# Säkerhet i AI-system (DV2607)

## Inlämningsuppgift

Martin Boldt
Anton Kanerva
Institutionen för Datavetenskap (DIDA)
Blekinge Tekniska Högskola

2023-11-01

## Introduktion

I likhet med alla mjukvarubaserade system kan även maskininlärningsystem (ML-system) och dess tränade modeller attackeras av personer med illasinnat uppsåt, t.ex. genom att skapa bakdörrar som lurar klassificeringsmodeller eller som tar bort nyttan med ML-modellen. Därför krävs också säkerhetsåtgärder för att kunna säkerställa att ML-modellerna kan utföra sina uppgifter på de sätt de är ämnade att utföras.

Denna inlämningsuppgift är uppdelad i två delar där ni i respektive del får stifta bekantskap med två vanliga attackertyper på ML-system, nämligen *Data poisoning attacker* (i del 1) respektive *Adversarial input attacker* (i del 2).

## Praktiskt upplägg och betygsättning

Respektive del av inlämningsuppgiften innehåller förklaringar, uppgifter ni ska utföra samt frågor ni ska besvara i en rapport. Rapporten betygssätts därefer med betygsskalan *A-F*. För att erhålla något av betygen *C, D eller E* behöver ni minst genomföra de obligatoriska uppgifterna och frågorna under Del 1 respektive Del 2 nedanför. För att ha en chans att få betyg *A eller B* krävs dessutom att ni genomför det som beskrivs i de frivilliga delarna för både Del 1 och Del 2 nedan.

Betygskriterierna är som följer. Om ni siktar på betyg i skalan C, D eller E måste ni slutföra uppgifterna 1.1 t.o.m. 1.4 och 2.1 t.o.m. 2.3. För betyg B måste ni dessutom slutföra uppgift 1.5.1 och 2.4. För betyg A måste ni dessutom slutföra uppgift 1.5.2.

Notera att det är obligatoriskt att jobba i grupper om **två (2)** studenter då ni genomför denna inlämningsuppgift respektive skriver er gemensamma rapport.

När ni är klara med er skriftliga rapport lämnar ni in en ZIP-fil som innehåller er: (1) Pythonkod samt (2) er rapport i PDF-format. ZIP-filen skickas in på kursens Canvas-sida innan deadline, se mer info på kurssidan. Innan ni skickar in er inlämningsuppgift så kontrollera att ni:

1. använder den obligatoriska rapportmallen som finns på kurssidan,
2. utförligt besvarat uppgifterna i både Del 1 och Del 2 nedan,
3. inkluderar all er Pythonkod som behövs för att köra era uppgifter (inkl. Jupyter Notebooks från Del 2),

4. skrivit båda personernas namn och e-postaddress i er rapport samt i headern i all källkod,
5. lämnar in er rapport i PDF-format, och
6. Packat ihop er kod och er PDF-rapport i en ZIP-fil som ni lämnar in.

# Del 1: Data poisoning attacker

*Data poisoning* är en attack-typ där maskininlärningsmodellen attackeras indirekt via modellens träningsdata. Attacken är vanligast förekommande på live-tränade modeller, dvs modeller som kontinuerligt tränas om på data som den får från det system som den är "deployad" i. Ett exempel på ett sådant system kan vara ett spam-filter för e-mail som får input av användarna som manuellt markerar e-mail som spam, varvid modellen, tillsynes harmlöst, kan tränas för att bli bättre på att upptäcka spam.

Ett sådant system är dock inte nödvändigtvis helt harmlöst. Om en grupp användare bestämmer sig för att markera tillräckligt många legitima mail som spam kan modellen felaktigt lära sig att flagga legitima mail som spam för alla användare som använder tjänsten. Del 1 i denna inlämningsuppgift går ut på att få en grundläggande förståelse kring hur maskininlärningsmodeller kan attackeras via träningsdata och hur attack-typen *data poisoning* fungerar. Dessutom kommer ni undersöka mer kring vilka säkerhetsåtgärder man kan vidta för att förhindra attacken.

Vid utförandet av uppgiften kommer ni att attackera ett ML-baserat demosystem som utför binärklassificering på bankkunders kreditdata. Systemet analyserar en valfri bankkunds data-features för att bestämma om denne ska få kredit (klass 1, positiv) eller ej (klass 0, negativ). Det finns en mängd färdiga funktioner i systemet som ni ska använda er utav, t.ex. att ta emot nya instanser av kreditdata (för bankkunder) tillsammans med etiketter som anger huruvida instanserna är positiva (1) eller ej (0). Målet är att få systemet att försämras avsevärt genom att lägga till instanser i träningsdatan.

Deluppgift 1 består av dels en teoretisk del där ni ska läsa på om tillgängliga säkerhetsåtgärder mot *data poisoning* och redogöra för dessa. Samt en praktisk del där ni ska implementera ett attack-skript och dessutom implementera en skyddsmekanism mot data poisoning attacker, t.ex. genom att modifiera klassificeringssystemet som ML-modellen använder.

# Systembeskrivning

På kurssidan finns de Python-skript som denna inlämningsuppgift bygger på. För denna del av inlämningsuppgiften är det koden i biblioteket `Part 1` som är aktuell. I detta bibliotek finns Python-skript med i förväg implementerade funktioner för en klient `client.py` och server `server.py`.

Inlämningsuppgiften har skrivits med Python version 3.9.1, och den kräver att det finns ett antal Python-bibliotek installerade, t.ex. `Numpy`, `Pandas`, `Sklearn`, `Flask` och `Pickle`. Om du saknar något av paketen så installera det med den pakethanterare du föredrar, t.ex. `pip` eller `conda`. När samtliga paket är installerade startar du servern i terminalen med kommandot: `python server.py`.

När servern startat lyssnar den på datorns lokala TCP-port 5000. Det är alltså denna port som klienten ska ansluta till för att kommunicera med servern. Detta görs automatiskt och är inget ni behöver hantera.

Nedan följer kortare beskrivningar av både serverns såväl som klientens API.

### Server-API (server.py)

Servern har bl.a. följande funktioner som ni kan komma åt:

**add_data()** Lägger till mottagna rader i träningsdatan om de är på rätt format. Returnerar hur många av de mottagna raderna som lyckades respektive misslyckades att läggas in. Denna anropas från klienten med flaggan `-e`, alltså såhär: `python client.py -e`

**train_model()** Tränar modellen på all den träningsdata som finns lagrad på servern. Denna anropas från klienten med flaggan `-t`.

**model_report()** Testar den senast tränade ML-modellen med test-datan och returnerar en klassifikationsrapport. Denna anropas från klienten med flaggan `-m`.

**reset()** Återställer modell(er) och träningsdata på servern till ursprungsläget. Denna anropas från klienten med flaggan `-r`.

### Klienten (client.py)

I detta skript finns några simpla funktioner för att kommunicera med serverns API. Beskrivningar av dem finns som DocStrings i källkoden samt genom hjälpfunktionen som körs med flaggan `-h`. Utöver funktionerna finns i klientens källkod även en lista med namnet `example_input` som visar dataformatet som serverns API förväntar sig. Variabeln `UID` ska alltid komma först i inputten och får inte ändras. För att starta klienten kör ni följande kommando i teminalen: `python client.py [flagga]`.

## Del 1: Uppgifter för betyg C, D eller E

I denna del ska ni utföra ett antal praktiska deluppgifter som både involverar att genomföra en data poisoning attack men även där ni får resonera kring försvarsmekanismer. Notera att ni har tillgång till all färdig källkod i dessa uppgifter. Därmed följer eget ansvar för att inte fuska till sig önskat resultat genom att modifiera koden på ett sådant sätt som inte är tänkt för att klara uppgifterna. Det kommer ju ändå upptäckas vid examinationen så ni lämnar in all er källkod.

**1.1) Baseline prestanda** Starta servern genom att köra `python server.py` i en terminal. Använd sedan en andra terminal för att begära en klassifikationsrapport m.h.a. funktionen *model_report* i *client.py*. Resultatet blir baseline för hur bra ML-modellen fungerade innan ni attackerade den. Skriv ner dessa resultat i rapporten.

**1.2) Attack** Implementera funktionen `launch_attack` i filen `client.py`. Ett tips är att använda funktionen `submit_data` för att skicka data till serverns API. Använd därefter er egen funktion för att attackera servern med så pass mycket felaktig data att ni når en ROC-AUC på 50%, eller lägre, jämfört med baseline. Notera och jämför resultaten från den ursprungliga baseline-rapporten.

Koden i `server.py` får **inte** modifieras i denna del av uppgiften. Det är alltså endast klienten som får modifieras här, se kommentarerna i källoden för funktionen `launch_attack` som anger var ni ska skriva in er attack-kod. Beskriv i er

rapport exakt hur ni valde att attackera ML-modellen samt hur många instanser ni behövde injicera för att lyckas med attacken.

**1.3 Skyddsåtgärder mot data poisoning** Beskriv utförligt i er rapport de *två* mest lämpade skyddsåtgärderna som ni hittat mot data poisoning attacker. Motivera varför ni valt just dessa två.

**1.4 Riskanalys av ML-systemet** Utför en riskanalys enligt den metod som beskrivs i Thomas Peltiers bok "Information Security Risk Analysis" som finns digitalt tillgänglig här[1]. Börja med att identifiera tillgångarna i systemet och gå därefter igenom var och en, och för vardera försök hitta så många risker som möljigt mot dessa tillgångar. När ni är klara med detta har ni en lista med risker, och då går ni igenom dem i ordning och estimerar *sannolikheten* för att risken kan inträffa respektive *konsekvensen* för systemet om den inträffar. Värdera båda på skalan 1-5 enligt följande:

1. Mycket låg/liten
2. Låg/liten
3. Medel
4. Hög/stor
5. Mycket hög/stor

Multiplicera därefter sannolikheterna med konsekvenserna för respektive risk för att räkna fram ett *riskvärde*. Detta värde kommer ligga mellan 1 (både sannolikhet och konsekvens på 1) upp till maximala 25 (både sannolikhet och konsekvens på 5). Sortera därefter era risker från högsta riskvärde till lägsta. Sista steget är att gå igenom var ocn en av riskerna och ge förslag på den bästa skyddsmekanismen ni kan hitta.

Beskriv resultatet av er riskanalys i rapporten. Det räcker med en tabell som har kolumnerna:

- risknamn,
- riskvärde (1-25),
- sannolikhet (1-5),
- konsekvens (1-5),
- kort beskrivning av risken, och
- förslag på skyddsmekanism.

Varje rad i tabellen beskriver därmed en enskild risk som ni identifierat i ML-systemet. Sortera listan på riskvärdet i fallande ordning.

## 1.5) Frivilliga uppgifter (krävs för betyg A och/eller B)

**1.5.1) Utökad attackanalys (Krävs för betyg B och A)** Återställ originalmodellen i servern m.h.a. av klientens funktion `reset`. Genomför därefter er poisoning attack genom att injicera följande olika mängder "förgiftade" instanser: 5%, 10%, 20%, 40%, 60%, 80% av totala antalet instanser i träningsdatan. För var och en av dem så spara undan följande prestandamått: *accuracy*, *F1* och *AUC*. Glöm inte att återställa modellen efter var och en av dessa körningar.

[1] Se PDF som ligger i uppgiften på Canvas.

När ni har all data ska ni skapa tre stycken linjeplottar, en för varje prestandamått. På linjeplotten för exempelvis *accuracy* så ska X-axeln visa andel av träningsdatan som har attackerats (alltså 5-80%) och Y-axeln visar det valda prestandamåttet. Inkludera de tre plottarna som skärmdumpar i er rapport och skriv en analys i text som beskriver vad ni drar för slutsatser från dessa.

**1.5.2) Implementation av försvar (Krävs endast för betyg A)** Återställ originalmodellen i servern m.h.a. av klientens funktion `reset`. Modifiera därefter `server.py` (och/eller dess tillhörande utility-funktioner i `utils/ml_utils.py`) för att implementera en skyddsåtgärd mot er egen data poisoning-attack. Notera att (1) användaren fortfarande ska kunna göra obegränsat med inmatningar (men vad som händer med den inmatade datan bestämmer ni) och (2) att rimligt användande av servern fortfarande upprätthålls (alltså inte tillåtet med orealistiska svarstider).

Utför därefter er attack på nytt och träna om modellen för att generera en ny rapport med prestandamått för ML-modellen. Förklara i er rapport vilken skyddsmekanism ni valt och hur den fungerar. Notera även prestandamåtten efter att ni implementerat skyddet i er rapport. Analysera hur detta förhåller sig jämfört med baseline samt ev. påverkan ert skydd haft på servern, t.ex. vad gäller svarstider.

# Del 2: Adversarial input attacker och skydd

*Evasion attack* är en attack-typ där maskininlärningsmodellen attackeras genom att modellen luras till att ta felaktiga beslut. När man talar om *evasion attacks* syftar man oftast på en attack som heter *adversarial input attack*. *Adversarial inputs* är instanser vars värden har manipulerats för att klassificeras till en annan klass utan att värdena egentligen representerar den klassen. Attacken är enklast utförd och demonstrerad på bilddata, då man enkelt kan lägga ett så pass lite brus på en bild att det är osynligt för människoögat, men får modellen att klassificera bilden som något helt annat än vad vi ser på bilden. Den går dock även att utföra inom andra datatyper såsom text- eller tabelldata, även om den senare medför en del utmaningar för att obfuskera "bruset".

Denna del av inlämningsuppgiften går ut på att få en grundläggande förståelse kring hur maskininlärningsmodeller kan attackeras via input i klassifikationssteget genom *adversarial input attacks*, samt vilka säkerhetsåtgärder man kan vidta för att försvåra eller kanske t.o.m. förhindra sådana attacker. Ni kommer att träna en modell på ett dataset för att sedan attackera den resulterande modellen med adversarial inputs som ni genererar för att lura modellen. Ni ska även undersöka och läsa in er på tillgängliga säkerhetsåtgärder mot *adversarial input attacks* och redogöra för dessa.

Uppgiften delas upp i följande två huvudsakliga delar: i) en teoretisk sammanfattning om adversarial input attacks och säkerhetsåtgärder som kan/bör vidtas, samt ii) en praktisk implementation av attacken och en eller flera försvarsmekanismer.

## 2.1 - Instudering

Innan ni implementerar attack och försvar för adversarial input attacks är det nödvändigt med en grundläggande förståelse för dessa. Ni ska här efterforska lite mer i detalj hur adversarial input attacks går till och vad det finns för skydd mot dem. Då det finns flera olika varianter räcker det med att välja en attacktyp och en skyddsåtgärd. Skriv ditt svar i rapportens del 2.1.

## 2.2 - Implementation av attack

I detta steg ska ni implementera en attack som gör så att bilden på koalan klassificeras som en traktor, utan att bilden ser annorlunda ut för mänskliga ögat. Mer info finns i Jupyter Notebooken under `Part 1`). Skriv er kod direkt i Notebooken och exekvera den där så att svaren syns i Notebooken när denna lämnas in. Så länge som ni kan förklara metoden får ni använda valfri metod, och såklart även redan tillgängliga paket, t.ex. Adversarial Robustness Toolbox (ART), för att lösa uppgiften.

## 2.3 - Säkerhetsåtgärder

I denna deluppgift ska ni välja ett så effektivt skydd som möjligt mot just er adversarial input attack. Eftersom det finns många olika typer av skydd ska ni motivera varför ni valt just det skydd ni gjort, istället för något annat skydd. Skriv ner era svar i rapportens del 2.3, och var noga med att motivera valet av skyddd väl.

## 2.4 - Implementation av skydd (betyg A eller B)

I rapportens del 2.4 summerar ni vilket skydd ni uppnådde då skyddsmekanismen hade implementerats. Det finns många olika varianter av skydd, och det är vanligt att skydden resulterar i en ny modell som är härdad mot attacker. Vid implementation av ett sådant skydd redovisar ni en jämförelse av resultaten

från er attack på originalmodellen med resultaten på den härdade modellen. Huvudsaken här är att ni tydligt kan uppvisa skyddets effekt och att ni kort kan redogöra för hur skyddsmekanismen fungerar.

Lycka till!