasificador multiclase el modelo es:

$$f(x) = \arg \max_i (x g_i(x)) \quad i = 1 \dots s \quad (3.8)$$

para ciertas funciones g_i . Si $g_i \in [0, 1]$ para todo i y $\sum_i g_i = 1$, entonces podemos interpretar $g_i(x)$ como la probabilidad (predicha) de que x pertenezca a la clase i .

Ahora supongamos que el atacante quiere hacer un ataque dirigido, y por lo tanto quiere que una instancia x sea clasificada como perteneciente a la clase k . Modificando (3.5), obtenemos como problema de optimización:

$$\min_x \{-g_k(x) + \lambda c(x, x^A)\}. \quad (3.9)$$

Funciones no convexas

Las funciones convexas tienen la propiedad de tener un mínimo local, que es también mínimo global, y por tanto es fácil encontrar el mínimo global cuando son diferenciables; solo tenemos que imponer que la derivada se anule. En cambio, las funciones no convexas pueden tener muchos mínimos locales, y en general se usan métodos como el descenso de gradiente para encontrar mínimos locales buenos.

Métodos de optimización

Para funciones convexas dos veces diferenciables, es común usar el método de Newton-Raphson, que tiene convergencia cuadrática (más información). Para funciones no convexas, hemos comentado anteriormente el descenso de gradiente (más información).

Si, por el contrario, se quiere hacer un ataque no dirigido y x^A pertenece a la clase k , el objetivo es evitar que x sea clasificado como perteneciente a esta clase, y por lo tanto el problema es:

$$\min_x \{g_k(x) + \lambda c(x, x^A)\}. \quad (3.10)$$

Observamos, no obstante, que esta formulación obvia el valor de las otras funciones g_i : centrándonos en el caso de un ataque dirigido, mientras intentamos maximizar el valor de g_k , el valor de las otras g_i podría crecer y de este modo no sería posible obtener la clasificación buscada. Lo que busca el atacante, pues, es que

$$g_k(x) \geq g_i(x) + \gamma \quad \forall i \neq k, \quad (3.11)$$

donde se añade un margen de seguridad γ para más robustez.

Definiendo la función

$$h(x) = \max\{-\gamma, \max_{i \neq k} g_i(x) - g_k(x)\}, \quad (3.12)$$

se tiene que $h(x) = -\gamma$ si se cumple la condición (3.11) y $h(x) > -\gamma$ en los otros casos, y por lo tanto el problema de optimización que debe resolver el atacante es

$$\min_x \{h(x) + \lambda c(x, x^A)\}. \quad (3.13)$$

Observamos, además, que esta formulación se aplica también a modelos de clusterización, puesto que si tenemos s clústeres podemos definir f como (3.8), y $g_i(x)$ nos indica la probabilidad de que x pertenezca al clúster i . Como consideramos ataques en tiempo de decisión, el hecho de que el algoritmo no disponga de las clases en el momento del entrenamiento nos es irrelevante.

3.2 Técnicas de defensa

Supongamos que queremos entrenar un clasificador sobre un cierto conjunto de datos $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$. El objetivo es encontrar un modelo f tal que

$$\sum_{i=1}^n l(f(x_i), y_i) \leq \sum_{i=1}^n l(f'(x_i), y_i) \quad \forall f' \in \mathcal{F}, \quad (3.14)$$

por un cierto espacio de funciones $\mathcal{F}$ y una cierta función de pérdida l . Denominamos $\sum_{i=1}^n l(f(x_i), y_i)$ el riesgo empírico de f , que denotaremos para $\mathcal{R}(f)$. Es decir, queremos encontrar el modelo que minimice el riesgo empírico de entre todos los modelos posibles.

Consideremos ahora un escenario adversarial, en el que un atacante intenta modificar una parte de los datos en tiempo de decisión:

Sea $\mathcal{S} \subset \mathcal{D}$ el subconjunto adversarial, que está formado por las instancias sobre las cuales el atacante quiere inducir predicciones incorrectas. Formalizamos las decisiones de modificación del adversario como la función $\mathcal{A}(x; f) : \mathcal{S} \times \mathcal{F} \rightarrow \mathcal{S}'$. Teniendo en cuenta esta situación, podemos separar la función de riesgo esperado en un término que contiene instancias modificadas por el atacante y en otro que contiene solo instancias originales:

$$\mathcal{R}_A(f) = \sum_{(x,y) \in \mathcal{S}} l(f(\mathcal{A}(x; f)), y) + \sum_{(x,y) \notin \mathcal{S}} l(f(x), y). \quad (3.15)$$

Denominamos esta función *riesgo empírico adversarial*.

Para defendernos del atacante, queremos resolver el problema de optimización siguiente:

$$\min_{f \in \mathcal{F}} \mathcal{R}_A(f) + \rho(f), \quad (3.16)$$

en qué $\rho(f)$ es una función de regularización cualquiera.

El problema fundamental de esta formulación es que contiene simultáneamente la función que quiere aprender el atacante y la que quiere aprender el defensor. Por lo tanto, se trata de un juego que involucra a las dos partes.

Formalizándolo algo más desde el punto de vista de la teoría de juegos, se trata de un juego de Stackelberg en el que hay dos jugadores: un líder, que actúa primero (en este caso el defensor, que elige el modelo f) y un seguidor, que actúa en respuesta en la acción del líder (en este caso el atacante, que diseña la función $\mathcal{A}(\cdot; f)$ para confundir al modelo f).

La solución del juego será un equilibrio consistente en una combinación $\{f, \mathcal{A}(\cdot; f)\}$ tal que:

- $\mathcal{A}(x; f)$ es una respuesta óptima a f por cada $(x, y) \in \mathcal{S}$ (el atacante seguirá una estrategia de optimización como las presentadas en la sección anterior).
- f minimiza el riesgo empírico.

Aunque existen ciertas estrategias para encontrar soluciones exactas en el problema anterior, son costosas debido a su complejidad además de ser difícilmente escalables. Por lo tanto, se pueden aplicar técnicas encaradas o bien a

Función de regularización

Una función de regularización es una restricción que se impone sobre los parámetros de un modelo a la hora de entrenarlo. Generalmente, se trata de alguna norma, como la norma euclídea o la norma del supremo. El objetivo es conseguir que los parámetros del modelo no crezcan de manera descontrolada, puesto que esto podría causar inestabilidades numéricas.

Teoría de juegos

La teoría de juegos es una rama de la matemática aplicada que estudia las situaciones estratégicas en las que dos o más jugadores escogen diferentes acciones en un intento de maximizar sus ganancias o retornos. Generalmente, la solución de estos juegos pasa para encontrar el llamado *equilibrio*, en que ambas partes encuentran simultáneamente un estado óptimo en el que no pueden mejorar sus ganancias.

hacer el problema más sencillo, o bien a obtener robustez evitando resolver el problema analíticamente.

Para conseguir robustez, se puede aplicar el método de reentrenamiento iterativo. Consiste en entrenar al modelo sobre el conjunto de datos $\mathcal{D}$, transformar el conjunto de datos con el modelo entrenado, transformar el subconjunto adversarial $\mathcal{D}_{\mathcal{A}}$ haciendo uso de la función adversarial $\mathcal{A}$ y añadir estos datos transformados al conjunto original. Repetimos este proceso iterativamente hasta conseguir convergencia u otra condición de parada. Con este proceso conseguimos que el modelo tenga un conocimiento óptimo respecto al funcionamiento adversarial, dados los datos disponibles, y lo hacemos más resiliente a sus ataques.

Algoritmo 3.2 Reentrenamiento iterativo

```

1: Entrada: Datos de entrenamiento  $\mathcal{D}$ , función adversarial  $\mathcal{A}(x; f)$ 
2:  $\mathcal{V} = \mathcal{D}$ 
3: while  $\mathcal{V} \neq \emptyset$  do
4:    $\mathcal{V} \leftarrow \emptyset$ 
5:    $f \leftarrow \text{Train}(\mathcal{D})$ 
6:   for  $(x, y) \in \mathcal{D}_{\mathcal{A}}$  do
7:      $x' = \mathcal{A}(x; f)$ 
8:     if  $x' \notin \mathcal{D}$  then
9:        $\mathcal{V} \leftarrow \mathcal{V} \cup \{x'\}$ 
10:   $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{V}$ 
11: return  $f$ 
  
```

Por otro lado, se pueden aplicar métodos de relajación a la formulación del problema adversarial, como transformar espacios de características discretos en espacios de características continuos, o bien simplificar la función de riesgo suponiendo que todos los ejemplos son susceptibles de ser maliciosos.

4. Otros tipos de ataques

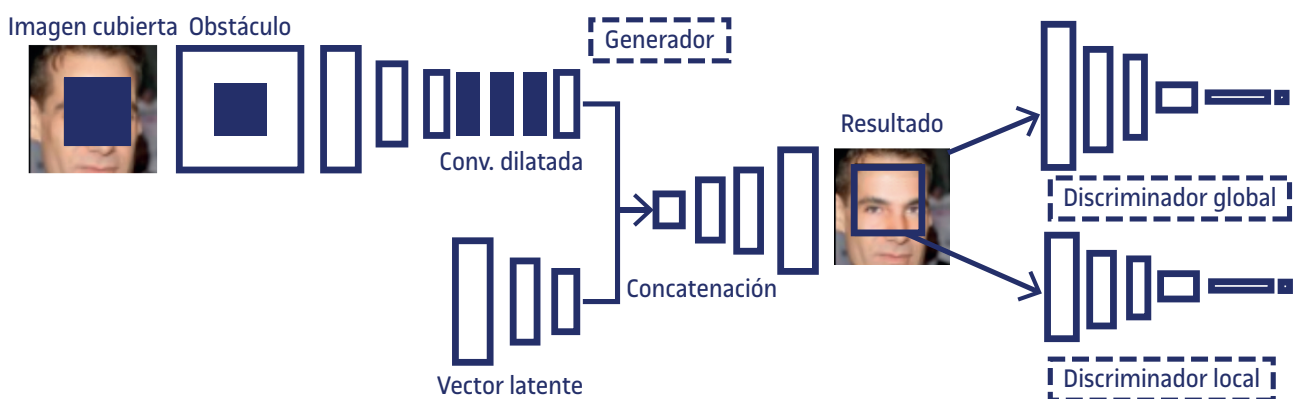
Para terminar, comentaremos dos tipos de ataques llevados a cabo recientemente que pueden ser interesantes y que buscan objetivos diferentes de los que hemos visto anteriormente.

- Actualmente se estudian los ataques denominados *de inversión de modelo*, en los cuales se abusa del acceso público a las predicciones de un modelo para inferir información sobre los datos que serán usados para su entrenamiento. Más concretamente, hacen uso de redes adversariales generativas (GAN) para generar datos sintéticos muy próximos a los reales, de los cuales solo disponen de información parcial.

Bibliografía recomendada

Yuheng Zhang y otros (2020). *The secret revealer: generative model-inversion attacks against deep neural networks*. arXiv.

Figura 5. Reconstrucción de una cara a partir de una imagen cubierta parcialmente.



- También se ha desarrollado una técnica llamada *extracción de modelo*, capaz de inferir los parámetros de un modelo conociendo solo su arquitectura. En particular, se aplica para encontrar los parámetros de clasificadores basados en aprendizaje profundo de la plataforma *Amazon Web Services*.

Aprendizaje profundo

El aprendizaje profundo (*deep learning*) es un término que engloba un conjunto de algoritmos de aprendizaje automático que intenta modelar abstracciones de alto nivel en los datos usando arquitecturas con transformaciones no lineales múltiples e iterativas, expresadas de manera matricial o tensorial.

Bibliografía recomendada

Florian Tramèr y otros (2016). *Stealing machine learning models via prediction APIs*. Proceedings of the 25th USENIX Security Symposium.

Resumen

En este módulo didáctico hemos tratado el problema de los ataques a los modelos de aprendizaje automático. En primer lugar, hemos hecho una definición del término y un estudio de los diferentes tipos, distinguiendo entre el momento del ataque, el objetivo del atacante y la información disponible sobre el modelo.

En segundo lugar, hemos desarrollado el estudio de los ataques de envenenamiento, que consisten en manipular los conjuntos de entrenamiento antes de que este se efectúe. Hemos estudiado las técnicas principales de ataque y los métodos de defensa más eficaces.

Después hemos presentado los ataques en tiempo de decisión, que se centran en manipular datos de entrada al modelo ya entrenado para inducir errores en las predicciones del modelo.

Finalmente, hemos hablado brevemente de las técnicas de inversión y de extracción de modelo, aportando referencias a artículos que las tratan en profundidad.

Ejercicios de autoevaluación

1. Explica la diferencia entre un ataque dirigido y uno no dirigido sobre un clasificador multiclase.

2. Considera los dos objetivos de ataque siguientes:

a) Introducir correo basura sin que acabe en la bandeja de correo no deseado.

b) Hacer que un sistema de conducción autónoma salga de la carretera.

Para cada uno de los casos, explica brevemente cómo generarías un ataque en tiempo de decisión y un ataque de envenenamiento para conseguir llevarlos a cabo.

3. Explica con tus palabras en qué consiste el reentrenamiento iterativo y razona por qué es un método efectivo de protección contra los ataques adversariales.

4. Decide si las afirmaciones siguientes son ciertas o falsas. Razona la respuesta en caso de que sean falsas.

a) Los ataques en tiempo de decisión consisten en modificar los datos de entrenamiento antes de que el algoritmo de aprendizaje automático haya sido entrenado para degradar el rendimiento del modelo que se obtendrá posteriormente.

b) El cimiento teórico de los métodos de defensa contra ataques en tiempo de decisión se basa en formalizar el problema como un juego de Stackelberg, en el cual tanto el atacante como el defensor son jugadores. Se llega a una solución óptima cuando una de las partes gana.

c) Los ataques en tiempo de decisión pretenden manipular los parámetros internos de los modelos a la hora de hacer las predicciones para provocar predicciones incorrectas.

d) La eliminación de valores extremos o la creación de micromodelos son técnicas efectivas de defensa contra los ataques de envenenamiento.

Solucionario

1. En un ataque dirigido se fija una clase i y se pretende que el modelo, o bien clasifique instancias de una clase j diferente de i como pertenecientes a la clase i o bien clasifique instancias de la clase i como pertenecientes a otra clase. Qué modalidad de las dos se efectúa depende del objetivo concreto del ataque.

Un ataque no dirigido, en cambio, tiene como objetivo degradar el rendimiento del clasificador de manera general, es decir, que el mayor número posible de instancias sean clasificadas incorrectamente independientemente de su clase.

2.

a) El modelo encargado de decidir si un correo es malicioso o no es un clasificador binario, y por tanto se podría lanzar un ataque de envenenamiento dirigido en que se generen una gran cantidad de correos maliciosos similares a los que se quieren introducir para confundir al modelo. Sería, por lo tanto, un ataque de inserción.

Por otro lado, podríamos hacer un ataque en tiempo de decisión modificando el correo basura que queremos introducir haciendo sustituciones de palabras por otras que obtengan una clasificación de riesgo inferior por parte del modelo.

b) El modelo encargado de controlar la conducción del vehículo autónomo es complejo, pero es probable que incorpore un sistema de visión artificial que detecte las líneas que delimitan la carretera.

Ejemplo de un ataque de envenenamiento: hacemos un ataque de modificación de etiquetas sobre el conjunto de entrenamiento asignando la etiqueta *carretera* a objetos que no lo son.

Ejemplo de un ataque en tiempo de decisión: este es más complicado, puesto que se tendrían que manipular físicamente elementos a lo largo de la carretera. De todos modos, en un entorno controlado de laboratorio se podrían hacer pruebas con líneas de la carretera borradas, mal pintadas, condiciones de luz adversas, etc. para ver qué confunde más al sistema de visión.

3. El reentrenamiento iterativo es una técnica de defensa contra los ataques en tiempo de decisión que consiste en entrenar muchas veces a un modelo, generar después de cada entrenamiento nuevos ejemplos adversariales e introducirlos en el conjunto de entrenamiento.

El método proporciona dos ventajas principales:

Por un lado, funciona como una forma de aumento de datos proporcionando al algoritmo de aprendizaje nuevas datos modificados ligeramente respecto de los originales. Esto puede ser especialmente relevante cuando se dispone de un conjunto de entrenamiento pequeño.

Por otro lado, los ejemplos introducidos son adversariales, es decir, han sido diseñados para ser difíciles de clasificar por el modelo. De este modo, a medida que las iteraciones del entrenamiento avanzan el problema de aprendizaje se hace más complejo. Esto prepara al modelo para posibles ataques futuros, que podrían ser similares a los introducidos, y le aporta robustez.

4.

a) Cierto.

b) Falso. La primera parte de la afirmación es cierta, pero la solución óptima se logra cuando se llega a un equilibrio en que ambas partes no pueden mejorar su ganancia.

c) Falso. Los ataques adversariales se efectúan en el momento de la predicción, pero no manipulan el modelo por sí mismo. En cambio, buscan manipular ligeramente los datos de entrada en el modelo para que generen predicciones incorrectas.

d) Cierto.

Glosario

abastecimiento participativo *m* Técnica de minería de datos que consiste en obtener datos para entrenar a un modelo de manera distribuida y colaborativa haciendo que un gran número de voluntarios aporten datos, generalmente de manera anónima.

aprendizaje profundo *m* Conjunto de algoritmos de aprendizaje automático que intenta modelar abstracciones de alto nivel en los datos usando arquitecturas con transformaciones no lineales múltiples e iterativas, expresadas de manera matricial o tensorial.

clusterización *f* Técnica que consiste en agrupar un conjunto de datos en una serie de grupos, generalmente disjuntos, obedeciendo a criterios semánticos, de localización espacial, etc.

descenso de gradiente *m* Algoritmo de optimización iterativo de primer orden para funciones diferenciables. Consiste en hacer una serie de pasos por la superficie de la función proporcionales y opuestos a la dirección del gradiente de la función en cada punto.

envenenamiento *m* Técnica de ataque a modelos de aprendizaje automático que consiste en manipular parcial o totalmente los datos de un conjunto de entrenamiento antes de que este se efectúe con el objetivo de degradar el rendimiento del modelo sobre los datos no alterados.

teoría de juegos *f* Rama de la matemática aplicada que estudia las situaciones estratégicas en las que dos o más jugadores escogen diferentes acciones en un intento de maximizar las ganancias o retornos.

Bibliografía

Fung, Clement; Beschastnikh, Ivan. *Privacy and security in cloud-based ML*. CPSC 416 guest lecture. University of British Columbia.

Goodfellow, Ian J.; Shlens, Jonathon; Szegedy, Christian (2015). *Explaining and harnessing adversarial examples*. ICLR 2015.

Shafahi, Ali; Ronny Huang, W.; Najibi, Mahyar y otros (2018). *Poison frogs! Targeted clean-label poisoning attacks on neural networks*. NeurIPS 2018.

Tramèr, Florian; Zhang, Fan; Juels, Ari y otros (2016). *Stealing machine learning models via prediction APIs*. Proceedings of the 25th USENIX Security Symposium.

Vorobeychik, Yevgeniy; Kantarcioglu, Murat (2018). *Adversarial machine learning*. Morgan & Claypool.

Zhang, Yuheng; Jia, Ruoxi; Pei, Hengzhi y otros (2020). *The secreto revealer: generative modelo-inversion attacks against deep neural networks*. arXiv.