DIGITAL FORENSICS & CO.

MACHINE LEARNING & DIGITAL FORENSICS

BY DR. NANNI BASSETTI

CHI SONO

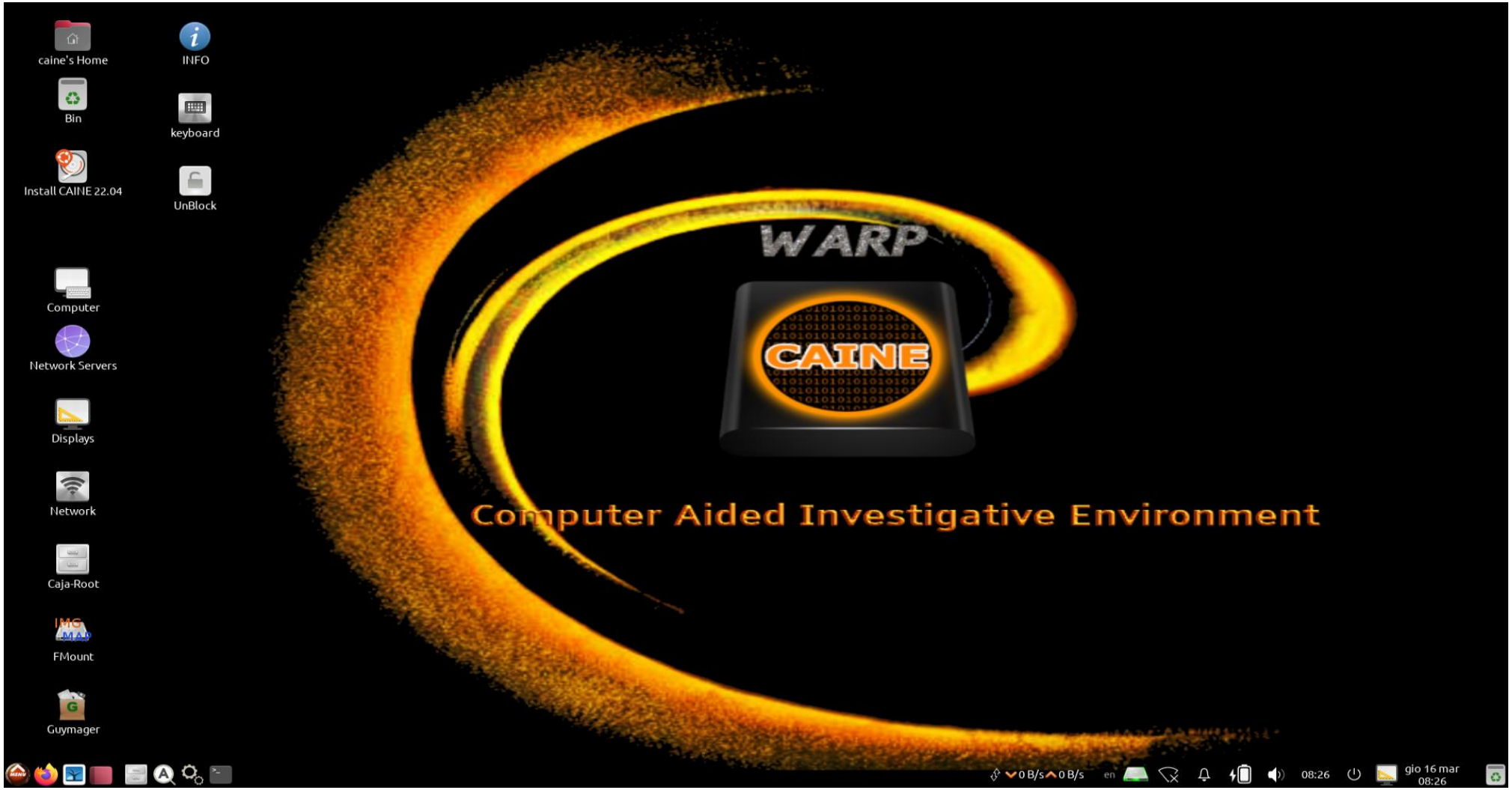
- Ciao, sono Nanni Bassetti, laureato in Informatica presso l'Università degli Studi di Bari. Lavoro come consulente di digital forensics in Italia e mi piace scrivere articoli e strumenti per questa disciplina,
- Sviluppo anche **CAINE** una distribuzione forense GNU/Linux utilizzata in tutto il mondo da consulenti e forze dell'ordine. Amo utilizzare e sviluppare software open source e sono anche socio fondatore e segretario dell'ONIF un'associazione di professionisti della digital forensics.
- Mi sono occupato di alcuni casi importanti in Italia e ho fondato CFI (Computer Forensics Italy) un'importante mailing list italiana per condividere le nostre conoscenze.
- principalmente mi piace la parte scientifica e di ricerca legata alla digital forensics

CHI SONO

CASI “MEDIATICI”

E TANTA FORMAZIONE

CAINE 13.0 «WARP» - HTTPS://WWW.CAINE-LIVE.NET/



APPLICATIONS OF ML AND GA IN DIGITAL FORENSICS

- I comportamenti dell'utente o del computer cambiano attraverso l'analisi di alcuni registri o modelli
- Trovare malware analizzando comportamenti o firme
- Rileva alcune anomalie nel traffico di rete
- Classificare alcuni file multimediali in base alla loro tipologia
- Trovare le relazioni tra gli artefatti nei Big Data
- Crea automaticamente uno storyboard partendo da eventi (ad esempio, ultimo file aperto, timeline, connessione di dispositivi USB, shellbag, modifica di data e ora, ecc.)
- Prevedere qualcosa utilizzando l'analisi dei social media
- Cracking qualcosa?

MY FORENSIC TOOLS - HTTPS://GITHUB.COM/NANNIB

Set status

Nanni Bassetti
nannib

Edit profile

digital forensics consultant
Bari - Italy
http://www.nannibassetti.com

Overview | Repositories 11 | Projects 0 | Packages 0 | Stars 51 | Followers 39 | Following 2

Pinned

Customize your pins

NBTEMPOW
NBTempoW V. 2.1 is a forensic tool for making timelines from block devices image files (raw, ewf,physicaldrive, etc.). It uses TSK (The Sleuthkit) and it has been developed with Lazarus V. 1.6.2 (...
Pascal 6 2

xall
This is a forensic data and file extractor from devices and image files. sudo ./xall_1.x.x.sh for running it. It mounts a DD/EWF image files or devices (e.g. /dev/sdb); it copies all the allocated ...
Shell 5

Raw2FS
Bash script for computer forensics - It's possible to resolve the file name starting from the carved file name generated by the Foremost tool and save it, it generates an HTML report. It's possible...
Shell 5 2

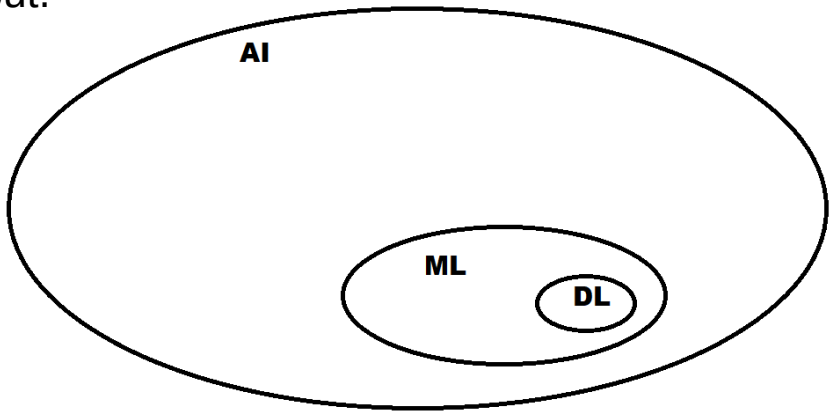
nbtempo
This is a GUI (Graphical User Interface) Bash script for making files timelines and reporting them in CSV (electronic sheet) format. It needs TSK (The SleuthKit) and YAD (Yet Another Dialog).(TSK b...
Shell 1 1

Imm2Virtual
This is a GUI (for Windows 64 bit) for a procedure to virtualize your EWF(E01), DD (raw), AFF disk image file without converting it, directly with VirtualBox, forensically proof.
Pascal 23 5

AI
some little scripts of machine learning and genetic algorithms
Python

AI – MACHINE LEARNING – DEEP LEARNING

L'IA è un macro insieme che contiene due sottoinsiemi, il Machine Learning ed il Deep Learning, essi sono due modalità d'apprendimento ed elaborazione dei dati in ingresso per ottenere delle previsioni in uscita.
L'intelligenza artificiale o IA è un campo di sviluppo dell'informatica, che tende ad emulare l'intelligenza umana e riprodurre gli schemi di ragionamento tramite algoritmi. Gli organismi biologici, come gli esseri umani, ma anche gli animali, imparano dall'esperienza e dall'osservazione del mondo, su questo concetto si sviluppano gli algoritmi che dovrebbero imparare analizzando i dati che ricevono in input.



Il concetto di IA "*general purpose*", detta anche AI forte o generale; quindi, quella che viene rappresentata nei film di fantascienza, è ancora un obbiettivo irraggiungibile.
Lo stato attuale dell'arte è rappresentato dall'IA *ristretta*, ossia dedicata allo svolgimento di specifici compiti, come la guida autonoma dei veicoli, la classificazione per immagini, la previsione di alcuni valori numerici, e.g. prezzi, azioni di borsa, la classificazione dei testi, etc...

TIPI DI APPRENDIMENTO

SUPERVISIONATO – le variabili di uscita, gli output sono mappati con una label, quindi nel dataset vi sono l'input e gli output attesi. Es.: 1 10 23 gatto 5 6 9 cane, ecc. ecc.

NON SUPERVISIONATO - nel dataset ci sono solo variabili di input senza alcun output atteso, quindi la rete deve imparare a trovare dei pattern e classificare o predire.

CON RINFORZO – è un apprendimento in ambiente dinamico, lo scopo è raggiungere un target e per farlo ci saranno delle ricompense o punizioni (numeriche, vedi Q-Learning).

CLASSIFICAZIONE – agli input vanno assegnate delle classi definite nel dataset, quindi l'algoritmo riesce a classificare un certo input (feature) ad un certa label (variabile Y) tipico dell'apprendimento supervisionato

REGRESSIONE LINEARE – le label non sono discrete ma valori continui, quindi l'algoritmo cercherà di predire la label relativa a certi input d'ingresso (es. valore azioni in borsa, ecc.) – sempre supervisionato

CLUSTERING – una classificazione per con gruppi non noti a priori, si cercano schemi che possano creare dei cluster omogenei – non supervisionato

AI – MACHINE LEARNING – DEEP LEARNING

Nel modello SUPERVISIONATO i DATASET sono formati da i campi FEATURE e la colonna LABEL

Prezzo	Oggetto
120	Occhiali
40	Camicia

Nel Deep Learning le features sono estratte automaticamente e non inputate da qualcuno, quindi è necessario un dataset più grande e più potenza di calcolo e tempo.

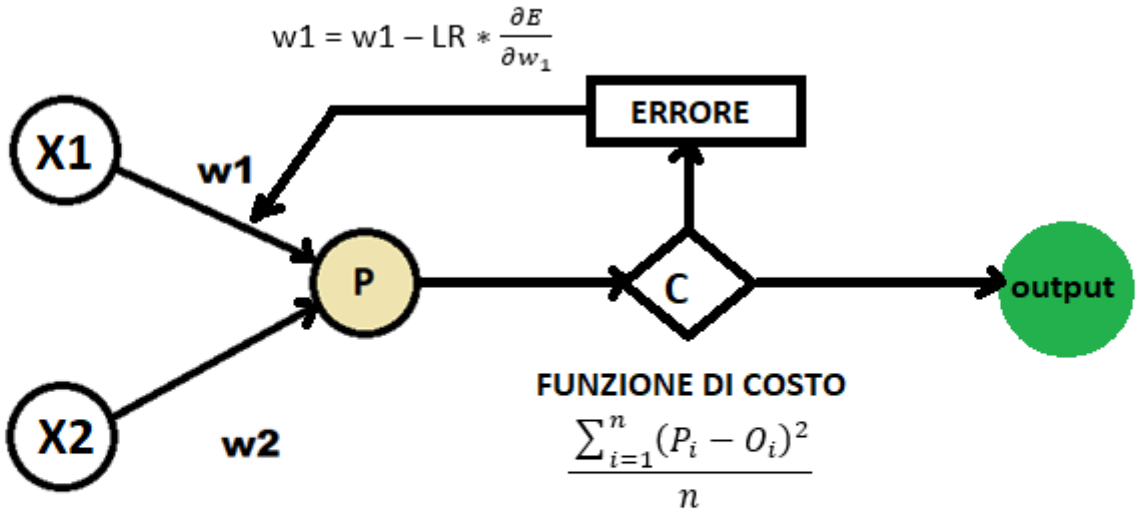
AI – MACHINE LEARNING – DEEP LEARNING

Dataset di gatti e cani nel ML

Baffi	**Orecchie**	
8 cm	5 cm	**gatto**
6 cm	20 cm	**cane**

Dataset di gatti e cani nel DL

IL NEURONE



$$\mathbf{P}(\mathbf{x1}, \mathbf{x2}) = sigmoide(\mathbf{x1} * w1 + \mathbf{x2} * w2 + b) = \frac{1}{1 + e^{-(x1*w1+x2*w2+b)}}$$

X1 e X2 sono due features del dataset, ossia due dati che caratterizzano un oggetto
Es. peso ed altezza di un cane

IL NEURONE

Il neurone si attiva tramite una **funzione d'attivazione**, che chiamerò **P(x1,x2) = x1*w1 + x2*w2 + b** dove w1 e w2 sono dei pesi random (le connessioni tra gli X input ed il neurone) e b è un numero chiamato **bias (**soglia minima per l'attivazione del neurone**).**

$$P(x1, x2) = sigmoide(x1 * w1 + x2 * w2 + b) = \frac{1}{1 + e^{-(x1*w1+x2*w2+b)}}$$

Usiamo la sigmoide per avere valori tra 0 e 1

così da facilitare la valutazione finale, questa valutazione si chiama **forward propagation**, che calcola la previsione.

Al fine di migliorare la scelta dei parametri w1,w2 e b, si deve introdurre una **funzione di costo o Loss Function**, che valuterà l'errore tra la previsione ed il risultato atteso e cercherà di minimizzarlo, questo avverrà tramite il calcolo **dell'errore quadratico medio o mean square error** (mse), dato da *somma((previsione – output)^2)/n*

$$\frac{\sum_{i=1}^{n}(P_i - O_i)^2}{n}$$

IL NEURONE

PREVISIONE	OUTPUT	ERRORE
3	5	-2
6	4	2

L'ERRORE totale sarebbe -2+2 = 0, ma se eleviamo al quadrato i singoli errori avremmo 4 e 4 con un errore di 8, l'elevamento al quadrato ci evita di azzerare o valutare male l'errore totale, causato da errori negativi e positivi.

In sintesi bisogna minimizzare l'errore tra i valori predetti ed i valori d'uscita, in modo da impostare correttamente i pesi ed il bias, nella fase di training, su un set di testing.

IL NEURONE

Poi ricalcoliamo w1, w2 e b utilizzando la formula:

$$\mathbf{w1} = \mathrm{w1} - \mathrm{LR} * \frac{\partial E}{\partial w_1} \qquad \mathbf{w2} = \mathrm{w2} - \mathrm{LR} * \frac{\partial E}{\partial w_2}$$

$$\mathbf{b} = \mathrm{b} - \mathrm{LR} * \frac{\partial E}{\partial b}$$

LR è il Learning Rate un valore (quanto devono variare i pesi) che non deve essere troppo grande o troppo piccolo che ci fornisce la velocità del calcolo e la convergenza verso la ricerca dell'errore minimo.
Se il learning rate è troppo piccolo il processo può diventare molto lento, se troppo grande potrebbe generare numeri troppo grandi, che non fanno convergere su un minimo per ridurre l'errore.
Mentre dE/dw1 è la derivata parziale della funzione di costo (Errore totale) rispetto a w1

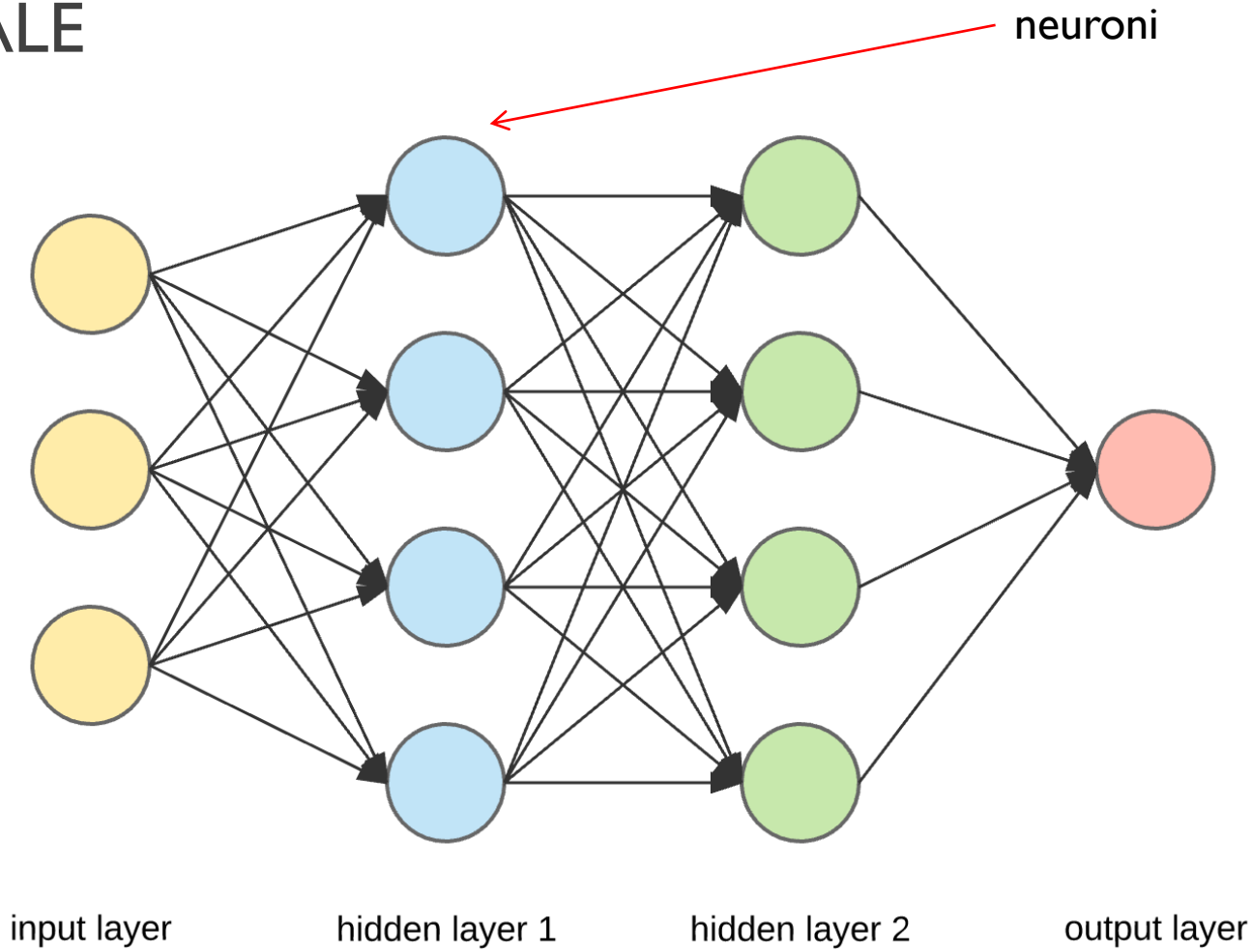
Il ricalcolo dei pesi e del bias si chiama **back propagation**.
Grazie a questo meccanismo il valore dei pesi e del bias sarà sempre più preciso al fine di rendere la funzione di costo più piccola e quindi far coincidere il valore previsto con quello di output.
Con questo sistema, la rete neurale è in grado di imparare e classificare al meglio un oggetto in input.
Quindi in sintesi gli input vanno nel neurone che calcola la previsione forward propagation, poi si ricalcolano i parametri w1,w2 e b, per minimizzare la funzione di costo, quindi avvicinarci alla previsione (back propagation), alla fine questo avanti ed indietro trova la miglior previsione e la rete impara.

IL NEURONE

SINTESI

Il ML ed il DL funzionano tramite la creazione di reti composte da neuroni artificiali, che sono algoritmi che simulano il funzionamento di un neurone biologico, tramite complesse elaborazioni matematiche.
L'algoritmo deve essere addestrato prima di poter esser considerato affidabile per le previsioni che dovrà effettuare.
La fase d'addestramento consiste nel valutare quanto un input sconosciuto si avvicini all'output atteso, calcolando la minimizzazione dell'errore (funzione di costo); quindi, se si sa che ad un dato X deve corrispondere un certo Y, l'algoritmo deve calcolare dei "pesi", valori numerici casuali, tali per cui dopo delle elaborazioni matematiche sarà sempre più preciso al fine di rendere la funzione di costo più piccola e quindi far coincidere il valore previsto con quello di output.

LA RETE NEURALE



LA MATRICE DI CONFUSIONE

Nel M.L. le regole si estraggono dai dati, non si pre-programmano, quindi se abbiamo un DATASET lo si divide in due parti Una di TRAINING e l'altra di TESTING, si danno in pasto al programma i dati di TEST ed il modello cercherà di classificarli correttamente.
Quindi si avrà una matrice di confusione per capire quanto sbaglia (confonde) e quanto è corretto.

Si crea una matrice con ASSE X rappresentante le risposte che il modello ha dato ed ASSE Y le categorie del dataset, quindi se abbiamo: CANE, GATTO e VOLPE come categorie, vediamo un modello di M.L. come potrebbe aver risposto:

	CANE	GATTO	VOLPE
CANE	2	0	4
GATTO	2	1	0
VOLPE	3	2	3

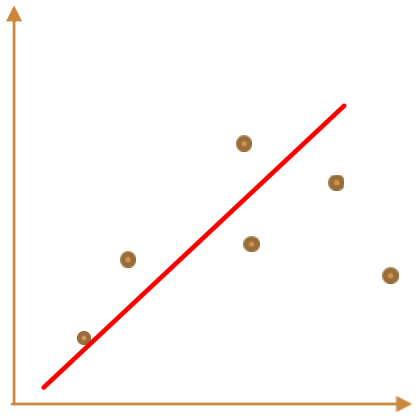
Quando nel test c'era un cane il modello lo ha riconosciuto 2 volte, poi ha sbagliato 0 volte col gatto e 4 con la volpe.
Quindi le risposte corrette sono sulla diagonale, per un totale di 2+1+3=6
ACCURATEZZA=6/17=0,3529 ossia 35,29%

THE MODEL - UNDERFITTING AND OVERFITTING

Quando l'accuratezza del training è alta c'è overfitting, quando è bassa c'è underfitting.

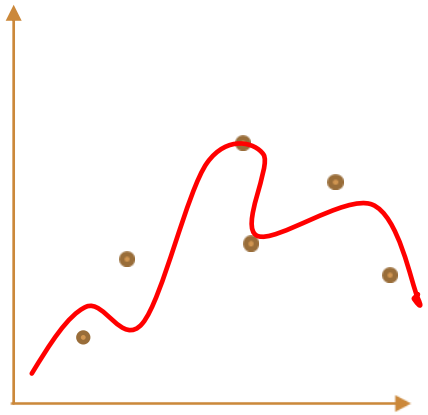
Qual è il modello migliore?

A



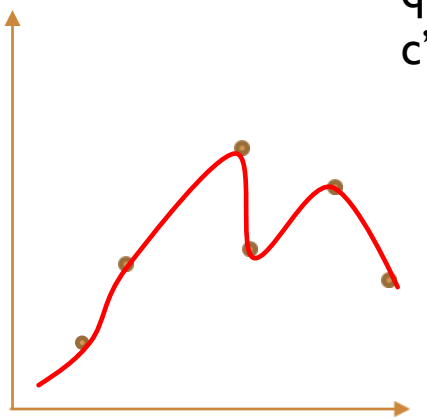
UNDERFIT

B



RIGHT FIT

C



OVERFIT

L'**underfit** generalizza male, troppo grandi gli errori
L'**overfit** è troppo preciso e generalizza male, di fronte ad un dato non presente nel dataset, non sa generalizzarlo non sa come classificarlo o prevederlo

Si regola tramite la scelta di splitting del dataset in training e test

***L'intelligenza non è la conoscenza ma è la capacità d'immaginazione* – A. Einstein.**

Error in Test data \ Error in Training data	LOW	HIGH
LOW	OK	underfitting
HIGH	overfitting	underfitting

KNN - CLASSIFICATORE

La vicinanza è un concetto che può essere geometrico o numerico, nel
k-Nearest Neighbors (k-NN), l'input è dato dal dataset, ossia gli oggetti inseriti sui quali il programma di machine learning dovrà addestrarsi per imparare e l'output è dato dal numero di oggetti più vicini definito dal parametro k.

Quindi, come in Figura 1, se k=3 allora il pallino verde (oggetto incognito) sarà classificato come simile ai triangoli rossi, perché i primi tre oggetti più vicini a lui sono due triangoli ed un quadrato, quindi è classificabile più come appartenente alla famiglia dei triangoli, se k=5 invece sarà classificato come più vicino ai quadrati blu, perché ci sono più quadrati, che triangoli, nelle sue vicinanze.

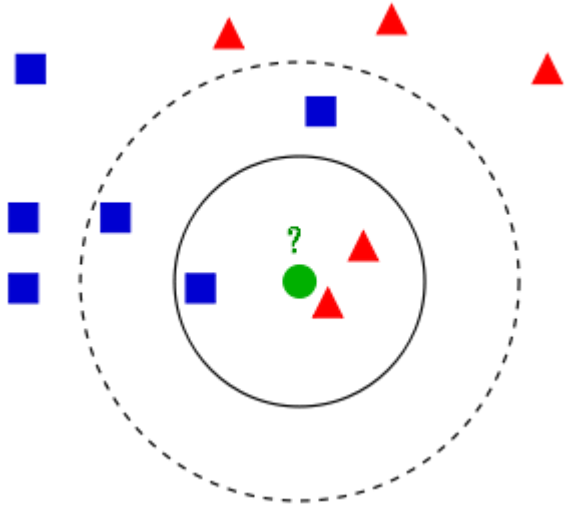


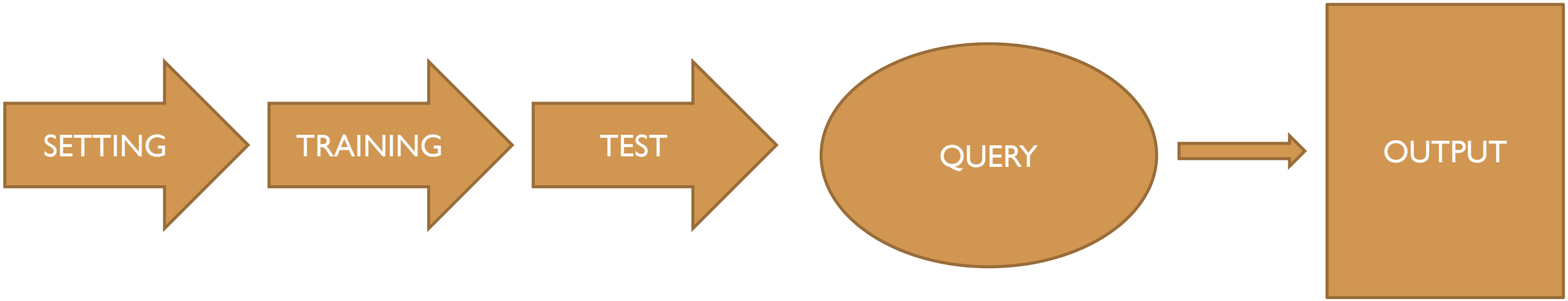
Figura 1- https://commons.wikimedia.org/wiki/File:KnnClassification.svg

La distanza che potremmo usare è quella Euclidea, ossia dati i punti [1,2,3,4] e [5,6,7,8]

La distanza è:

$$\sqrt{(5-1)^2+(6-2)^2+(7-3)^2+(8-4)^2} = 8$$

MACHINE LEARNING



Creiamo un dataset

Addestriamo e testiamo la macchina su di esso

Quindi possiamo interrogare la macchina per ottenere un output con una misura di precisione

ANOMALY DETECTION – IP ADDRESSES AND TIME

Pensiamo ad un dataset di connessioni formato da indirizzi IP ed orari di connessione, quindi un tabulato, potremmo addestrare l'algoritmo a capire qual è la connessione "anomala" e basterebbero poche righe di codice (figura 5).

L'algoritmo sfrutta l'Isolation Forest, un algoritmo non supervisionato e quindi non necessita di etichette per identificare il valore anomalo / anomalia. Il fondamento dell'algoritmo è quello di "isolare" le anomalie creando alberi decisionali su attributi casuali, questo partizionamento casuale produce percorsi più brevi per le anomalie quindi, se ci sono poche anomalie ci saranno partizioni più piccole di dati, perché i valori delle anomalie hanno maggiori probabilità di essere separati nel partizionamento iniziale.

Quindi, quando una foresta di alberi casuali produce lunghezze di percorso più brevi per alcuni punti particolari, è molto probabile che siano anomalie.

ANOMALY DETECTION – IP ADDRESSES AND TIME

Il dataset

date	time	ip
20210101	100352	192168001010
20210101	110411	192168001010
20210101	125609	192168001010
20210102	100722	192168001010
20210102	130351	192168001010
20210102	142337	192168001010
20210103	90112	192168001010
20210103	100312	192168001010
20210103	151111	192168001010
20210103	100352	110175021210
20210104	43211	192168001010
20210104	91451	192168001010
20210104	101223	192168001010
20210105	92701	192168001010
20210105	113211	192168001010
20210105	121504	192168001010
20210105	144604	192168001010
20210106	155901	192168001010
20210107	111109	192168001010
20210107	160001	192168001010
20210108	90101	192168001010
20210108	154556	192168001010
20210108	160504	192168001010
20210109	160032	192168001010
20210110	90000	192168001010
20210110	101724	192168001010
20210110	160341	192168001010
20210101	100221	192168001033
20210101	103305	192168001033

ANOMALY DETECTION – IP ADDRESSES AND TIME

beahvior.py | gab64_final.py | neural_deep.py | random_gen.py | integer-predictor.py | corona_predict.py | email_predict.py | anomaly.py*

```python
https://blog.paperspace.com/anomaly-detection-isolation-forest/
@author: nannib
"""
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import IsolationForest

df = pd.read_csv('ip.csv', names=['date', 'time', 'ip'], header=0)

F = df[['time','ip']]

#print(F)

model=IsolationForest(n_estimators=50, max_samples='auto', contamination=float(0.05),max_feature
model.fit(F)
df['scores']=model.decision_function(F)
df['anomaly']=model.fit_predict(F)

anomaly=df.loc[df['anomaly']==-1]
normal=df.loc[df['anomaly']>0]

anomaly_index=list(anomaly.index)
outliers_counter = len(df[df['anomaly'] < 0])

print(anomaly)
print ('Anomalies number: ',outliers_counter)
print("Accuracy percentage:", 100*list(df['anomaly']).count(-1)/(outliers_counter))

p1 = plt.scatter(normal, normal, c="green", s=50, edgecolor="black")
#p2 = plt.scatter(X_test.x, X_test.y, c="green", s=50, edgecolor="black")
p3 = plt.scatter(anomaly, anomaly, c="red", s=50, edgecolor="black")
plt.xlim((-0.3, 0.3))
plt.ylim((-0.3, 0.3))
plt.legend(
    [p1, p3],
    ["normal", "anomalous"],
    loc="lower right",
)

plt.show()
```

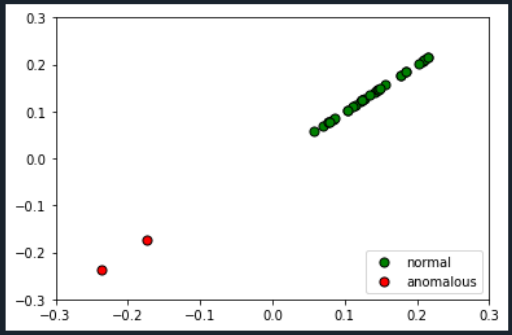
from

No further documentation available

Variable explorer | Help | Profiler | Plots | Files

Console 1/A

```
In [15]: runfile('D:/anomaly.py', wdir='D:')
        date    time            ip    scores  anomaly
9   20210103  100352  110175021210 -0.236649       -1
10  20210104   43211  192168001010 -0.174191       -1
Anomalies number:  2
Accuracy percentage: 100.0
```




```
In [16]:
```

COMPUTER BEHAVIOR

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier

# Load dataset
url = "data.csv"
names = ["time_online","cpu","data_exchanged","type"]
dataset = pd.read_csv(url, names=names,header=0)
# Split-out validation dataset
dataset= pd.DataFrame(dataset)
array = dataset.values
X = array[:,0:3]
y = array[:,3]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.6, random_state=0)
model = KNeighborsClassifier(n_neighbors=5)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
acc=accuracy_score(y_test, y_pred)
valori=[3,67,85]
prediction = model.predict([valori])
print("Accuracy:",round(acc*100,2),"%")
print("Predicted target name: {}".format(prediction)," \n",names," \n",valori,"   ",prediction)
```

LOAD DATASET

TRAIN & TEST

QUERY

OUTPUT

Dataset

time	cpu	Data_exchanged	Type
7	73	32	normal
9	66	94	normal
9	56	25	normal
8	29	58	normal
5	90	66	suspect
6	79	71	suspect
1	36	32	suspect
2	68	98	suspect

COMPUTER BEHAVIOR

```
Accuracy Train: 81.82 % Accuracy Test:  77.27 %
Predicted target name: ['suspect']
 ['time_online', 'cpu', 'data_exchanged', 'type']
 [3, 67, 85]     ['suspect']
```

SRUM (System Resource Utilization Monitor) può essere una fonte per un dataset?
Traccia l'utilizzo dell'applicazione, l'utilizzo della rete e lo stato energetico del sistema.
C:\Windows\System32\sru\SRUDB.dat. Il file è un ESE (Extensible Storage Engine) database

Per cominciare un tool open source: https://github.com/EricZimmerman/Srum ;-)

THE STATE OF THE ART

Attualmente nella cybersecurity e nella digital forensics, cioè la disciplina che regola le attività di investigazione informatica per finalità forensi, quindi giudiziarie, si affrontano spesso problematiche legate a file illeciti, che vanno dal materiale audio/video ai malware o ad altre azioni dannose per il sabotaggio, dati esfiltrazione e spionaggio.

Durante un Incident Response o un'analisi Digital Forensics, cerchiamo tracce di file che hanno causato qualcosa o che non dovrebbero risiedere sul computer, perché illegali o dannosi, per fare questo ci vuole molto tempo e attenzione, bisogna effettuare ricerche nei log, nelle timeline, a volte anche aprendo manualmente i file o eseguendoli nelle sandbox, ove possibile, anche se spesso molti malware sono dotati di tecniche di evasione, che permettono di non essere identificati nelle macchine virtuali o nelle sandbox.

Le tecniche di ricerca più comuni si basano sulla ricerca per parole chiave e sul confronto di codici hash, cioè si utilizzano grandi database contenenti codici hash di file conosciuti e se sul dispositivo in esame ci sono file che hanno quei codici allora vengono contrassegnati come sospettoso.

THE STATE OF THE ART

Ad esempio, se sul tuo computer è presente un file XYZ.exe con il codice hash:
644C8BB3CB5C73A89A128855E58EC32D

e questo codice è presente nel database dei file "cattivi", il file XYZ.exe verrà contrassegnato come sospetto, chiaramente quel codice hash potrebbe appartenere a un malware noto che può assumere vari nomi di file; quindi il nome da solo non basterebbe.
Il database di file più famoso e conosciuto è il progetto **NSRL**, supportato dal Dipartimento per la Sicurezza Nazionale degli Stati Uniti, dalle forze dell'ordine federali, statali e locali e dal National Institute of Standards and Technology (NIST), la National Software Reference Library (NSRL) è progettata per raccogliere software da varie fonti e incorporare profili di file calcolati da questo software in un set di dati di riferimento (RDS), RDS è una raccolta di codici hash di applicazioni software noti e tracciabili. Nel set di hash sono presenti valori hash dell'applicazione che possono essere considerati dannosi, come strumenti di steganografia e script di hacking. Non sono presenti valori hash di dati illegali, come immagini di abusi sui minori.

THE STATE OF THE ART

La stessa cosa vale per la ricerca per parola chiave, magari un malware noto ha al suo interno la stringa "**EVILcorp**", quindi cercando quella stringa possiamo trovarla.

Il problema più grande è che spesso i file dannosi sono leggermente diversi tra loro, finché è cambiato qualcosa in una versione del file, il codice hash cambia; pertanto non corrisponde a quello presente nel database.

Per questo motivo possiamo pensare di riconoscere la tipologia del file, per poi classificarlo attraverso i suoi metadati, cioè quei dati che vanno oltre quelli del file, come nome, percorso, dimensione, timestamp, autore, occorrenze, ecc.

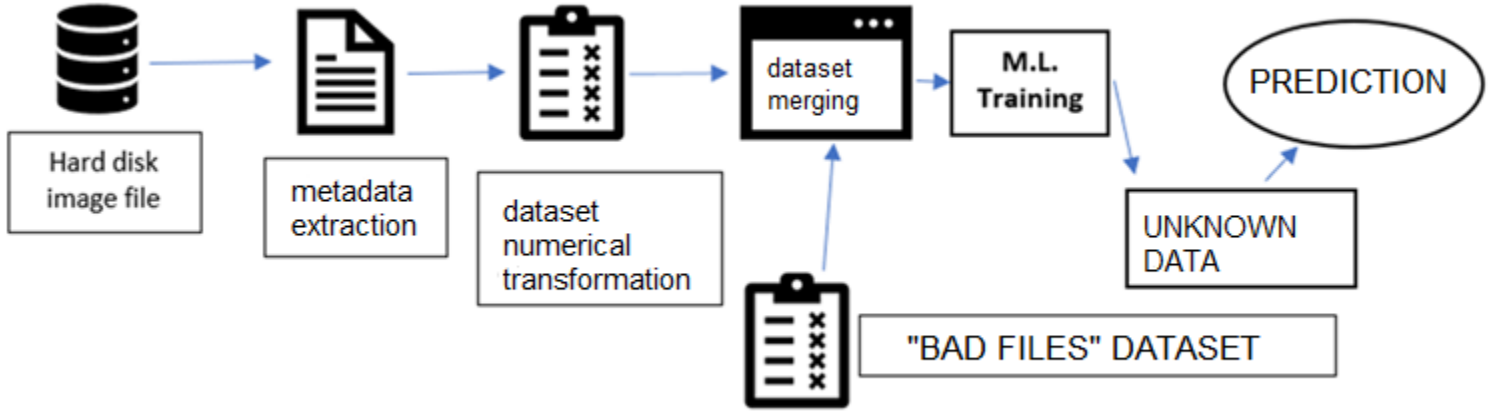
MACHINE LEARNING PER L'IDENTIFICAZIONE DEI FILE SOSPETTI

Mi sono ispirato ad un paper del 2019 ad opera di Xiaoyu Du e Mark Scanlon *"Methodology for the Automated Metadata-Based Classification of Incriminating Digital Forensic Artefacts"*
https://www.researchgate.net/publication/334192020_Methodology_for_the_Automated_Metadata-Based_Classification_of_Incriminating_Digital_Forensic_Artefacts

Cercherò di rendere più "*tangibile*" e visuale il concetto espresso nel paper e nella premessa dell'articolo, ossia quello di classificare/identificare un file dannoso tramite lo studio dei metadati ed artefatti, in modo che anche se un file ha cambiato nome, ha un contenuto leggermente differente da quello già noto e presente nei database, lo possiamo trovare addestrando un machine learning a capire dai metadati se quel file è vicino ai metadati di qualcuno già noto e sospetto.

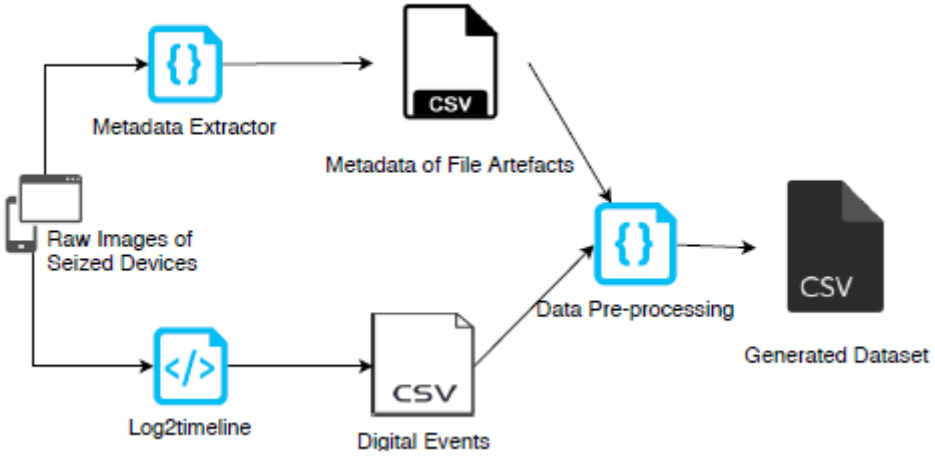
MACHINE LEARNING AND DIGITAL FORENSICS

Insomma, una specie di “profiling” del file, si osservano molti profili, chiaramente non psicologici, ma tecnici e quando ci si trova di fronte a qualcosa di nuovo, ma che ha un “comportamento” e delle caratteristiche simili ai “bad files”, la macchina dovrebbe segnalarlo.
Chiaramente l’esempio ed il codice qui riportato è rudimentale ed a scopo didattico, ma può servire a comprendere meglio lo schema:



LEARNING BY ARTIFACTS

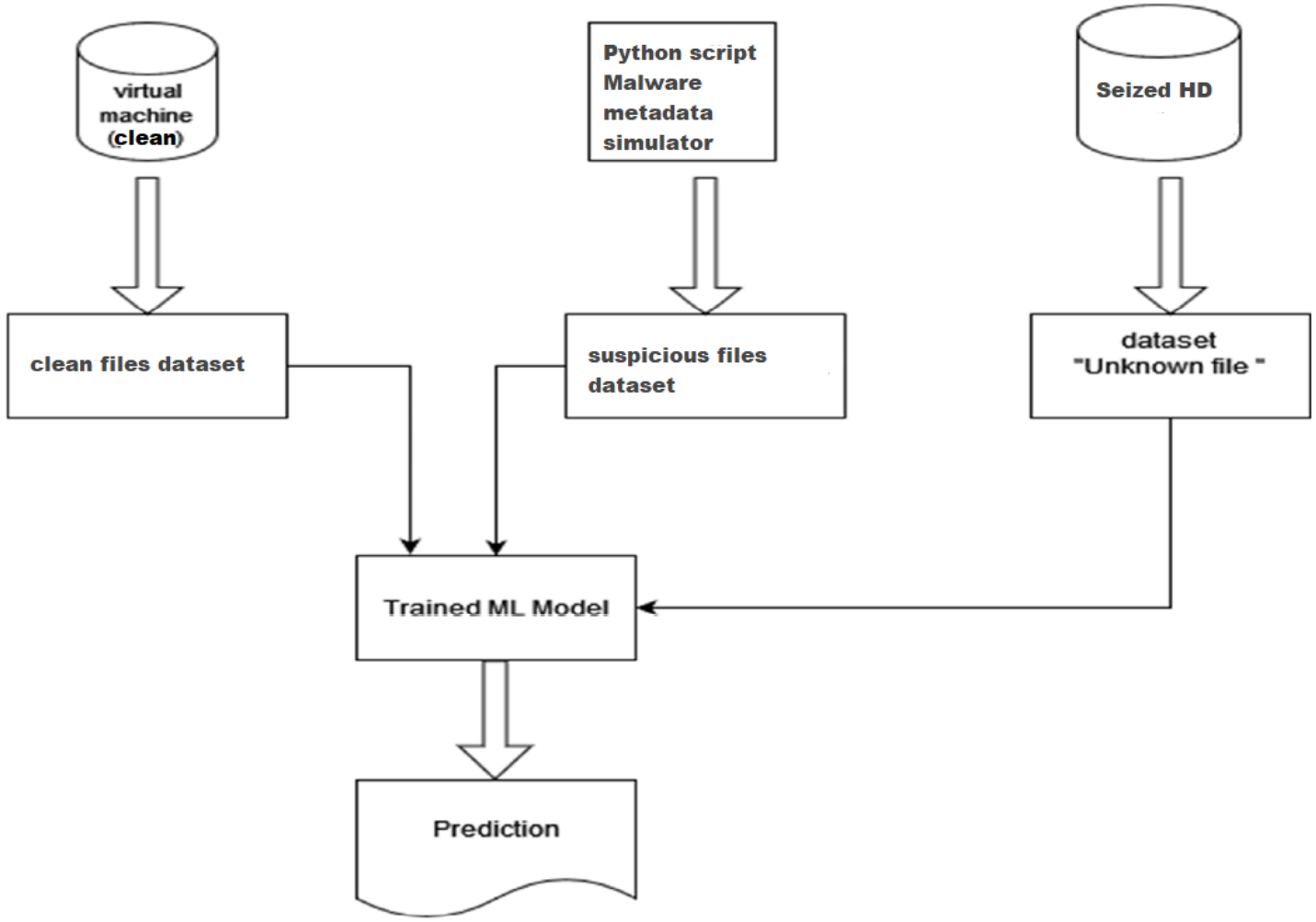
immaginiamo di estrarre da un'immagine forense di un disco, utilizzando software come Plaso (log2timeline) e trasformarli in numeri, ottenendo così un dataset che chiameremo CLEAN.csv
INFINE uniamo i due set di dati (metadata.csv) e addestriamo la macchina.



https://www.researchgate.net/publication/334192020_Methodology_for_the_Automated_Metadata-Based_Classification_of_Incriminating_Digital_Forensic_Artefacts

file_type	path_depth	size	namefile_lenght	occurences	
4	4	6074	1	51	clean
1	4	1719	5	193	clean
3	5	3277	10	180	suspect
6	6	4029	17	25	clean
1	10	3087	17	135	clean
4	10	7074	15	60	clean
5	10	3983	20	197	clean
4	5	5126	16	181	clean
1	3	967	8	127	clean
4	3	4303	5	85	clean

YES, ANOTHER FLOW CHART



Source: tesi Master del mio alunno - Ing. Lindo Nepi

IL FLUSSO DELLE OPERAZIONI

In particolare, rispetto alla serie di operazioni per l'apprendimento automatico illustrate nel precedente paragrafo, il work-flow adottato nel presente lavoro prevede le fasi:

- pre-elaborazione
- apprendimento
- valutazione
- predizione
- salvataggio del modello su file
- caricamento del modello da file e predizione su dataset "unknown file" proveniente da supporto oggetto di analisi forense.

La possibilità di memorizzare il modello su file, permette di poter effettuare previsioni senza dover ogni volta effettuare la fase di apprendimento del modello, con un risparmio di tempo e di risorse, al tempo stesso questa soluzione permette di utilizzare il modello precedentemente addestrato su macchine diverse da quella su cui è stato effettuato il training del modello.

MACHINE LEARNING AND DIGITAL FORENSICS

I passi sono come segue:

1) Estrazione dei metadati del file dal file immagine (copia forense) di un disco in cui è stata simulata molta attività normale e legale, è possibile utilizzare vari strumenti da Plaso a pyTSK.
2) I metadati vengono trasformati in valori numerici e viene creata una tabella dove le caratteristiche sono i metadati e l'etichetta è “clean”.
3) Aggiungere il set di dati dei file cattivi al set di dati "pulito", l'etichetta dei cattivi sarà "suspect".
4) Il modello viene addestrato sul database complessivo.
5) Un set di dati estratto da un disco in esame viene alimentato all'algoritmo
6) L'algoritmo effettua la sua previsione/classificazione.

Generazione del set di dati
Ai fini di questo intervento ho preferito generare un dataset casuale, per evitare tutta la parte di raccolta dei metadati e trasformazione numerica.

PARSING FROM THE DISK DIRECTLY

```
VBoxManage.exe clonehd --format RAW cleanmachine.vdi cleandisk.dd
```

PLASO OUTPUT

```
C:\plaso>log2timeline.exe --hashers sha256 Disk01.dmp Disk01.dd
2023-10-18 10:07:44,738 [INFO] (MainProcess) PID:33820 <data_location> Determined data location:
C:\plaso\data
2023-10-18 10:07:44,753 [INFO] (MainProcess) PID:33820 <artifact_definitions> Determined artifact
definitions path: C:\plaso\artifacts
Checking availability and versions of dependencies.
[OPTIONAL]      missing: lz4.
[OK]


Source path            : C:\plaso\PenLab.dd.001
Source type            : storage media image
Processing time          : 00:00:00

Processing started.
```

PLASO OUTPUT

log2timeline.exe --hashers sha256 Disk01.dmp Disk01.dd
psort -o dynamic --additional_fields sha256_hash Disk01.dmp -w Disk01.csv

A	B	C	D	E	F	G	I
datetime	timestamp_desc	source	source_long	message	parser	display_name	sha256_hash
2000-10-19T17:53:02+00:00	Creation Time	PE	PE Compilation time	PE Type: Dynamic L	pe	TSK:/my Folder/glib-1.3.dll	-
2000-10-19T17:53:02+00:00	Creation Time	PE	PE Compilation time	PE Type: Dynamic L	pe	TSK:/my Folder/glib-1.3.dll	c7fe30cbf9a71109b886a4fb4998e1293922601ad685bb6432cdb06e05505841
2000-10-19T19:53:02+00:00	Content Modification	FILE	NTFS Content Modification Tim	TSK:/my Folder/glib-	filestat	TSK:/my Folder/glib-1.3.dll	-
2000-10-19T19:53:02+00:00	Content Modification	FILE	NTFS Content Modification Tim	TSK:/my Folder/glib-	filestat	TSK:/my Folder/glib-1.3.dll	c7fe30cbf9a71109b886a4fb4998e1293922601ad685bb6432cdb06e05505841
2000-10-19T19:53:02+00:00	Content Modification	FILE	NTFS Content Modification Tim	TSK:/$MFT File refer	mft	TSK:/$MFT	-
2000-10-22T20:05:11+00:00	Creation Time	PE	PE Compilation time	PE Type: Dynamic L	pe	TSK:/my Folder/gdk-1.3.dll	-
2000-10-22T20:05:11+00:00	Creation Time	PE	PE Compilation time	PE Type: Dynamic L	pe	TSK:/my Folder/gdk-1.3.dll	a9ea35921dbbbd890db17786ffcea70611a69adb6507d316d8da83bf415aeea6
2000-10-22T22:05:12+00:00	Content Modification	FILE	NTFS Content Modification Tim	TSK:/my Folder/gdk-	filestat	TSK:/my Folder/gdk-1.3.dll	-
2000-10-22T22:05:12+00:00	Content Modification	FILE	NTFS Content Modification Tim	TSK:/my Folder/gdk-	filestat	TSK:/my Folder/gdk-1.3.dll	a9ea35921dbbbd890db17786ffcea70611a69adb6507d316d8da83bf415aeea6
2000-10-22T22:05:12+00:00	Content Modification	FILE	NTFS Content Modification Tim	TSK:/$MFT File refer	mft	TSK:/$MFT	-
2000-11-19T19:59:34+00:00	Content Modification	FILE	NTFS Content Modification Tim	TSK:/my Folder/icon	filestat	TSK:/my Folder/iconv-1.3.dll	-
2000-11-19T19:59:34+00:00	Content Modification	FILE	NTFS Content Modification Tim	TSK:/my Folder/icon	filestat	TSK:/my Folder/iconv-1.3.dll	41c83f04646f8300846878978ca471b3812750d766b34a068d7bca98d70ed6f9
2000-11-19T19:59:34+00:00	Content Modification	FILE	NTFS Content Modification Tim	TSK:/$MFT File refer	mft	TSK:/$MFT	-
2000-11-20T02:59:33+00:00	Creation Time	PE	PE Compilation time	PE Type: Dynamic L	pe	TSK:/my Folder/iconv-1.3.dll	-
2000-11-20T02:59:33+00:00	Creation Time	PE	PE Compilation time	PE Type: Dynamic L	pe	TSK:/my Folder/iconv-1.3.dll	41c83f04646f8300846878978ca471b3812750d766b34a068d7bca98d70ed6f9
2001-08-17T20:52:55+00:00	Creation Time	PE	PE Compilation time	PE Type: Executable	pe	TSK:/Downloads/00582365.exe	-
2001-08-17T20:52:55+00:00	Creation Time	PE	PE Compilation time	PE Type: Executable	pe	TSK:/Downloads/00582365.exe	113c7c85d7c144dc2967d3652fae81d0bb21ccb784551367f3078a9382edd481
2001-08-24T22:18:10+00:00	Content Modification	FILE	NTFS Content Modification Tim	TSK:/Programs/Gam	filestat	TSK:/Programs/Game1/Examples/Vb/Form1.frm	-

FINAL DATASET

Sfruttando l'output di Plaso si può creare uno script Python che seleziona i campi utili e genere altri come l'entropia, ecc.

sha256file	nomefile	percorsose	lunghezzar	profondita	dimensione	magicnuml	magicnuml	mimetype	estensione	lunghezzar	mtime	atime	ctime	timedifferen	timedifferen	timedifferen	entropy	maxfilentro	owner	eventi	label
47ad17b1	/tmp/imgpulita/Users/u		0	40	11	11	PEM certif	0	None		0	455913	456280	455913	-367	0	-367	6.01	6.01	0	0	clean
21dac95d	/tmp/imgpulita/Users/u		0	40	11	12	gzip compr	0	None		0	455915	456280	455915	-365	0	-365	7.0	7.0	0	0	clean
6618595c	/tmp/imgpulita/Users/u		0	40	11	11	PEM certif	0	None		0	455913	456280	455913	-367	0	-367	6.02	6.02	0	0	clean
94bac5cde	/tmp/imgpulita/Users/u		0	40	11	11	PEM certif	0	None		0	455913	456280	455913	-367	0	-367	6.01	6.01	0	0	clean
7abd8777	/tmp/imgpulita/Users/u		0	40	11	111	JPEG imag	0	None		0	455915	456280	455915	-365	0	-365	7.97	7.97	0	0	clean
fffc719dbf	/tmp/imgpulita/Users/u		0	40	11	10	PNG image	0	None		0	455915	456280	455915	-365	0	-365	6.69	6.69	0	0	clean
39306be0	/tmp/imgpulita/Users/u		0	40	11	11	PEM certif	0	None		0	455913	456280	455913	-367	0	-367	6.02	6.02	0	0	clean
3459410e	/tmp/imgpulita/Users/u		0	40	11	18	JSON data	0	None		0	455915	455915	455915	0	0	0	6.17	6.17	0	0	clean
4d405018	/tmp/imgpulita/Users/u		0	40	11	33	JPEG imag	0	None		0	455915	456280	455915	-365	0	-365	7.8	7.8	0	0	clean
bb7613fc7	/tmp/imgpulita/Users/u		0	40	11	44	data	1	None		0	456255	456280	456255	-25	0	-25	7.76	7.76	0	0	clean
074570cf7	/tmp/imgpulita/Users/u		0	40	11	11	PEM certif	0	None		0	455913	456280	455913	-367	0	-367	6.02	6.02	0	0	clean
ccccb46b1	/tmp/imgp	4	31	3	860	PE32+ exe	2	.dll		0	363407	416304	327770	-88534	-35637	-88534	6.77	7.24	0	4	suspect
24b1c15d	/tmp/imgp	0	31	3	1003	PE32+ exe	2	.sys		4	431968	423571	347193	-76378	-84775	-76378	6.75	6.82	0	1	suspect
35013180	/tmp/imgp	2	66	3	851	PE32 exec	2	.exe		7	416321	432021	397715	-34306	-18606	-34306	6.08	7.77	0	5	suspect
3323ace7	/tmp/imgp	4	42	3	175	PE32+ exe	2	.com		3	403055	412662	405154	-7508	2099	-7508	7.42	7.66	0	0	suspect
5b093206	/tmp/imgp	2	25	3	153	PE32+ exe	2	.sys		4	368307	421340	388834	-32506	20527	-32506	4.94	7.0	0	4	suspect
90add676	/tmp/imgp	2	28	3	113	PE32+ exe	2	.dll		7	384256	416173	415177	-996	30921	-996	7.87	7.89	0	4	suspect

METADATI

I file “puliti” hanno tipicamente un’entropia compresa tra 4.8 e 7.2

- I file con entropia > 7.2 sono tipicamente dannosi
- Quasi il 30% dei file dannosi analizzati ha entropia vicina al valore 8 (valore massimo dell’entropia dei file)
- Solo l’1% dei file puliti ha un’entropia vicina al valore 8.

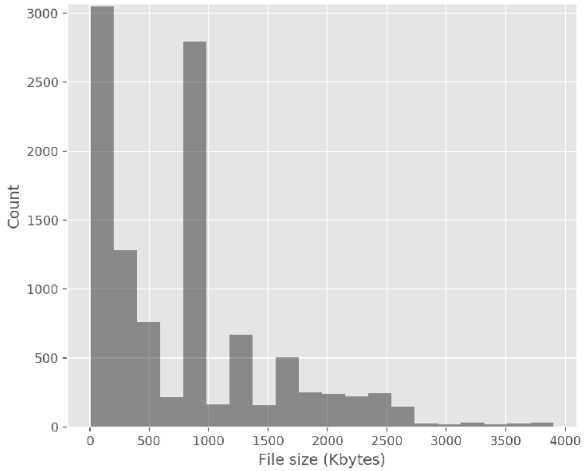
17

- Circa il 55% di tutti i campioni dannosi analizzati ha entropia > 7.2 mentre solo l’8% dei file puliti ha un’entropia maggiore o uguale a questo valore.

Per i file EXE si potrebbe indagare sulle sezioni interne (.text, .rdata, .data, .rsrc, .reloc) al file eseguibile, misurandone la singola entropia anziché limitarsi a misurare tale valore su tutto il file (entropia media).

Le dimensioni di un malware sono in genere sotto i 3Mb

Fonte: “Deep learning at the shallow end: Malware classification for non-domain experts” - Lee e Scanlon



THE BAD GUYS

Tabella delle caratteristiche dei file eseguibili contenenti malware:

Metadata type	Data type/size	Features
File size	Kb	0-1500 Kb. In 30% dei casi in cui la dimensione è inferiore a 200Kb
Entropy	Float	tipicamente l'entropia massima è tra 6.5 e 8
Path	String	C;\Users\user\AppData\ C:\Windows\temp C;\Users\user\Downloads\ C:\Users\user\[some_dir]
Type (extension)	String	PE (.exe, .dll, .sys, .com, .drv ,.ocx) , HTML, JS, PDF, ZIP
File Header (magic number)	String	Not change
Timestamps	timestamps	Differenza tra atime e mtime è accentuata nei malware

THE BAD GUYS

THE KNOWLEDGE SOURCES:

Examples:

samples of the Microsoft Malware Classification Challenge (BIG, 2015) present on Kaggle.com [https://www.kaggle.com/c/malware-detection/data?select=data.csv], from the "theZoo" [https://github.com/ytisf/theZoo] and Endermanch [https://github.com/Endermanch/MalwareDatabase] archives on Github and from the GovDocs1 file archive [https://digitalcorpora.org/corpora/files].

MAKING THE SUSPECT DATASET

```
import random
import csv

header = ['file_type', 'path_depth', 'size', 'namefile_lenght','occurences']
with open('metadata.csv', 'w', encoding='UTF8', newline='') as f:
    writer = csv.writer(f)
    writer.writerow(header)
i=0
while i<3000:
      i=i+1
      type_f=random.randint(1, 6)
      depth=random.randint(1,10)
      size=random.randint(0,10000)
      lenght_nf= random.randint(1,20)
      occurences=random.randint(0,200)
      if ((type_f==2) or (type_f==3) or (type_f==4)) and (depth>3) and (size>1024 and size<8048) and (lenght_nf>2 and lenght_nf<15) and (occurences>10):
          label="suspect"
      else:
          label="clean"
      #print(type_f,",",depth,",",size,",",lenght_nf,",",occurences,",",label)
      #print (s)
      data=[type_f,depth,size,lenght_nf,occurences,label]
      with open('metadata.csv', 'a', encoding='UTF8', newline='') as f:
          writer = csv.writer(f)
          writer.writerow(data)
random.seed()
```

MACHINE LEARNING AND DIGITAL FORENSICS

Questo programma in Python genera 3000 records dove le caratteristiche sono:
['file_type', 'path_depth', 'size', 'namefile_lenght', 'occurences']
Il tipo di file (**1** jpg, **2** exe, **3** docx, **4** pdf)
Il path_depth, es. Quante directory sotto la root (es. c: \ users \ xxx \ file.exe sono 2 directory sotto).

Poi ci sono la dimensione (size), la lunghezza del nome del file e il numero di volte in cui appare nel file system (occorrenze).

Impostando alcuni criteri arbitrari ho impostato che se il file è di tipo 2 o 3 o 4 e si trova ad almeno 3 directory da quella principale e la dimensione è compresa tra 1024 Kb e 8048 Kb e il nome del file è lungo tra 3 e 14 caratteri e si verifica almeno 11 volte, allora verrà contrassegnato come "**suspect**", altrimenti sarà "**clean**".

MACHINE LEARNING AND DIGITAL FORENSICS

This is a sample of the dataset:

file_type	path_depth	size	namefile_lenght	occurences	
4	4	6074	1	51	clean
1	4	1719	5	193	clean
3	5	3277	10	180	suspect
6	6	4029	17	25	clean
1	10	3087	17	135	clean
4	10	7074	15	60	clean
5	10	3983	20	197	clean
4	5	5126	16	181	clean
1	3	967	8	127	clean
4	3	4303	5	85	clean

LET'S START THE ENGINES!

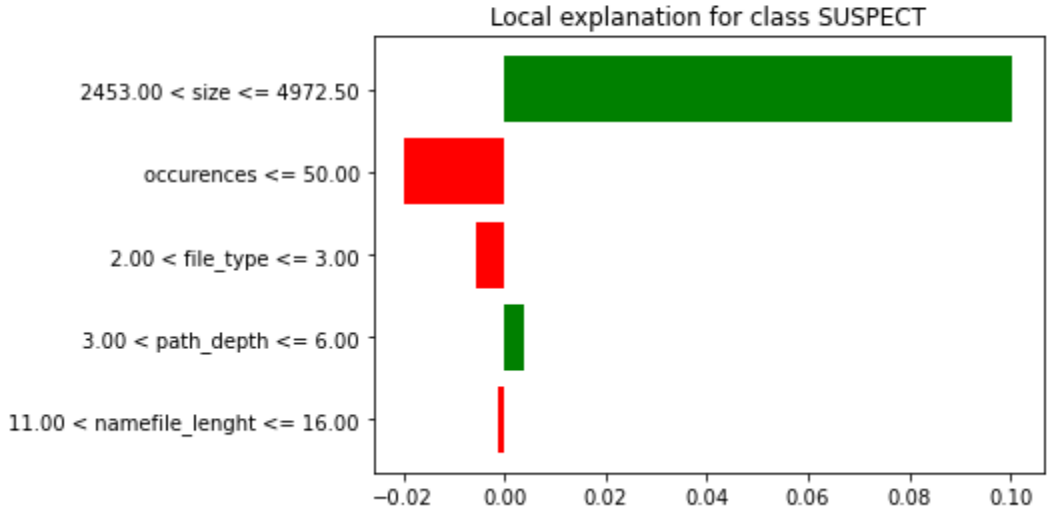
Ecco il file in Python, che implementa un Machine Learning utilizzando il modello di classificatore kNN (K-Nearest Neighbors) della libreria SKLEARN

MACHINE LEARNING AND DIGITAL FORENSICS

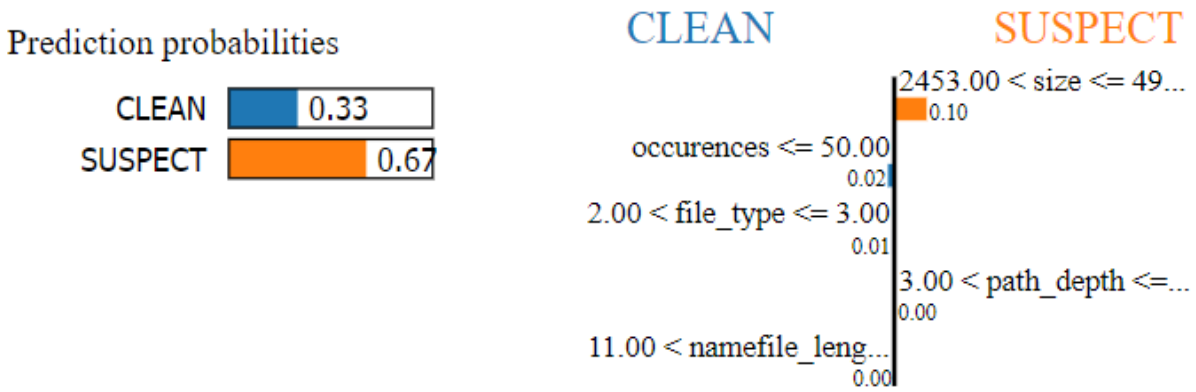
```
from sklearn.model_selection import train_test_split
#from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score
from sklearn import preprocessing
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from interpret.glassbox import ClassificationTree
#from interpret import show
import lime
import lime.lime_tabular
import numpy as np
import warnings
warnings.filterwarnings("ignore")

# Load dataset
url = "metadata.csv"
names = ['file_type','path_depth','size','namefile_lenght','occurences','label']
dataset = pd.read_csv(url, names=names,header=0)
# Split-out validation dataset
dataset= pd.DataFrame(dataset)
array = dataset.values
X = array[:,0:5]
y = array[:,5]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
model = KNeighborsClassifier(n_neighbors=3)
model.fit(X_train, y_train)
pred_train = model.predict(X_train)
pred_test = model.predict(X_test)
acctest=accuracy_score(y_test,pred_test)
acctrain=accuracy_score(y_train,pred_train)
#valori=[1,2,131,13,40]
valori=[3,6,3131,13,40]
prediction = model.predict([valori])
print("Accuracy Train:",round(acctrain*100,2),"% Accuracy Test:
",round(acctest*100,2),"%")
print("Predicted target name: {}".format(prediction)," \n",names," \n",valori,"
",prediction)
#start XAI by LIME
label=['CLEAN','SUSPECT']
X_featurenames = names
predict_fn=lambda x:model.predict_proba(x).astype(float)
explainer=lime.lime_tabular.LimeTabularExplainer(np.array(X_train),mode='classificatio
n',feature_names= X_featurenames,class_names=label)
# asking for explanation for LIME model
exp = explainer.explain_instance(np.asarray(valori), predict_fn, num_features=5)
exp.as_pyplot_figure()
exp.save_to_file('lime.html')
print("Red is for CLEAN (0), Green is for Suspect YES (1)")
```

LIME – A WAY TO UNDERSTAND THE BLACKBOX

```
Accuracy Train: 88.95 % Accuracy Test:  82.67 %
Predicted target name: ['suspect']
 ['file_type', 'path_depth', 'size', 'namefile_lenght', 'occurences', 'label']
 [3, 6, 3131, 13, 40]     ['suspect']
Red is for CLEAN (0), Green is for Suspect YES (1)
```



LIME – A WAY TO UNDERSTAND THE BLACKBOX



Feature	Value
size	3131.00
occurences	40.00
file_type	3.00
path_depth	6.00
namefile_lenght	13.00

MACHINE LEARNING AND DIGITAL FORENSICS

Il programma inserisce le caratteristiche nella variabile X, le etichette nella variabile Y, quindi esegue il training utilizzando il 30% del dataset per i test, infine assume nuovi valori nella variabile "values"
valori = [3,6,3131,13,40]

dove abbiamo file di tipo 3, quindi DOCX, una profondità di directory 6, una dimensione di 3131 Kb, un nome file lungo 13 caratteri e occorre 40 volte.
Il risultato è:
Accuracy Train: 88.95% Accuracy Test: 82.67%
Predicted target name: [‘suspect']
['file_type', 'path_depth', 'size', 'namefile_lenght', 'occurences', 'label']
[3, 6, 131, 13, 40] [‘**suspect**']

CONCLUSIONS AND BIBLIOGRAPHY & SOFTWARE

Quello descritto in questo intervento è un modo per approcciarsi ad un nuovo concetto di riconoscimento dei file malevoli, sicuramente va implementato in modo preciso ed affidabile, anche cambiando il classificatore, magari con altre tipologie si avrebbero risultati più precisi, bilanciare bene il set di dati tra "pulito" e "sospetto" e molto altro ancora.

Lo scopo principale di questo talk è quello di dare spunto al possibile matrimonio tra Digital Forensics e Intelligenza Artificiale e anche di fornire un'idea delle potenzialità e delle criticità di queste due discipline molto affascinanti ma anche molto complesse.

https://www.researchgate.net/publication/334192020_Methodology_for_the_Automated_Metadata-Based_Classification_of_Incriminating_Digital_Forensic_Artefacts
https://github.com/nannib/AI/tree/master/metadata
https://nannibassetti.com/articoli.htm

THANK YOU!

Dr. Nanni Bassetti

https://nannibassetti.com