Detecció i classificació d'atacs en una xarxa informàtica mitjançant machine learning

Jiahao Ye

Resum– Aquest projecte descriu com implementar un sistema d'aprenentatge automàtic per detectar i classificar els atacs d'una xarxa informàtica. Durant el treball, es presenten diferents algoritmes d'aprenentatge automàtic i es compara els resultats que ens dóna per veure quins són els millors models per detectar anomalies. El treball comença explicant el preprocessament de les dades, fins l'entrenament i optimització dels models, com l'arbre de decisions, Gradient Boost Machine, Support Vector Machine i Xarxes neuronals. Els models s'entrenen a partir de les dades de la competició KDDCup 1999.

Paraules clau– Python, Machine learning, Scikit-learn, Xarxes neuronals, Keras, Ciberseguretat.

Abstract– This project describes how to implement a machine learning system to detect and classify attacks on a computer network. During the work, differents machine learning algorithms are presented and are compared the results it gives us to see which are the best models to detect anomalies. The work begins by explaining the preprocessing of the data, up to the training and optimization of the models, such as Decision Tree, Gradient Boost Machine, Support Vector Machine and Neural Networks. The models are trained using data from the KDDCup 1999 competition.

Keywords– Python, Machine learning, Scikit-learn, Neural Networks, Keras, Cybersecurity.

1 Introduction

Les xarxes de telecomunicacions transmeten informació a molt alta velocitat, per assegurar que les xarxes no siguin atacades necessitem sistemes que detectin automàticament els atacs informàtics.

Els sistemes de detecció d'intrusions (IDS) són sistemes que monitoritzen les xarxes informàtiques per detectar possibles activitats maliciosos. Tenim sistemes IDS basats en signatura, que detecten atacs a partir de patrons coneguts del sistema. Però aquest tipus de sistemes no tenen la capacitat de detectar atacs no coneguts, necessiten afegir els patrons manualment. Per això van crear els sistemes IDS basats en anomalies, que l'aprenentatge automàtic per detectar atacs desconeguts. [1]

Per crear un detector d'intrusions eficient, hem construït un sistema a partir de models d'aprenentatge automàtic que es poden entrenar per intentar detectar els possibles atacs a partir de dades de tràfic on presenten connexions normals i connexions amb diferents tipus d'atac.

• E-mail de contacte: jiahao.ye@e-campus.uab.cat
• Menció realitzada: Tecnologies de la Informació
• Treball tutoritzat per: Jordi Casas Roma (DEIC)
• Curs 2019/20

2 Estat de l'art

El *machine learning* és l'àmbit d'estudi que dóna als computadors la capacitat d'aprendre, a partir d'unes dades donades, com resoldre un problema. Aquests models es basen en inferir patrons en les dades que permetin al algoritme detectar comportaments de forma autònoma, és a dir, sense intervenció humana, un cop han estat entrenats.

Podem classificar els sistemes d'aprenentatge automàtic en 4 tipus:

- **Aprenentatge supervisat**: Les dades d'entrenament (*training set*) ha de ser supervisada per un humà i etiquetada amb la solució correcta. Les tasques de classificació i regressió són d'aquest tipus. Algun dels principals algoritmes són el SVM (*Suport Vector Machine*), Arbre de decisions, *Random forest*, *Linear Regression*, *Logistic Regression*, *Neural Network*, entre d'altres.

  En l'aprenentatge supervisat partim les dades d'entrenament en dos conjunts: el conjunt d'entrenament, i

el conjunt de validació. On el conjunt d'entrenament s'utilitza expressament per entrenar el model, i el conjunt de validació per veure si les prediccions que ha fet el model són bones o no. El motiu de partir les dades en dos conjunts és per comprovar si el model està generalitzant correctament davant d'un conjunt de dades noves.

Els models d'aprenentatge supervisat sempre tenen millors resultats quan fan prediccions amb les dades d'entrenament, normalment baixa el percentatge d'encerts quan prediu dades diferents del conjunt d'entrenament. El conjunt de validació serveix per comprovar que el model pot fer bones prediccions en dades "desconegudes".

- **Aprenentatge no supervisat**: El conjunt d'entrenament no està etiquetat. El sistema haurà de trobar la solució per ell mateix. Les tasques típiques d'aquest tipus són el clustering, i reducció de dimensió. Alguns exemples d'aquest tipus són PCA (*Principal Component Analysis*), kMeans, EM (*Expectation Maximization*), entre d'altres.

- **Aprenentatge semisupervisat**: Les dades d'entrenament són una barreja de dades sense etiquetes i amb etiquetes. Els algoritmes d'aprenentatge són una combinació de la supervisada i no supervisada.

- **Aprenentatge per reforç**: El sistema observa l'entorn, fa una acció i rep una recompensa o una penalització. Amb això el sistema decideix quines són les accions correctes i en la propera vegada farà una acció per obtenir la millor recompensa.

3 OBJECTIUS DEL TREBALL

L'objectiu principal d'aquest treball és la creació de sistemes d'aprenentatge automàtic que permetin detectar els atacs d'una xarxa informàtica. L'objectiu principal es pot descomposar en el següent conjunt d'objectius:

1. Analitzar la informació de connexions de la xarxa per poder determinar quins són els atributs més rellevants per a detectar el tipus d'atac.

2. Entendre la informació que ens dóna les trames d'una xarxa informàtica.

3. Construir un sistema capaç de detectar si una connexió en una xarxa informàtica és normal o un atac.

4. Ser capaç de distinguir el tipus d'atac que s'està produint en una xarxa informàtica.

5. Estudiar els diferents mètodes de l'aprenentatge supervisat, així com les seves avantatges i incovenients, per poder ser aplicats en aquest problema.

6. Entrenar, avaluar i comparar la capacitat dels diferents models d'aprenentatge automàtic (*machine learning*) per a detectar atacs a una xarxa informàtica.

4 METODOLOGIA

El projecte es desenvolupa iterativament. La metodologia àgil que s'aplica és el kanban [5]. Aquesta metodologia es basa en escriure les tasques en targetes i posar-les en un tauler de 3 columnes on cada columna representa el estat d'una tasca (pendents, treballant, realitzat). Amb el tauler podem visualitzar el flux de treball sabent les tasques pendents , realitzades i en procés. També podem gestionar les tasques, limitar el màxim de tasques que es pot treballar en el moment, i saber el temps previst per acabar les tasques.

La eina que utilitzarem és Trello [6]. Aquesta aplicació web ens permet crear taulers i gestionar les tasques amb la metodologia de kanban.

5 PLANIFICACIÓ DEL PROJECTE

Els passos del projecte s'ha desenvolupat iterativament, i s'ha dividit en tres fases:

5.1 Preprocessament de les dades

Abans d'entrenar el model, hem de mirar la base de dades que tenim, realitzar certes tasques per garantir que aquestes dades són acurades, fiables i de qualitat. A més, cal tenir en compte la tipologia de les dades i els models que volem utilitzar. Per exemple, hi han models que no poden processar els *strings*, i per tant, els haurem de convertir en números reals. També podem aplicar algunes transformacions en les dades, que ens permeten millorar considerablement els resultat final d'un model. Algunes transformacions que podem aplicar són:

- Convertir les variables categòriques en números enters.
- Tractament dels valor absents, podem eliminar les mostres que hi han valors buits o omplir per el valor mitjà de la columna.
- Escalar les dades en el mateix rang.
- Seleccionar els atributs més representatius.
- Combinar atributs per generar nous coneixements.

5.2 Entrenaments dels models

En aquest treball provarem diferents models d'aprenentage automàtic i realitzarem una comparació dels resultats obtinguts en cadascun d'ells. Els models hem entrenat són:

- Arbre de decisions
- *Gradient boosting*
- Support Vector Machine
- Xarxes neuronals

Hem seleccionat aquests models perquè cadascun es basa en diferents tipus de funcionament, i volem veure quin model és el que dóna millor resultat per aquest treball.
L'arbre de decisió es basa en crear regles automàticament, depenent si compleixen les condicions classifiquen les dades en una classe o altre. És el model més senzill i permet

explicar, de forma sencilla, els resultats obtinguts i les regles creades.
El *Gradient boosting* es basa en la combinació d'arbres, aquí podem veure com es comporta quan combinem diversos arbres.
Els SVM són models basats en la distàncies, representen les instàncies en un espai i classifica com a una classe determinada les instàncies que estiguin en una mateixa "zona".
Les xarxes neuronals són models basats en el funcionament del cervell humà, molt més complexos que els models anteriors i tarden molt més temps en entrenar.

5.3 Avaluació de resultats i optimitzacions

Un cop acabat l'entrenament dels models veurem els resultats que dóna cada model. Per millorar els resultats haurem de repetir iterativament les dues fases anteriors, aplicant diferents transformacions en les dades o provant diferents hiperparàmetres dels models, per obtenir la combinació que ens dóna el millor resultat.

5.4 Estimació de les etapes

La taula 2 indica les diferents etapes del projecte i les dates aproximades de realització de cadascuna de les etapes.

TAULA 1: ESTIMACIÓ DE LES ETAPES

Pasos	Durada	Data prevista
Preparació dades	4 setmanes	fins a 3 nov
Model: Decision Tree	2 setmanes	fins a 17 nov
Model: Gradient Boosting	2 setmanes	fins a 1 dec
Model: SVM	2 setmanes	fins a 15 dec
Model: Neural Networks	2 setmanes	fins a 29 dec
Optimitzacions	3 setmanes	fins a 19 gener
Informe Final	3 setmanes	fins a 10 febrer

6 DESCRIPCIÓ DE LES DADES

La base de dades utilitzada per realitzar el treball, va ser publicada en la tercera competició internacional d'eines de descobriment de coneixement i minería de dades en el any 1999.[4]
El conjunt de dades està formada per varis fitxers:

- **kddcup.names**: llistat amb els atributs de les dades, en total 41 atributs. Bàsicament estan definides per 3 categories d'atributs, atributs bàsics d'una connexió TCP/IP, atributs sobre el contingut de la connexió i atributs sobre el trànsit en una finestra de 2 segons.
- **kddcup.data**: conté totes les dades per l'entrenament del model, cada fila representa una connexió tcp/ip entre una adreça IP d'origen i una altre adreça IP de destinació . L'última columna està etiquetada si aquesta connexió és tipus atac.
- **kddcup.testdata**: dades sense les etiquetes de tipus d'atac, s'utilitzarà en la dase de test per avaluar el rendiment del model.

```
import pandas as pd

train_df=pd.read_csv("kdd-cup-1999-data/kddcup.data")
train_df.head()
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment
0	0	tcp	http	SF	181	5450	0	0
1	0	tcp	http	SF	239	486	0	0
2	0	tcp	http	SF	235	1337	0	0
3	0	tcp	http	SF	219	1337	0	0
4	0	tcp	http	SF	217	2032	0	0

Fig. 1: DataFrame

- **corrected**: conté el valor de les classes del fitxer *kddcup.testdata*.
- **training_attack_type**: llistat amb els tipus d'atac. Els atacs poden ser de 4 categories principals: atacs de denegació de servei (DOS), atacs a una màquina remota sense accés autoritzat (R2L), atacs per aconseguir privilegis *root* (U2R) i atacs d'escaneig (*probing*).

7 DESENVOLUPAMENT DEL PROJECTE

En aquesta secció detallarem els passos aplicats i resultats obtinguts en cadascuna de les etapes del projecte.

7.1 Lectura de dades

El primer pas consisteix en llegir les dades dels fitxers. Per això, emplearem el llenguatge Python, concretament la llibreria *pandas* que ens proporciona funcions per facilitar aquest procés. `pd.read_csv()` ens guarda totes les dades del fitxer en un objecte *DataFrame*.
El objecte *Dataframe* és una taula de 2 dimensions, tenim 42 columnes que representen el llistat dels atributs de les connexions, on cada fila representa una connexió, tal i com podem veure en la Fig.1. El fitxer `kddcup.data` conté 4.898.431 mostres diferents, mentre que el fitxer `kddcup.corrected` conté 311.029 mostres que farem servir en la fase de test dels models.

7.2 Preprocessament de dades

7.2.1 Conversió de variables categòriques

Abans d'entrenar el model hem de mirar els tipus de les dades. Com que els models de SVM i xarxes neuronals no permeten treballar amb cadenes de caràcters, haurem de transformar aquests atributs a un tipus numèric (real) per poder entrenar aquests models.
En el nostre *dataFrame* trobem 3 columnes de cadenes de caràcters (`protocol_type, service, flag`). Per transformar, podem aplicar 2 mètodes diferents:

- ***OneHotEncoding***: a partir de la variable categòrica crea un vector de n columnes, on cada columna representa una categoria, i asigna un 1 en la columna que representa la categoria de la mostra corresponent, i en totes les altres columnes un 0.
  El problema que té aquest mètode es que s'ha de crear una columna que cada valor categòric diferent, això

TAULA 2: LLISTAT AMB ELS PRINCIPALS ATRIBUTS[4]

Nom atribut	Descripció	Tipus
duration	duració de la connexió en segons	numèric
protocol_type	tipus de la connexió (tcp,udp,etc)	discret
src_bytes	número de bytes desde el origen fins al destinació	numèric
dst_bytes	desde el destinació fins al origen	numèric
num_root	nombre dels accessos a *root*	numèric
num_file_creations	nombre de fitxers creats	numèric
num_access_file	nombre d'accessos a fitxers	numèric
num_failed_logins	nombre d'intents de sessió fallits	numèric
count	nombre de connexions al mateix host que la connexió actualen un interval de 2 segons	numèric

implica crear tants nous atributs com valors únics hi ha en el conjunt de dades. En conseqüència, aquest mètode consumeix molta memòria quan hi ha molts valors únics, i per tant la fase d'entrenament serà més lent per els models.

- ***BinaryEncoding***: Aquest mètode seria una alternativa al *One-Hot*. Consisteix en representar les cadenes de caràcters en seqüencies binàries de 0,1. En aquest cas, amb n columnes podem representar n^2 valors diferents, produint un estalvi de memòria important respecte el mètode *OneHotEncoding*.

En el nostre cas hem emprat les funcions de la llibreria *sklearn*[1] per fer aquestes operacions.

7.2.2 Generació d'atributs

En aquest pas intentem crear nous atributs a partir dels que ja tenim. La idea és que els nous atributs aportin nova informació a partir dels atributs simples originals, repercutint en la millora dels resultats de la predicció dels models.
Atributs nous:

- `num_actions`: és la suma de `num_root`, `num_file_creations`, `num_shells`, `num_access_files`

7.2.3 Feature Scaling

Per els models basats en la distància millora el rendiment quan les dades estan a una mateixa escala [8]. Per tant, en els models de SVM i NN pot millorar bastant, mestre que en els model d'arbres no té una millora significativa. Per escalar tenim dues tècniques:

- **Estandarització**: es resta per la mitjana i divideix per la desviació estàndard:

$$x'_i = \frac{x_i - \overline{X}}{\sigma}$$

X és un conjunt de dades, $X = \{x_1, x_2, ..., x_n\}$
Cada x_i representan un valor del conjunt X, i el x'_i és el nou valor de x_i després d'aplicar la normalització o estandarització.

- **Normalització**: es resta per el valor mínim de la columna i divideix per la diferència entre el valor màxim i mínim.

$$x'_i = \frac{x_i - min(X)}{max(X) - min(X)}$$

El $min(X)$ retorna el valor mínim del conjunt X, el $max(X)$ retorn el valor màxim, $\overline{X}$ és la mitjana de conjunt X i el σ és la desviació estàndard.

7.2.4 Selecció dels atributs

Per saber quins són els atributs rellevants podem utilitzar la classe `SelectKBest`[2] de la llibreria `sklearn`. Aquesta classe proporciona una funció per puntuar els atributs,i amb el mètode `fit()` podem calcular la puntuació introduint el conjunt d'entrenament. Un cop calculat ens quedem amb els atributs que tenen major puntuació.

Les funciones de puntuació que utilitzarem són el següent:

- ANOVA : agrupa les mostres en diferents classes d'atacs, en les mostres de cada clase es calcula la mitjana, i finalment dóna un test estadístic [10].
- χ^2: test per veure les variables categòriques si són independents o no [12].

L'atribut `scores_` de la classe `SelectKBest` ens permet mostrar la puntuació que dóna a cada atribut. Amb aquesta informació podem decidir nosaltres mateixos quins atributs descartar, com podem veure en la Fig. 2, la puntuació de cada atribut.

7.2.5 Atributs descartats

En el conjunt de dades hem trobat que hi han atributs que no aporten informació, tenen el mateix valor en totes les mostres. Aquests atributs són el següent:

[1] https://scikit-learn.org/

[2] bit.ly/2RswJzR

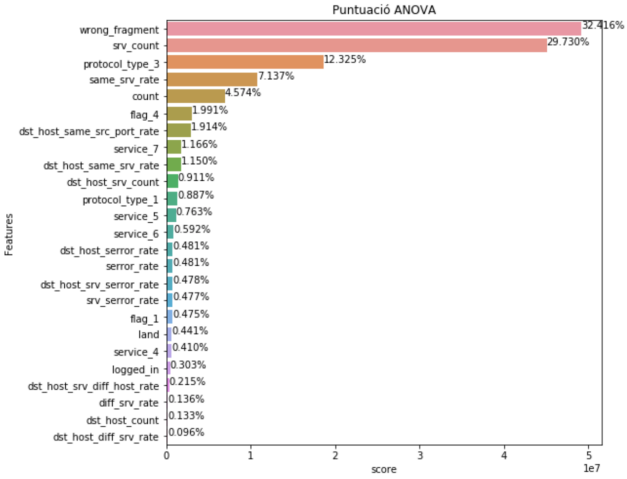

Fig. 2: Top 25, puntuació ANOVA

- `num_outbound_cmds`
- `is_host_login`
- `service_0`
- `flag_0`

Per tant, podem treure aquests atributs per estalviar memòria.
A partir dels tests de l'apartat anterior podem determinar la rellevància dels atributs segons la puntuació. Hem decidit treure alguns atributs que han obtingut una baixa puntuació en els 2 tests, que són els següents:

- `num_root`
- `service_1`
- `su_attempted`
- `urgent`
- `num_acces_file`
- `num_shells`
- `num_file_creations`
- `root_shell`

7.3 Entrenament del model

Durant el procés d'entrenament cal indicar quin és el valor objectiu (*target*) de cada una de les connexions, per tal que el model sigui capaç d'aprendre a classificar les dades.

En les dades originals trobem fins a un total de 23 tipus d'atac, que hem condensant en un total de 4 categories: U2R, DOS, R2L, probe. Addicionalment, també considerem la categoria de connexió legítima, és la dir, quan no hi ha atac associat.

Com ja hem comentat utilitzarem els 4 models següents:

- Arbre de decisió
- Gradient Boosting
- Support Vector Machine
- Xarxa neuronal

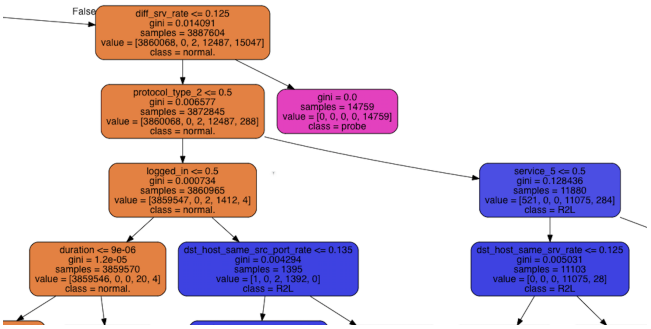

Fig. 3: Exemple de nodes de l'arbre de decisió, on es pot veure cada node conté una condició que deriva en dos nodes fills, segons si compleix la condició.

7.3.1 Arbre de decisió

Aquest model es basa en representar les decisions en una estructura d'arbre. La idea és simple, cada node representa una pregunta o condició; si compleix es classifica la instància cap a un dels nodes fills, i en cas contrari, cap a l'altre node fill. En general, els arbres són binaris, és a dir, la condició presenta només dues opcions de resposta (p.ex. l'atribut `diff_srv_rate` és menor o igual que 0.125). Cada mostra passa per tots els nodes de l'arbre fins arribar a un node terminal (anomenat fulla) que decideix quina classe pertany, com podem veure Fig.3.

Les preguntes o condicions que fan han de ser rellevants per la decisió final de la classe. Per això utilitzem la funció de mesura `gini impurity`.

Per entrenar l'arbre de decisió hem utilitzat el model de la llibreria *sklearn*, `DecisionTreeClassifier()` amb els paràmetres per defecte de l'arbre.

Tots els models de la llibreria `sklearn`, venen preparats amb el mètodes `fit()`, que entrena les dades amb el corresponent *label* que hem introduït com a paràmetre, i un cop acabat l'entrenament, amb el mètode `predict()` fan les prediccions de les dades i ens retorna les classes predites.

7.3.2 Gradient Boost

Aquest model es basa en la combinació de classificadors (*Ensemble learning*) que consisteix en entrenar un grup de classificadors i que cada classificador doni la seva predicció. Llavors per cada classificador obtindrem un resultat de la predicció, i emprant algun tipus de mecanisme de votació, podem determinar la classe predita de forma global per la instància. Normalment, entrenar un grup de classificadors pot obtenir millors resultats que un classificador individual. Existeixen diferents mètodes dins d'aquest grup, entre els quals hi ha el *Gradient boosting*.

El *gradient boosting* entrena un grup d'arbres de decisió seqüencialment, el primer arbre s'entrena a partir del el conjunt de dades d'entrenament, el segon arbre s'entrena amb els errors residuals del primer arbre, el tercer arbre s'entrena amb els errors residuals del segon arbre, i així successivament. L'objectiu és que cada arbre s'apren dels errors comesos de l'arbre anterior, i finalment tenim un conjunt d'arbres que intenten donar prediccions amb el mínim error possible.

Durant el procés d'entrenament, hem provat diferents configuracions de hiperparàmetres per tal d'ajustar i reduir

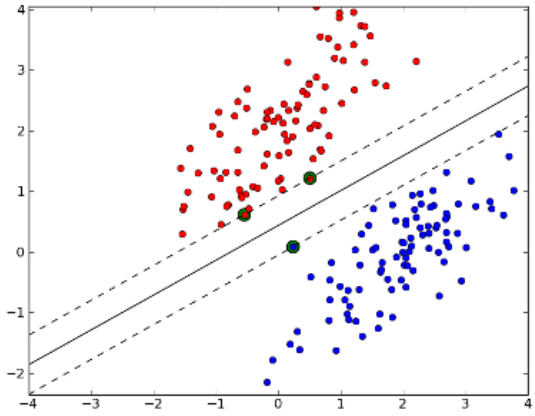

Fig. 4: Exemple SVM lineal [14]

els errors comesos. Entre els paràmetres disponibles podem trobar:

- el `n_estimators` per decidir quants arbres volem entrenar,
- el `learning rate` si volem que els arbres s'aprengui més ràpid o més lent.

El millor resultat que ens ha donat es quan el *learning rate*=1 i `n_estimators`=100.

En aquest treball, per entrenar el model *Gradient boosting* hem utilitzat el `lightGBM`[3], que és un *framework* de *gradient boost* desenvolupat per Microsoft. L'avantatge d'aquest model és que fa els càlculs molt més ràpid i ocupa menys memòria que el `XGBoost`, tot i que els resultats no tenen perquè ser millors.

7.3.3 Support Vector Machine

Aquest tipus de model consisteix en trobar un hiperplà de n dimensions que permeti una separació òptima entre el conjunt de dades, on n nombre de paràmetres d'entrada (en el nostre cas, n és 47). El hiperplà ha de separar les dades en dues classes diferents, com es pot veure en l'exemple de la Fig.4.

En el procés d'aprenentatge, el model troba el millor hiperplà per classificar les dades d'entrenament maximitzant la distància (marge) entre les dades i el hiperplà.

Com que el SVM es un classificador binari cada hiperplà només pot diferenciar entre dues classes. Per fer que el model pugui classificar múltiples classes necessitem combinar varis classificadors SVM i que cada classificador aprengui a classificar una classe determinada. El model utilitzat és el `sklearn.svm.SVC()` de la llibreria *sklearn*. Aquest model és un classificador multiclasse de tipus *one vs rest classifier*, és a dir, crea un classificador per detectar una classe, per exemple si és normal/altres, DOS/altres, etc. En total es crea n classificadors on *n* és el nombre de classes.

[3]https://lightgbm.readthedocs.io/en/latest/

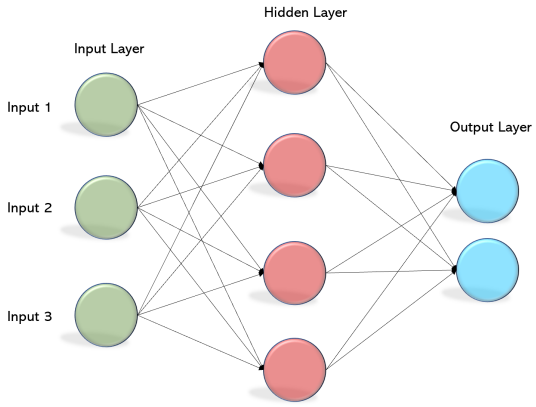

Fig. 5: Exemple *Multi Layer Perceptron* [13]

7.3.4 Xarxa neuronal

Una xarxa neuronal és un sistema de neurones interconnectat per diferents nivells de capes, com mostra la figura 5. Generalment, una xarxa neuronal té una capa d'entrada, una capa de sortida i una o més capes ocultes. El número de neurones en la capa d'entrada correspon amb el número de paràmetres d'entrada, que en el nostre cas són 47 neurones d'entrada, i el número de neurones en la capa de sortida correspon a les classes que volem classificar, en aquest cas 5 neurones que correspon a les 5 classes: normal, DOS, probe, U2R, R2L.

Cada neurona rep els valors de les neurones de la capa prèvia, suma els valors amb els seus pesos corresponents i calcula el valor de sortida mitjançant una funció d'activació, com es pot veure en la figura 6. Cada una de les instàncies del conjunt d'entrenament s'introdueix a la xarxa a través de la capa d'entrada. La xarxa aplica els càlculs per a cada neurona i cada capa fins a obtenir els valors de la capa de sortida, que indiquen la classe de la instància. Si el valor de classe predit no és correcta, llavors es calcula l'error comès i es propoga cap a les capes internes per tal que modifiquin els pesos i s'ajustin per millorar la predicció. Aquest procés es basa en el mètode de la retropropagació (*Backpropagation*) [11].

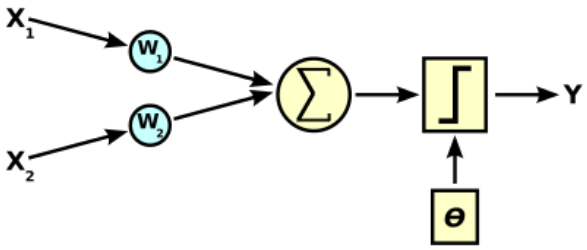

Fig. 6: Exemple *neurona* [16]

Resumint i simplificant podem dir que el procés d'aprenentatge consisteix en actualitzar els pesos w_i de les connexions entre tots els parells de neurones en capes adjacents, fins que els valors de la capa de sortida minimitza l'error. Per fer això necessitem un algoritme d'optimització i una funció de cost.

- **Funció de cost**: serveix per mesurar l'error que hi ha

entre el valor predit i el valor real. Com més baix sigui l'error, obtindrà millors resultats. Existeixen moltes funcions per calcular el cost, i cada funció s'utilitza en un tipus de problema concret. Per exemple en un problema de regressió normalment s'utilitza la funció MSE (Error mitjà quadràtic):

$$MSE = \frac{1}{n}\sum_{i=0}^{n}(\widehat{Y_i} - Y_i)^2 \quad (1)$$

on la variable n és el nombre total de dades, Y_i valor real, $\widehat{Y_i}$ valor predit.

En el nostre cas, com que es tracta d'un problema de classificació de 5 classes, la funció de cost que tenim es *categorical_crossentropy*:

$$Loss(y, \widehat{y}) = -\sum_{i=1}^{N}\sum_{j=1}^{C}(y_{ij} * log(\widehat{y}_{ij})) \quad (2)$$

on n representa el nombre de dades, C el nombre de classes, i els resultats estan representades en vectors *onehot* de C dimensions. És a dir, el vector y_i contindrà un 1 en la posició de la classe correcte i 0 en les altres posicions. Els valors del vector $\widehat{y_i}$ són representades per probabilitats, valors entre 0 i 1. Com més s'aproxima els valors del vector $\widehat{y_i}$ al y_i, més baix serà el cost i l'error produit.

- **Algoritme d'optimització**: l'objectiu d'aquests algoritmes es minimitzar el cost, actualitzant els pesos w pas a pas. Sempre es defineix un hiperparàmetre (*learning rate*) que és el ritme d'aprenentatge i el grau de modificació dels pesos (w).

Per crear xarxes neuronals hem escollit la llibreria *keras*. Aquesta llibreria ens proporciona funcions per construir i entrenar el model fàcilment.

Durant el treball, hem provat varies combinacions de hiperparàmetres: el nombre de neurones, el nombre de capes ocultes, la funció d'activació. Finalment, l'arquitectura que ens ha donat millors resultats és la següent:

- Capa entrada: 47 neurones
- Capa oculta 1: 128 neurones
- Capa oculta 2: 64 neurones
- Capa sortida: 5 neurones

La funció d'activació utilitzada per les capes d'entrada i ocultes és el ReLU. Aquesta funció simplement canvia els valors negatius a 0, la formula és:

$$f(x) = max(0, x)$$

En la capa de sortida hem utilitzat la funció Softmax. Aquesta funció converteix el vector de la capa de sortida en un vector de probabilitats, com podem veure en la Fig.7. Com que la funció de cost és la *categorical crossentropy* hem de utilitzar sempre el *softmax* en la sortida per poder tenir un vector en el rang 0 a 1.

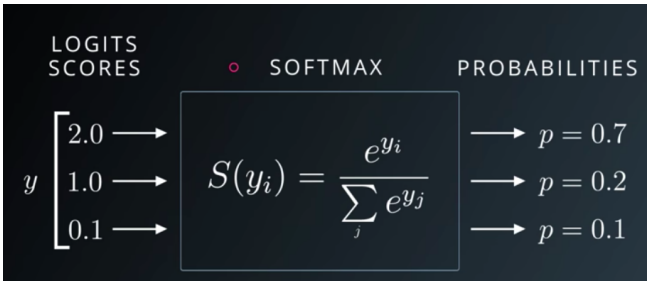

Fig. 7: Funció d'activació *Softmax* [15]

7.4 Mètriques d'avaluació

Un dels principals problemes de l'aprenentatge supervisat és el sobreentrenament (*overfitting*) dels models. El sobreentrenamet es produeix quan el model s'ajusta massa a les dades d'entrenament i no és capaç de generalitzar correctament davant de noves dades, només poden classificar correctament les dades que són idèntiques a les dades d'entrenament. En l'aprenentatge supervisat és molt important la generalització del model, ja que si està ben generalitzat ens permet fer prediccions fiables en dades desconegudes.

Per saber si el nostre model pateix el problema de sobreentrenament, hem separat en dos el conjunts de dades original: una part en dades d'entrenament i una altre part en dades de validació. El model s'entrena amb les dades d'entrenament, llavors fem les prediccions utilitzant les dades de validació, i es compara els resultats predits i els resultats reals del conjunt de validació. Així podem determinar si els resultats predits concideixen amb els reals i quants errors ha comès el model en les seves prediccions.

Generalment, s'avalua amb les dades de validació, ja que aquest model ha estat entrenat amb les dades d'entrenament, i cal verificar el funcionament del model amb dades noves, que no hagin estat "vistes"pel model anteriorment.

Hem de mirar que passa quan el model fa prediccions amb dades desconegudes, si segueix tenint bons resultats vol dir que el model generalitza bé. En canvi, si presenten resultats sensiblement inferiors amb el conjunt de validació, vol dir que el model només pot fer bones prediccions amb les dades d'entrenament, i per tant, estem en un cas de sobreentrenament (*overfitting*).

TAULA 3: MATRIU CONFUSIÓ

Predit / Real	Positiu	Negatiu
Positiu	True Positive	False Negative
Negatiu	False Positive	True Negative

Matriu de confusió

En les tasques de classificació podem crear una matriu de confusió. Aquest matriu ens permet visualitzar de forma intuïtiva com estan classificades les dades de cada classe, incloent el nombre d'encerts i els erros comesos (indicant el valor correcta i el valor predit).

La matriu de confusió presenta, en la columna representa les prediccions, i en les files, els resultats reals (és a dir,

TAULA 4: INFORMACIÓ DELS MODELS

Model	**Paràmetres**	**Temps execució**	**Accuracy**	
			kddcup.data	corrected
Arbre decisió	`criterion = 'gini_impurity'`	44s	99,99%	73,39%
Gradient boosting	`learning_rate = 1` `n_estimators = 100`	7min 19s	99,92%	90,70%

SVM	`kernel = 'rbf'`
	`penalty = 'l2'`
	`C = 1.0`
		7min 56s	99,85%	91,92%

Xarxa neuronal	`epochs = 5`
	`batch_size = 64`
	`optimizer = 'rmsprop'`
	`loss = 'categorical_crossentropy'`
		50min	99,96%	90,80%

les etiquetes correctes del conjunt de dades). Es pot veure un exemple de classificació binari en la taula 3. Només es considera encertades les dades que estan en la diagonal de la matriu, és a dir, en la posició *True Positive, True Negative*. Les dades en la posició de *False Positive* són les dades que el model ha classificat com a positiu però en realitat és negatiu, i els *False Negative* són el contrari.

Per avaluar la qualitat dels resultats predits, hem aplicat 3 mètriques:

- ***accuracy***: mira només el percentatge de dades encertades entre el conjunt total de dades de validació.

$$accuracy = \frac{encertades}{total}$$

- ***recall***: per cada classe, mira el percentatge de dades encertades d'aquesta classe entre el total de dades que s'han classificat com aquesta classe.

$$recall = \frac{TP}{TP + FN}$$

- ***precisió***: per cada classe, mira el percentatge de dades encertades d'aquesta classe entre el total de dades reals d'aquesta classe.

$$precision = \frac{TP}{TP + FP}$$

TP: True Positive
TN: True Negative
FN: False Negative
FP: False Positive

Depèn del context de cada problema ens importa més el *recall* o la precisió. En el nostre cas, volem que el detector tingui un recall alt (no deixa escapar posibles atacs). És difícil mantenir el recall i precisió en un mateix nivell, normalment quan augmentes el recall la precisió baixa i viceversa, per això hem de decidir en cada problema a tractar quina mesura és més important, el recall o precisió.

8 RESULTATS

Un cop finalitzat l'entrenament dels models, utilitzem la funció `predict()` de cada model per a predir la categoria d'atac amb les dades del conjunt de validació. És a dir, les dades del fitxer `kddcup.corrected`. A continuació obtenim la classe prevista de les dades i comparem els resultats que dóna cada model amb els resultats reals.

Abans de crear la matriu de confusió hem comparat els resultats de les prediccions fetes amb el conjunt d'entrenament i el conjunt de validació, que podem veure en la taula 4. En aquest cas, la mètrica d'avaluació aplicada és l'*accuracy*.

La columna *Accuracy* mostra la diferència de les puntuacions obtingudes per cada model en els conjunts de dades dels fitxers `kddcup.data` (utilitzat per l'entrenament del model) i `kddcup.corrected` (utilitzat per la fase de test o validació). Podem veure que les puntuacions amb les dades del fitxer `kddcup.data` són molt altes, ja que són les dades que hem utilitzat per entrenar el model. Tots els models aconsegueixen aproximadament 99,90% d'*accuracy*.

En canvi, amb les dades del fitxer `kddcup.corrected`, l'*accuracy* baixa considerablement. Això es degut a que les dades són diferents que les dades d'entrenament, els models no han "vist"mai aquestes dades. Quan no hi ha problema de sobreentrament, les dades d'*accuracy* en l'entrenament i el test haurien de ser similars, però quan les dades en el test són considerablement inferiors, podem concloure que els models han patit un problema de sobreentrenament.

Tal i com podem veure en la taula 4, el model que ha aconseguit el major rendiment és el *Support Vector Machine*, i el que obté pitjor resultat és l'arbre de decisió.

En la taula 4, podem veure els paràmetres dels models que s'ha utilitzat en el procés d'entrenament, i també els temps d'execució. Podem veure que el model més ràpid d'entrenar és l'arbre de decisió i el model de xarxa neuronal és el que tarda més.

Per cada model, mostrarem la matriu de confusió per conèixer millor els resultats, on podem veure els detalls de les instàncies correctes i incorrectament classificades per a cada model.

8.1 Arbre de decisió

En la taula 5, podem veure la precisió i *recall* de cada classe. Les classes 'normal' i 'DOS' són les que estan ben classificades i les altres classes no obtenen bones puntuacions. Hi ha una part de mostres de 'DOS' que ho classifica com a 'probe' provocant que baixi molt el *recall* de 'probe'. La classe 'U2R' presenta els pitjors resultats, el model no ha predit correctament cap instància. La classe 'R2L' té un *recall* molt alt però la precisió és baixa, perque la majoria de 'R2L' han estat classificades com a 'normal'. En resum, podem conclure que el model només classifica bé les classes 'normal' i 'DOS'.

TAULA 5: MATRIU DE CONFUSIÓ D'ARBRE DE DECISIÓ

Real \ Predit	normal	R2L	DOS	probe	U2R	pr.[1]
normal	58.580	10	1.724	277	2	96,67%
R2L	15.321	700	5	316	3	4,28%
DOS	8.076	0	164.675	57.104	0	71,64%
probe	680	4	15	3.467	0	83,22%
U2R	64	6	0	0	0	0,00%
recall	70,81%	97,22%	98,95%	5,66%	0,00%	51,16% / 54,53%

[1] precisió

8.2 Gradient boosting

Comparant amb el model anterior, el *gradient boost machine* ha millorat l'*accuracy* i el *recall*, però ha baixat la precisió.

Observant la taula 6, podem veure que les classes 'R2L' i 'U2R' obtenen resultats molt baixos, segueix el problema del model anterior. En canvi, la classe 'probe' ha augmenta el *recall* i ha baixat la precisió, just el contrari que hi ha en el model anterior. En aquest cas, com que ja no classifica els 'DOS' com a 'probe' el *recall* és bastant alt, però el problema és que el model tendeix a classificar les instàncies de 'probe' com a 'normal' o 'DOS'.

Les classes 'normal' i 'DOS' són les fàcils de classificar, i els altres 3 classes fallen freqüentment.

TAULA 6: MATRIU DE CONFUSIÓ DE *gradient boosting*

Real \ Predit	normal	R2L	DOS	probe	U2R	pr.
normal	58.638	4	1.874	75	2	96,77%
R2L	16.268	64	6	9	0	0,37%
DOS	7.000	0	222.850	5	0	96,95%
probe	1.317	0	2.292	528	29	12,67%
U2R	64	0	0	6	0	0%
recall	70,40%	93,93%	98,16%	84,75%	0,00%	41,35% / 69,45%

8.3 Support vector machine

La taula 7 mostra la matriu de confusió del model SVM, on podem veure que la classificació de les classes 'normal', 'DOS' i 'probe' mantenen una alta precisió i *recall*. Però les classes 'U2R' i 'R2L' s'obtenen uns resultats pitjors que els dos models anteriors. En concret, no hi ha cap instància que s'hagui classificat com a 'U2R', per tant, intuïm que el model no es capaç de detectar cap atac de la classe 'U2R'. En el cas de la classe 'R2L', només detecta algunes instàncies, però la gran majoria ho classifica com a 'normal'.

Podem dir que aquest model pot detectar bé les classes 'normal', 'DOS' i 'probe', però no les classes 'R2L' i 'U2R'.

TAULA 7: MATRIU DE CONFUSIÓ DE SUPPORT VECTOR MACHINE

Real \ Predit	normal	R2L	DOS	probe	U2R	pr.
normal	60.288	7	81	217	0	99,50%
R2L	16.118	8	118	101	0	0,05%
DOS	7.554	0	222.262	39	0	96,70%
probe	368	4	432	3.362	0	80,70%
U2R	67	2	0	1	0	0,00%
recall	71,43%	38,10%	99,71%	90,37%	0,00%	55,38% / 59,92%

8.4 Xarxa neuronal

En la taula 8, veiem que la nostre xarxa neuronal no ha donat cap predicció en les classes 'U2R' i 'R2L'. Per tant, podem dir que en aquest cas passa el mateix que en el classificador SVM amb la classe 'U2R', el model no és capaç de detectar cap atac de 'U2R' i 'R2L'. Però igualment ha aconseguit bones resultats generals.

TAULA 8: MATRIU DE CONFUSIÓ DE XARXA NEURONAL

Real \ Predit	normal	R2L	DOS	probe	U2R	pr.
normal	59.613	0	780	200	0	98,38%
R2L	16.340	0	1	4	0	0,00%
DOS	83.780	0	219.795	1.690	0	95,62%
probe	1.117	0	12	3.037	0	75,89%
U2R	70	0	0	0	0	0,00%
recall	69,71%	0,00%	99,64%	61,58%	0,00%	53,38% / 46,18%

En els resultats anteriors, veiem que tots 4 models han tingut dificultats en classificar les classes 'U2R' i 'R2L'. En la xarxa neuronal directament no n'ha detectat cap. En la classe 'R2L' la majoria s'han classificat com a 'normal', i en la classe 'U2R', com que hi ha tan poques mostres d'a-

questa classe, sembla que els models no han tingut suficient mostres d'entrenament i no poden detectar la classe.

Encara que han fallat en aquestes dues classes, els models han pogut classificar raonablement bé les classes 'normal' i 'DOS'. Com que la classe 'normal' i 'DOS' ocupa més del 80% del conjunt de dades, només que classifiquin correctement les dades d'aquestes dues classes hem pogut obtenir bones resultats, en general.

9 Conclusions

Per concloure, hem pogut assolir correctament el objectiu principal d'aquest treball, entrenar models d'aprenentatge automàtic per la detecció i classificació dels atacs en les xarxes informàtiques. Els resultats final mostren que els models han pogut per prediccions amb una precisió raonable en el conjunt de validació (més de 70%). No obstant, cap dels models han donat bones resultats en les prediccions de les classes 'R2L' i 'U2R' perque hi havia poques mostres en el conjunt d'entrenament, els models no han entrenat suficient per classificar correctament aquestes dues classes.

Durant el treball he pogut d'aprendre tots els passos a seguir d'un projecte de *machine learning*. Des de la fase de preprocessament que intentem comprendre la informació donade en les dades, trobar els atributs més rellevants, aplica diferents transformacions en les dades per facilitar l'entrenament del model. I de la fase d'entrenament i optimització, s'aplica tècniques per trobar els paràmetres que ens donen millor resultat, també he aprés com crear un model de xarxa neuronal amb la llibreria de `keras`.

Agraïments

En primer lloc vull agrair el meu tutor Jordi Casas Roma, m'ha ajudat molt durant tot el treball, guiar-me pas a pas, explicant els passos per desenvolupar el projecte en els reunions, i millorar els informes.

També vull agrair a l'associació SIGKDD per publicar aquesta base de dades, ens ha permès treballar amb un conjunt de dades fiables.

Referències

[1] Ïntrusion Detection System (IDS) "geeksforgeeks [Online] Disponible en https://www.geeksforgeeks.org/intrusion-detection-system-ids/ [data d'accés: 26 gener 2020]

[2] Géron, Aurélien. Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems. Sebastopol, CA : O'Reilly Media, Inc., 2017.

[3] Pawel Cichosz, Data Mining Algorithms: Explained Using R, Wiley.

[4] "KDD Cup 1999 DataÜCI KDD-99 Data [Online] Disponible en http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html [data d'accés: 30 de novembre 2019]

[5] "Que es Kanban"Kanbanize [Online] Disponible en https://kanbanize.com/es/recursos-de-kanban/primeros-pasos/que-es-kanban/ [data d'accés: 5 de octubre 2019]

[6] "Trello"Trello [Online] Disponible en https://trello.com [data d'accés: 5 de octubre 2019]

[7] Çategory Encoders"Scikit-learn [Online] Disponible en https://contrib.scikit-learn.org/categorical-encoding/index.html [data d'accés: 22 d'octubre 2019]

[8] S.Geller (4 d'abril) "Normalization vs Standardization - Quantitative Analysis"Towards DataScience [Online] Disponible en https://towardsdatascience.com/normalization-vs-standardization-quantitative-analysis-a91e8a79cebf

[9] "Feature Selection"Scikit-learn [Online] Disponible en https://scikit-learn.org/stable/modules/classes.html#module-sklearn.feature_selection [data d'accés: 6 de novembre 2019]

[10] "Analysis of variace"Wikipedia [Online] Disponible en https://en.wikipedia.org/wiki/Analysis_of_variance

[11] "Neural networks and back-propagation explained in a simple way"Medium [Online] Disponible en https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e [data d'accés: 3 de febrer 2020]

[12] Çhi-Squared test"Wikipedia [Online] Disponible en https://en.wikipedia.org/wiki/Chi-squared_test

[13] "Multi layer Perceptron (MLP) Models on Real World Banking Data"BecomingHuman.ai [Online] Disponible en https://becominghuman.ai/multi-layer-perceptron-mlp-models-on-real-world-banking-data-f6dd3d7e998f [data d'accés: 24 gener 2020]

[14] "why is SVM a "black box" learning algorithm?"StackOverflow [Online] Disponible en https://stackoverflow.com/questions/48133830/why-is-svm-a-black-box-learning-algorithm [data d'accés: 24 gener 2020]

[15] Ünderstand the Softmax Function in Minutes"Medium [Online] Disponible en https://medium.com/data-science-bootcamp/understand-the-softmax-function-in-minutes-f3a59641e86d [data d'accés: 24 gener 2020]

[16] "Xarxa neuronal artificial "Wikiwand [Online] Disponible en https://www.wikiwand.com/ca/Xarxa_neuronal_artificial [data d'accés: 24 gener 2020]