DV2607 Uppgift 1 Rapport

November 27, 2022

Student 1	Namn	Emil Karlsström
	E-Mail	emkl19@student.bth.se
	Program	DVAMI19h
Student 2	Namn	Samuel Jonsson
	E-Mail	sajs19@student.bth.se
	Program	DVAMI19h

1 Del 1: Data poisoning attacker

1.1 Baseline-prestanda

Algorithm:	Random Forest
Last trained:	2022–09–20 11:49
Trained on:	20100 entries
Accuracy:	81.67\%
Precision:	79.8\%
Recall:	81.67\%
F1–score:	79.79\%
ROC–AUC:	65.7\%

1.2 Attack

Algorithm:	Random Forest
Last trained:	2022–10–04 13:30
Trained on:	40200 entries
Accuracy:	49.05\%
Precision:	65.34\%
Recall:	49.05\%
F1–score:	53.43\%
ROC–AUC:	49.18\%

Efter att ha försökt med flera olika metoder, såsom att slumpa enskilda, och många kolonner, slumpa inom samma standardavvikelse och median, nollat enskilda, många och alla kolonner, så är det bästa vi har åstakommit ett fall på drygt 23%.

1.3 Skyddsåtgärder mot data poisoning

De skyddsåtgärder vi har valt är

1. **Offline/Online inspection**: Eftersom vi har tillgång till den förgiftade datan så kan vi applicera offline/online inpection på datan och använda oss av Spectral signature defense. Detta innebär avlägsning av uteliggande värden och sedan träna om modellen. Detta är byggt på idén att avlägsnandet av uteliggande värden huvudsakligen kommer påverka förgiftande värden, och därmed ta bort majoriteten av dem (Boldt, 2022).

2. **Post Backdoor removal**: Igen, eftersom vi har tillgång till datan, så kan vi även implementera Post Backdoor Removal försvarstekniken. Detta innebär att man tar förgiftade element och rättar dess label, sedan tränar om modellen med all data, men denna gång med korrekt a labelvärden (Boldt, 2022).

1.4 Riskanalys av ML-system

1.5 Högre betyg

1.5.1 Utökad attackanalys

För att komma ner till 50% ROC-AUC värde behöver man injicera minst 100% ny data om den enda attack metoden du använder är label-flip. Innan dess så sjunker värdena väldigt långsamt. Det är en betydande skillnad mellan 75- och 100% giftpunkter, då accuracy går från runt 80%till under 50%.

Från detta kan man dra slutsatsen att man behöver lägga till en stor mängd giftig data för att göra en skillnad, men eftersom datan är lätt att generera (speciellt i ett whitebox scenario), så kan man göra en stor mängd skada med en relativt liten prestation.

,

Table 2: Riskanalystabell. $\hat{p}$ = sannolikhet; C = konsekvens; RV = Risk Värde

Risk	$\hat{p}$	C	RV	Beskrivning	Skyddsmekanism
Anyone can upload	5	5	25	Vem som helst kan ladda upp data	Implementera ett autentiserings- eller verifieringssystem
No defences	4	5	20	Det finns skyddsmekanismer implementerade i systemet	Implementera skyddsmekanismer emot attacker på AI system.
Anyone can reset	5	3	15	Vem som helst kanåterställa algoritmen	Implementera ett autentiserings- eller verifieringssystem.
Anyone can train	5	2	10	Vem som helst kan träna algortimen	Implementera ett autentiserings- eller verifieringssystem.
DoS vulnerability	2	2	4	Systemet är svagt mod DoS attacker	Begränsa användarinput till en viss mängt gånger per minut.

Table 3: Attackanalys

n%	Accuracy	F1-Score	AUC
1	81.59	79.70	65.58
5	81.71	79.94	66.08
10	81.52	79.70	65.69
25	81.45	79.63	64.54
50	80.16	78.50	65.54
75	75.05	74.57	61.64
100	49.05	53.43	49.18

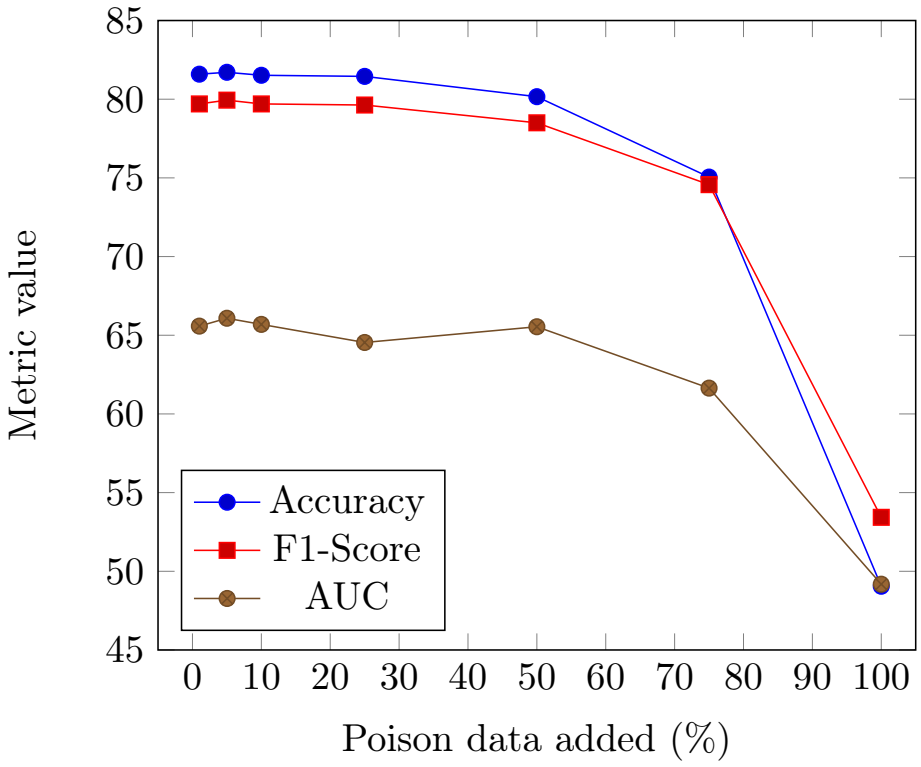

Figure 1: Attackanalys graf

1.5.2 Implementation av försvar

```
Algorithm:      Random Forest
Last trained:   2022-10-04 13:30
Trained on:     20100 entries
Accuracy:       81.55\%
Precision:      79.64\%
Recall:         81.55\%
F1-score:       79.67\%
ROC-AUC:        65.56\%
```

Vi lade till en funktion som heter `safe_to_add` i `ml_utils.py` som kollar om en datapunkt (minus label), redan är tillagd i datasetet, vilket fungerar eftersom vi endast flippar label i vår attack.

Med försvar tog det 16 sekunder att attackera, och utan tog det 33 sekunder. Detta kan bero på att den inte lägger till onödiga datapunkter, vilket gör den snabbare.

2 Del 2: Adversarial input attacker

2.1 Vad är adversarial input attacker?

Adversarial input attacker baseras på att man skapar ett "adverial sample" av en eller flera datapunkter, t.ex. genom att introducera brus i datan. Denna ändrade data används sedan för att förvilla ett neutralt nätvärk att klassificera datapunkterna som något annat. De exempel som ofta kommer upp är bildigenkänning där förändring i bara några pixlar kan får ett neutralt nätverk att klassificera en bild som något helt annat (Huber, 2020-12-29/n.d.).

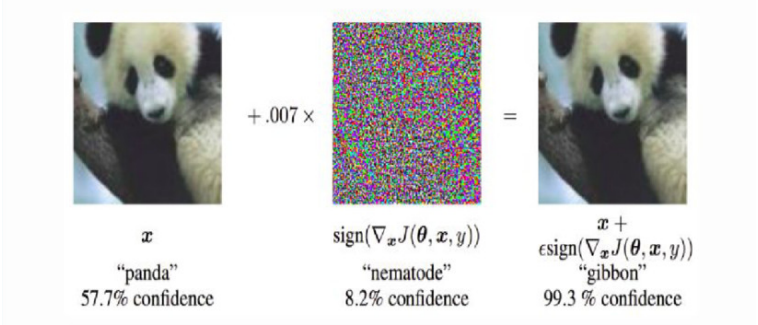

Figure 2: Det exempel som ofta tas upp när det kommer till att förklara vad advesarial inout attacker är. I exemplet intruduceras ett brus på enbild av en panda, vilket får nätverket att klassifiera pandan som en gibbonapa. bild tagen från (Huber, 2020-12-29/n.d.)

Den första metoden som introducerades inom området av Goodfellow, Shlens, and Szegedy (2014) var FGSM, som bygger på att använda sig av nätverkets gradienter för att ändra vart den klassifierar en specifik datapunkt. Problemet med denna metod är att den skapar pertuberingar som kan ses av det mänskliga ögat, såsom FGSM. Detta är naturligtvis inte optimalt, men den tjänar sitt syfte att visa att det går att påverka neurala nätverk på ett negativt sätt. Utifrån denna grund har nya metoder utvecklats, som till exempel den metod vi använder i denna uppgift, *Projected Gradient Descent* (PGD) metoden, som ger bättre resultat där man knappt kan se ändringarna, ens om man vet att de är där. PGD använder sig av samma metoder som FGSM, men gör så i mindre, upprepade steg, vilket är varför ändringarna är så mycket midre (Huang, Menkovski, Pei, & Pechenizkiy, 2020).

Eftersom adveseriella attacker bygger på data som finns i datasettet, och ofta är svåra att uptäcka, är de även svåra att skydda sig mot. Det finns även ingen "silver bullet" som löser alla hot mot den modell man utvecklat. Det har därför utvecklats fler specialiserade metoder för att hantera specifika hot. Exempel på dessa är *Adversarial instance detection*, *Maximum Mean Discrepancy* (MMD) eller *Shattered Gradients*

(Boldt, 2022). Man kan implementera en eller flera av dessa metoder, men det de har gemensamt är att de ofta försämrar accuracy hos modellen, men i utbyte blir den mer robust.

2.2 Implementation av attack

Vi bestämde oss för att använda oss av *Projected Gradient Descent*, vilket är en white-box attack. Vi använde oss av Advesarial Robustness Toolbox (ART) modulen för PGD (The Adversarial Robustness Toolbox (ART) Authors, 2018b) till Python för att implementera attacken.

Vi börjar med att skapa vår estimator och modell:

```
keras_estimator = KerasClassifier(model,
                      clip_values=(0,255))

pogboi = ProjectedGradientDescentNumpy(
    estimator=keras_estimator,
    targeted=True,
    eps=10,
    eps_step=0.5,
    max_iter=100,
    random_eps=True,
    batch_size=1024
)
```

där vi kom fram till värdena på varje parameter huvudsakligen genom att testa oss fram för att få det resultat vi ville nå. Till en början testade vi med ett lägre `max_iter` värde och använda oss av en for-loop, men vi insåg fort att detta var onödigt.

Under *modellering* skapar vi sedan en numpy array med formatet `(1, 224, 224, 3)` fylld med nollor för att spara vårt advesariella element.

Vår attack börjar sedan med att vi laddar in vår koalabild och formaterar om den till samma numpy array format som det vi har till vår advereiella numpy array och genererar vår adveseriella bild innana vi sedan testar modellen på den:

```
attack_image = init_image.reshape((1, 224, 224, 3))

attack_image = pogboi.generate(
    x = attack_image,
    y = np.array(target_id).reshape((1,)),
    mask = np.array(1.).repeat(224*224*3).reshape((1,224,224,3))
    )
gen = attack_image
```

Resultatet blir en bild som klassas som *tractor (100%)*, men är oskiljbar från originalbilden med det mänskliga ögat.

2.3 Säkerhetsåtgärder

Den metod vi tänkt använda i denna uppgift är *JPEG compression* metoden, som beskrivs mer utförligt av Das (2017) i hans artikel *Defending AI with JPEG Compression*, men kortfattat innebär den att man använder sig av JPEG kompression för att ta bort delar av bilden som mänskliga sinnet inte uppfattar, vilket är de delar av bilden som de flesta Advesarial Input attacker påverkar.

Anledningen till att vi bestämde oss för denna metod är att den verkade effektiv för syftet, samt att den var relativt enkel att implementera. Då den attack vi valt (PGD) påverkar enstaka pixlar tror vi att det är en effektiv metod att "sprida ut" pixlarnas värden med en kompression i syfte att komma undan attackens påverkan på bilden. Problemet är naturligtvis att accuracy blir sämre, men vi anser att det är värt det för en robustare modell.

2.4 Implementation av skydd

Vi använde oss av ARTs JPEG compression modul (The Adversarial Robustness Toolbox (ART) Authors, 2018a) för att implementera försvaret i koden.

Försvaret började med att skapa en JPEG kompressor

```
jpeg_compressor = JpegCompression(clip_values=(0,255), quality=25)
```

och sedan köra vårt adveseriella sample genom den

```
jpeg_gen = jpeg_compressor(gen)[0]
```

Resultatet blir att bilden åter igen klassificeras som en koala, dock inte med 100% konfidensgrad, utan snarare runt 70-80%, men vi anser att det är ett gott nog resultat med tanke på att bildkvalitén blir sämre i och med kompressionen.

Anledningen till att vi valde `quality=25` är att lägre än så så blev bilden för dålig för att klassificeras korrekt, varken som traktor eller koala, och om man hade högre kvalité gjorde det nästan ingen skillnad.

References

Boldt, M. (2022, 11). *Säkerhetsåtgärder.* lecture. (Personlig kommunikation)

Das, N. (2017, dec). *Defending ai with jpeg compression* (online). Atlanta, Georgia, United States: Georgia Institute of Technology -. Retrieved from `https://istc-arsa.iisp.gatech.edu/defending-ai-jpeg.html`

Goodfellow, I. J., Shlens, J., & Szegedy, C. (2014). *Explaining and harnessing adversarial examples.* arXiv. Retrieved from `https://arxiv.org/abs/1412.6572` doi: 10.48550/ARXIV.1412.6572

Huang, T., Menkovski, V., Pei, Y., & Pechenizkiy, M. (2020). *Bridging the performance gap between fgsm and pgd adversarial training.* arXiv. Retrieved from `https://arxiv.org/abs/2011.05157` doi: 10.48550/ARXIV.2011.05157

Huber, L. (n.d.). *A friendly intro to adversarial attacks.* Retrieved 2022-11-23, from `https://towardsdatascience.com/fooling-neural-networks-with-adversarial-examples-8afd36258a036`

The Adversarial Robustness Toolbox (ART) Authors. (2018a). *JPEG Compression.* Retrieved 2022-11-20, from `https://adversarial-robustness-toolbox.readthedocs.io/en/latest/modules/defences/preprocessor.html#jpeg-compression`

The Adversarial Robustness Toolbox (ART) Authors. (2018b). *Projected Gradient Descent (PGD) - Numpy.* Retrieved 2022-11-20, from `https://adversarial-robustness-toolbox.readthedocs.io/en/latest/modules/attacks/evasion.html#projected-gradient-descent-pgd-numpy`