UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

Single e Cross-layer Detection di Siti Web Malevoli: Un Confronto Empirico

RELATORE

Prof. **Fabio Palomba**

Università degli studi di Salerno

CANDIDATO

Nicolapio Gagliarde

Matricola: 0512106980

Anno Accademico 2021-2022

A chi indossa il cappello del colore giusto.

Sommario

Oggigiorno chiunque navighi sul Web può ritrovarsi in siti web che cercano di estrapolare dati sensibili, trarre in inganno l'utente o infettare il suo PC. Per evitare queste situazioni sono state ideate varie tecniche che, grazie all'utilizzo del machine learning, sono in grado di riconoscere i siti malevoli. Quindi lo scopo di questa tesi è quello di analizzare alcune di queste tecniche che, ad esempio, si basano sull'analisi dell'URL, del DOM e del codice JavaScript. Successivamente si discute delle tecniche single e cross-layer, la prima si basa sull'usare solo i dati del livello rete o applicazione. La seconda, attraverso vari tipi di aggregazione, combina i dati dei due livelli con i dati ricavati dalle richieste DNS e dal servizio Whois. Nello specifico si discute dell'implementazione di Xu [2014] e dell'implementazione realizzata utilizzando un dataset diverso per effettuare un confronto dei risultati. Dall'analisi di quest'ultimi risulta che alcuni algoritmi sono in grado di ottenere performance ottime, altri invece presentano falsi positivi e falsi negativi alti, soprattutto riguardo l'implementazione di confronto. Questo è dato sicuramente dal dataset utilizzato, e siccome durante il lavoro svolto si è notata una carenza di dataset progettati per il single e cross-layer, come obiettivo futuro si desidera realizzare un sistema per la costruzione di tali dataset, per poi ripetere il lavoro utilizzando altri algoritmi e altre tecniche.

Indice

Elenco delle figure

Elenco delle tabelle

CAPITOLO 1

Introduzione

1.1 Contesto applicativo

Ad oggi, secondo molte società come Pingdom e Netcraft, i siti web presenti in rete superano gli 1,8 miliardi, di cui solo il 25% risultano attivi. Come ben sappiamo la tipologie di siti web sono moltissime, basti pensare ai siti di e-commerce, forum, blog, siti aziendali etc. Purtroppo però non tutti i siti hanno scopi benevoli, infatti, come riportato nel Kaspersky Security Bulletin 2020[1], oltre 170 milioni di URL sono stati riconosciuti come "siti malevoli" dal software "Web Anti-Virus" della nota azienda. Inoltre lo stesso software ha bloccato oltre 650 milioni di attacchi lanciati da siti web e risorse online.

Questi numeri mostrano quanto importante sia proteggere gli utenti che quotidianamente usano i servizi del World Wide Web.

In termini economici le cifre sono ben più alte, infatti, secondo le società RiskIQ e IBM, le aziende di tutto il mondo perdono circa 1,7 milioni di dollari al minuto a causa dei crimini informatici[2]. Le stesse società hanno riscontrato che l'utilizzo dell'intelligenza artificiale porta a una riduzione delle perdite, questo perché, grazie all'utilizzo di specifici algoritmi, è possibile bloccare email e siti web pericolosi, e in generale attacchi informatici indirizzati ad aziende e utenti.

[1] https://go.kaspersky.com/rs/802-IJN-240/images/KSB_statistics_2020_en.pdf

[2] https://www.tessian.com/blog/phishing-statistics-2020/

Infatti oggi sono presenti molteplici tecniche basate sul machine learning, che analizzando l'URL, il comportamento del sito, i contenuti e altri dati, riescono a classificare i siti web in "malevoli" o "benigni".

1.2 Motivazioni ed Obiettivi

Come accennato nella sezione precedente, negli ultimi anni lo sviluppo di tecniche per la classificazione di siti web, e quindi tecniche create con lo scopo di proteggere gli utenti che navigando nel web potrebbero incappare in siti web pericolosi, si è basato sull'utilizzo del machine learning. La maggior parte di queste tecniche si basa sull'analizzare il livello "superficiale" del sito web, ad esempio la struttura della pagina, l'URL, i contenuti e il codice JavaScript. Il problema di quest'analisi è che può fallire con le cosiddette tecniche di offuscamento, cioè tecniche che cercano di nascondere il codice JS malevolo. Oppure analizzando solo l'URL non si sta considerando la possibile presenza di siti web malevoli che hanno un URL assolutamente normale e con un formato uguale agli URL dei siti benigni. Un'ulteriore esempio è dato da siti malevoli creati con la stessa interfaccia dei siti benigni, e che cercano di estrapolare dati sensibili all'utente. Quindi siti web che rispecchiano gli standard dei siti web benigni, ma che in realtà hanno uno scopo malevolo.

Di conseguenza, lo scopo della tesi è effettuare un'analisi dello stato dell'arte, riportando i vantaggi e gli svantaggi di alcune tecniche di classificazione. Per poi analizzare le tecniche single e cross-layer, in grado di classificare i siti web sfruttando non solo i dati del livello applicazione, come quelli già elencati, ma anche i dati del livello rete, cioè dati estrapolati dai protocolli di comunicazione tra client e server. Successivamente si discute dell'implementazione della tecnica realizzata per effettuare un confronto empirico, e quindi confrontare i risultati ottenuti, sia in termini di single che di cross-layer, con i risultati della stessa tecnica già realizzata da Xu [2014] ma con un altro dataset.

1.3 Risultati ottenuti

I risultati ottenuti dalle due implementazioni mostrano che l'algoritmo che presenta le metriche migliori è l'albero decisionale J48, sia in termini di single-layer che di cross-layer. Inoltre con la XOR-aggregation e la data-aggregation presenta un accuracy superiore al 97% e falsi positivi e falsi negativi quasi nulli.

Ulteriori uguaglianze tra i risultati sono date delle metriche ottenute dagli algoritmi SVM, Logistic Regression e Naive Bayes, che infatti hanno mostrato falsi positivi e falsi negativi alti in entrambe le implementazioni.

Questi punti in comune dimostrano che l'idea della classificazione cross-layer potrebbe essere generalizzabile su vari dataset, e che funziona in modo ottimale solo sotto certe condizioni.

1.4 Struttura della tesi

All'interno di questa sezione verrà esposta la struttura della tesi, la quale è composta da cinque capitoli con le rispettive sotto sezioni.

Il primo capitolo fornisce un introduzione al lavoro svolto, descrivendo il contesto applicativo, e quindi alcuni dati per capire la gravità degli attacchi informatici, le motivazioni e gli obiettivi da raggiungere. Infine una breve panoramica dei risultati ottenuti.

Il secondo capitolo oltre a descrivere concetti fondamentali per capire il lavoro svolto, presenta un analisi di alcune tecniche oggi utilizzate per il rilevamento di siti web malevoli. Sottolineando i problemi e i vantaggi di tali tecniche.

All'interno del terzo capitolo si discute dell'implementazione di una tecnica single-layer e cross-layer. In particolare, nella prima sezione si discute dell'implementazione realizzata da Xu [2014], nella seconda sezione si discute dell'implementazione realizzata su un dataset diverso per effettuare un confronto tra i risultati ottenuti dalle due implementazioni. E quindi confermare o confutare, almeno in parte, i risultati del suddetto autore.

Il quarto capitolo è diviso in tre sezioni: la prima analizza i risultati ottenuti da Xu [2014], la seconda i risultati ottenuti dall'implementazione di confronto, e la terza sezione mette in evidenza le differenze e le uguaglianze dei risultati ottenuti dalle due implementazioni.

Nell'ultimo capitolo si riassumono i risultati più importanti e gli sviluppi futuri.

CAPITOLO 2

Background e stato dell'arte

Per capire a fondo le tecniche e i problemi descritti nei capitoli successivi è indispensabile fornire al lettore alcune definizioni, e descrizioni di alcuni algoritmi. Di conseguenza nel capitolo 2.1 verranno presentati gli elementi necessari per comprendere il lavoro svolto. Successivamente, nel capitolo 2.2, verranno analizzate delle tecniche per il rilevamento di siti web malevoli.

2.1 Background

2.1.1 Significato di cross e single-layer

Un sito web può essere visto come un insieme di pagine HTML, statiche o generate dinamicamente, che risiedono su un server web e vengono inviate ai client grazie al protocollo HTTP o HTTPS. I browser, si occupano di creare una request HTTP e di inviarla al server che risponderà con una response HTTP, contente codice che viene interpretato ed eseguito dal browser. Quindi le due macchine (client e server) comunicano grazie alla rete Internet, che a sua volta si basa sullo stack ISO/OSI. Partendo dal basso, i livelli dello stack sono: fisico, data link, rete, trasporto, sessione, presentazione e applicazione. Ogni livello punta a risolvere dei problemi specifici grazie a standard e protocolli.

L'approccio cross-layer, in questo caso, si basa sull'analizzare i dati provenienti da più livelli dello stack ISO/OSI (come il numero di pacchetti TCP), i dati estraibili dalla pagina

HTML (come il numero di contenuti invisibili) e i dati che si ottengono da servizi come il DNS.

L'approccio single-layer, al contrario del cross-layer, si basa sull'analisi dei dati provenienti da un solo livello ISO/OSI, in particolare nei capitoli 3 e 4 verranno discussi due approcci single-layer: uno basato sul livello rete e uno sul livello applicazione.

2.1.2 Quando un sito è considerato malevolo

Un sito malevolo è un sito web che è stato progettato o modificato per causare danni alla macchina client o all'utente.

Tra le tipologie più comuni di siti malevoli troviamo i siti web che cercano di trarre in inganno l'utente, clonando e simulando l'interfaccia grafica di un sito benigno (ad esempio il sito di una banca). Questa tipologia di attacco è nota con il nome di "phishing" e ha lo scopo di estrapolare password, codici e informazioni personali dell'utente.
Inoltre un sito benigno può essere modificato dai cybercriminali per reindirizzare il client su un URL malevolo, questo può avvenire modificando i file di configurazione o i file sorgenti del sito benigno, Mohammad et al. [2015]. Quindi anche un sito progettato per scopi leciti, che però è stato manomesso, risulta un sito malevolo. Nei successivi capitoli per far riferimento a un sito malevolo verranno utilizzati vari sinonimi come: URL malevolo, pagina pericolosa, sito pericoloso…

2.1.3 Approccio statico e approccio dinamico

Per classificare un sito web ci sono due approcci: approccio statico e approccio dinamico. Quello statico si basa sull'analisi dell'URL e/o del contenuto del sito web, senza osservare il comportamento del sito a runtime. E' un approccio efficace per un grande numero di siti web ma fallisce con attacchi basati sull'offuscamento, Ma et al. [2009], Rajab et al. [2011].

L'approccio dinamico invece si basa sull'analisi del comportamento a runtime del sito, anche questo approccio è efficace, ma non è efficiente. Infatti richiede l'uso di client honeypot e quindi emulazione di browser e sistemi operativi, Canali et al. [2011].

2.1.4 Classificazione

La classificazione, come dice la parola stessa, consiste nel predire una variabile categorica con lo scopo di classificare degli elementi analizzando le caratteristiche già note (chiamate anche features). L'algoritmo, anche detto classificatore, viene addestrato sul training set, cioè

un insieme di istanze già correttamente classificate ed etichettate. Successivamente viene valutato su un altro insieme di istanze chiamato test set, da questa fase si ottengono diverse metriche che indicano la qualità dell'algoritmo. In generale possiamo dire che un insieme di istanze utilizzate dal modello viene chiamato dataset.
In questo contesto la variabile categorica può assumere due valori: maligno o benigno, e ovviamente indica la pericolosità del sito web. Il training set sarà formato da un insieme di righe, dove ogni riga contiene le features di un sito web precedentemente classificato.

2.1.5 Support-vector machine

Il Support-vector machine (SVM) è un algoritmo di machine learning capace di apprendere da un insieme di istanze (training set) già classificate ed etichettate.

SVM mappa gli esempi del training set in punti nello spazio con l'obiettivo di massimizzare la distanza tra le due categorie (in questo caso siti malevoli e siti benigni). Per la separazione delle istanze SVM usa gli iperpiani, intuitivamente una buona separazione è data dall'iperpiano che ha la più grande distanza al punto più vicino di ogni classe. Per chiarire questo concetto si riporta un esempio di uno spazio bidimensionale:

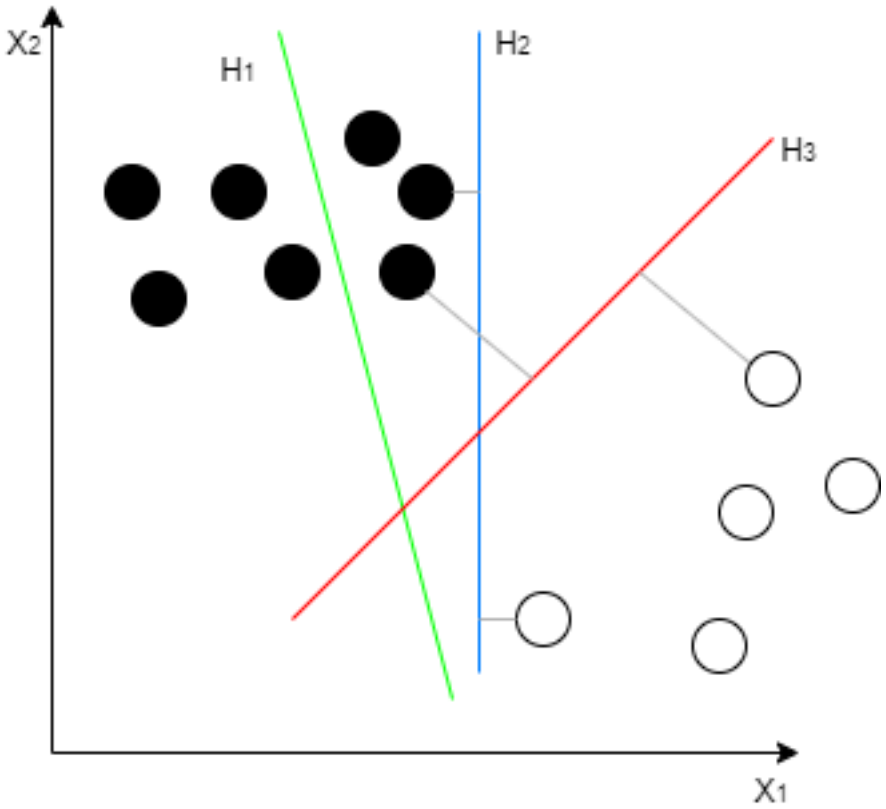

Figura 2.1: Esempio di separazione lineare usando le SVM

L'iperpiano H1 non separa correttamente le istanza, e quindi può essere scartato. L'iperpiano H2 invece separa le istanze ma lo fa con un margine minimo. L'iperpiano H3 è quello ideale siccome separa le istanze e massimizza la distanza tra i due punti più vicini delle due classi. Quando il modello deve classificare una nuova istanza la mappa nello stesso spazio. Se ricade "sotto" l'iperpiano H3 allora la nuova istanza viene classificata come "Bianca", se ricade "sopra" allora viene classificata come "Nera".

Il numero delle dimensioni che formano lo spazio è dato dal numero di features, con tre features si ottiene uno spazio di tre dimensioni e un iperpiano di due dimensioni, quattro features implicano uno spazio di quattro dimensioni e un iperpiano di tre, e cosi via. Le istanze che sono più vicine all'iperpiano vengono chiamate "support vector".

Senza scendere troppo nel dettaglio, il mapping delle istanze nel piano avviene grazie alla funzione di kernel, questo rappresenta un punto di forza dell'SVM perché oltre alle funzioni standard si possono creare funzioni di kernel ad-hoc, e quindi SVM risulta molto versatile. Inoltre lavora bene quando le istanze sul piano sono ben separate e con la minima presenza di accavallamenti o rumore, Jakkula [2006].

2.1.6 Alberi decisionali e Random forest

L'algoritmo di classificazione "Decision tree" mira a creare un albero i cui nodi rappresentano un sotto-insieme di caratteristiche e i cui archi rappresentano delle decisioni.
Nel contesto della classificazione di siti web un albero di esempio potrebbe essere il seguente:

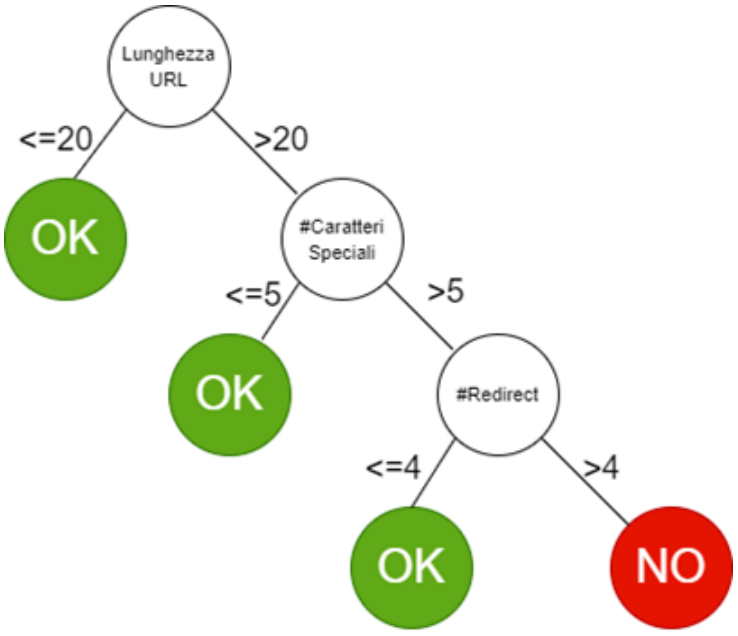

Figura 2.2: Esempio di decision tree

Gli alberi decisionali sono particolarmente utili perché permettono un alto livello di leggibilità, infatti come è facilmente intuibile dalla figura di esempio, un sito web viene classificato maligno solo se la lunghezza dell'URL supera i venti caratteri, il numero di caratteri speciali è maggiore di cinque e il numero di redirect è maggiore di quattro. In tutti gli altri casi viene classificato come benigno.
La radice dell'albero sarà la caratteristica che riesce a dividere meglio il dataset, cioè la caratteristica con information gain massima. Il nodo che è figlio della radice avrà un information gain minore rispetto al padre ma maggiore rispetto al nodo figlio. Quindi un nodo al livello n avrà un information gain minore rispetto al nodo $n-1$, ma maggiore rispetto al nodo del livello $n+1$.
Nei capitoli successivi, l'algoritmo "Decision tree" sarà indicato con vari nomi, tra cui "J48", cioè l'implementazione fornita dal tool Weka.

L'algoritmo Random forest è un algoritmo di ensemble. In pratica costruisce un certo numero di alberi decisionali e calcola la predizione finale seguendo la tecnica del majority voting. Il che significa che la predizione finale del Random forest è uguale alla predizione fatta dalla maggior parte degli alberi decisionali. Ad esempio se sette alberi su dieci considerano un sito web malevolo, allora il Random forest classifica il sito come malevolo.
Una cosa molto importante da notare è che il Random forest crea gli alberi decisionali scegliendo le features randomicamente e non calcolando l'information gain. Se considerasse l'information gain creerebbe alberi tutti uguali il che non avrebbe senso perché farebbero tutti la stessa predizione.

2.2 Stato dell'arte

All'interno di questo capitolo si discute di alcune delle tecniche più caratteristiche e già presenti, sulla rilevazione di siti web malevoli. Alcune si basano sull'analisi del contenuto, altre sulla struttura del DOM e del codice JavaScript, altre considerano l'URL.
Una cosa che accumuna sicuramente queste tecniche è l'utilizzo del machine learning, infatti facendo feature extraction, ad esempio, sulla base dei record forniti dal servizio WHOIS, dall'URL del sito o dai redirect è possibile ottenere grandi moli di dati su cui addestrare i modelli di machine learning.

- Approccio basato sull'analisi dell'URL e del DOM, sviluppato da Do Xuan et al. [2020] e da Zhang et al. [2011] ;

- Approccio basato sulla creazione di "Redirection Subgraph", sviluppato da Shibahara et al. [2017];

2.2.1 Analisi dell'URL

L'approccio sviluppato da Zhang et al. [2011] si basa sull'utilizzo dei metodi di apprendimento online per classificare le pagine maligne. L'addestrando dei modelli avviene utilizzando le informazioni ricavate dall'URL, quindi non bisogna scaricare e analizzare la pagina web, e di conseguenza il client non corre nessun rischio.
Inoltre l'analisi del contenuto web non viene evitato solo per questioni di sicurezza, ma anche perché l'analisi delle pagine web è un lavoro complesso e il classificatore avrebbe prestazioni peggiori.
Le features ricavate dall'URL si dividono in due gruppi: features lessicali e feature host-based (come l'IP address, servizio Whois...). Sulle features estratte vengono poi addestrati i i modelli di machine learning, in questo caso è stato scelto l'apprendimento online perché gli algoritmi sono in grado di elaborare grandi numeri di esempi in poco tempo e perché si adattano ai cambiamenti che possono avvenire sugli URL.

Nello specifico gli algoritmi utilizzati sono: Perceptron, Passive-Aggressive Algorithm e Confidence-Weighted Algorithm. Di seguito si riportano le metriche ottenute addestrando gli algoritmi su 881mila URL e applicando la tenfold cross-validation.

Algoritmo	Accuracy	TPR	TNR
Perceptron	0.95	0.85	0.97
PA	0.96	0.92	0.97
CW	0.96	0.82	0.99

Tabella 2.1: Risultati di Zhang et al. [2011]

Un lavoro simile è stato svolto da Do Xuan et al. [2020] , infatti per poter addestrare l'algoritmo Random Forest e il Support Vector Machine, hanno eseguito la fase di estrazione delle feature basandosi sull'URL e sul DOM del sito. In particolare dividono le features in tre gruppi:

- Lexical group: contiene le feature estratte interamente dall'URL. Come la lunghezza, il numero di underscore, cancelletti e tilde, l'utilizzo o meno del protocollo HTTPS,

l'indirizzo IP, la lunghezza della query...

- Host-based feature group: contiene le feature estratte dal DOM della pagina web e dal codice JS. Come il numero di funzioni eval() ed exec(), la presenza o meno di popup e iframe...
- Correlated feature group: contiene le feature correlate tra di loro.

Gli algoritmi citati sopra sono stati poi addestrati su due dataset, di seguito la tabella delle metriche:

Dataset	Algoritmo	Accuracy	Precision	Recall
10.000 URLs	SVM (100 iterazioni)	93.39	94.67	92.51
	SVM (10 iterazioni)	93.35	94.84	92.71
	RF (10 alberi)	99.10	98.43	97.45
	RF (100 alberi)	99.77	98.75	97.85
470.000 URLs	SVM (100 iterazioni)	90.70	93.43	88.45
	SVM (10 iterazioni)	91.07	93.75	88.85
	RF (10 alberi)	95.45	90.21	95.12
	RF (100 alberi)	96.28	91.44	94.42

Tabella 2.2: Risultati di Do Xuan et al. [2020]

Le metriche ottenute dagli algoritmi utilizzati dal lavoro di Do Xuan et al. [2020] e di Zhang et al. [2011] dimostrano che l'approccio basato su URL è sicuramente una buona soluzione. Però bisognerebbe considerare la presenza di siti web malevoli con URL che rispecchiano i formati dei siti benigni infatti, come riportato da Rao et al. [2020], i modelli basati sul Random Forest potrebbero fallire con gli URL di lunghezza corta.

2.2.2 Analisi dei Redirection Subgraph

Questo approccio si basa sull'analisi dei redirect, infatti molti attacchi informatici puntano a compromettere siti benigni con lo scopo di reindirizzare il client in siti e pagine web maligne.
Oltre all'analizzare i redirect, il lavoro di Shibahara et al. [2017], punta ad analizzare solo una piccola parte del DOM e del codice JS, questo perché collezionare "tutti" i dati di un sito

malevolo non è affatto facile, siccome gli hackers utilizzano spesso tecniche di evasione e offuscamento. Di conseguenza le feature estratte, oltre ai redirect, sono il numero di contenuti invisibili, il numero di funzioni eval(), presenza di codice eseguibile nella shell del browser, contenuti offuscati…

Questa tecnica si basa sull'utilizzo dei "Redirection Subgraph", in particolare l'obiettivo è quello di creare dei grafi dove i nodi sono gli URLs e gli archi sono i redirect tra le pagine web. Quindi l'estrazione delle feature non si basa solo sul codice JS e HTML, ma considera anche i redirect, che come dice l'autore, possono essere di tre tipi: http 3xx, HTML tag (link, iframe…) e JavaScript (document.write, innerHTML…).

La classificazione di un sito, in questo caso, si basa sulla somiglianza del grafo ottenuto analizzando il sito da classificare e dei grafi presenti nel dataset, usato per l'addestramento del modello di machine learning. Per analizzare due grafi si estraggono i sotto-grafi, e la similarità viene calcolata sulla base del numero di sotto-grafi che sono condivisi dai due grafi. Da ogni percorso del grafo il sistema calcola un sotto-grafo che è rappresentabile attraverso una coppia di vettori, dove il primo vettore rappresenta informazioni sui vertici e il secondo vettore rappresenta informazioni sugli archi, e cioè sui redirect.

Di seguito viene riportato un esempio di grafo.

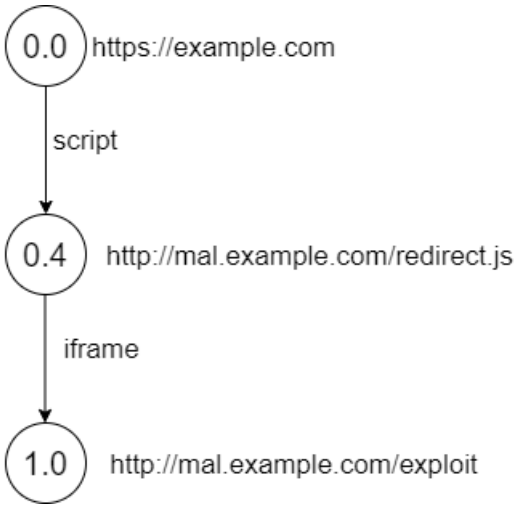

Figura 2.3: Esempio di redirection subgraph

Possiamo osservare un grafo con tre vertici, ogni vertice raffigura un sito o pagina web ed è affiancato dal proprio URL, inoltre abbiamo due redirect: il primo effettuato da uno script e il secondo da un iframe. All'interno di ogni vertice è presente un numero che rappresenta il grado di affidabilità, questo numero viene calcolato utilizzando l'algoritmo Random Forest.

In questo esempio la coppia di vettori che ne deriva è: $sg = ([0.0, 0.4, 1.0], [script, iframe])$
Successivamente la tecnica prosegue con la template construction, usando una algoritmo di clustering che calcola la massima somiglianza tra due grafi, se la somiglianza supera una certa soglia allora i due grafi vengono fusi. Questo processo è iterato finché ogni cluster è composto da un redirection graph e quindi nessun cluster può essere fuso con un altro. Quindi si procede con la feature extraction, dove si codifica un redirection graph in un valore numerico, calcolando la similarità tra il template e i sotto-grafi. In altre parole di costruisce un vettore $x = [S(SG, T_1), \dots S(SG, T_N)]$, dove SG è il grafo, T_i è il template i-esimo, N è il numero di template, e $S()$ è la funzione di similarità. Infine questi vettori verranno utilizzati dal'algoritmo di classificazione Random Forest, che utilizerrà i template costrutiti nella fase precedente come label.
Dalla fase di validazione si ottengono le seguenti metriche:

Algoritmo	Recall	TPR	Precision	f-measure
Random Forest	0.9057	0.9171	0.6631	0.7657

Tabella 2.3: Risultati di Shibahara et al. [2017]

Purtroppo le metriche in tabella sono soggette a degradazione con il passare del tempo, infatti la nascita e l'aggiornamento di nuovi exploit kits potrebbero rendere i siti web difficili da classificare.
Il secondo problema si presenta durante l'analisi dei siti web costruiti attraverso l'uso di un CMS, infatti il grafo che ne deriva è simile ai grafi ottenuti dall'analisi di siti malevoli, grazie all'utilizzo di molti file JavaScript. Questo è sicuramente un grave problema, considerando l'ampio uso di WordPress, Wix, Joomla! e tanti alti.

CAPITOLO 3

Re-implementazione di una tecnica di single-layer e cross-layer detection

All'interno di questo capitolo si discute della tecnica sviluppata da Xu [2014] e dell'implementazione realizzata su un dataset differente con lo scopo di confrontare i risultati della classificazione single e cross-layer. Infine è presente una sezione che riporta le differenze tra le due implementazioni.

3.1 Implementazione di Xu [2014]

Di seguito si discute la tecnica implementata da Xu [2014]. La prima sezione riporta il processo di data collection, in cui vengono descritte anche alcune feature utilizzate. La sezione successiva parla di come le feature sono state organizzate in vettori e di come questi vettori sono stati combinati per l'addestramento degli algoritmi.

3.1.1 Data Collection e data description

La raccolta dei dati è avvenuta in maniera automatica utilizzando un sistema software chiamato "crawler". Il crawler prende in input una lista di URL e automaticamente, per ogni URL, effettua una richiesta HTTP. A questo punto inizia a conservare in un database le informazioni ricavate dalla response HTTP e dai redirect, ma anche le informazioni ricavate dall'URL stesso, dal DNS, dal servizio Whois e dai livelli application e network.

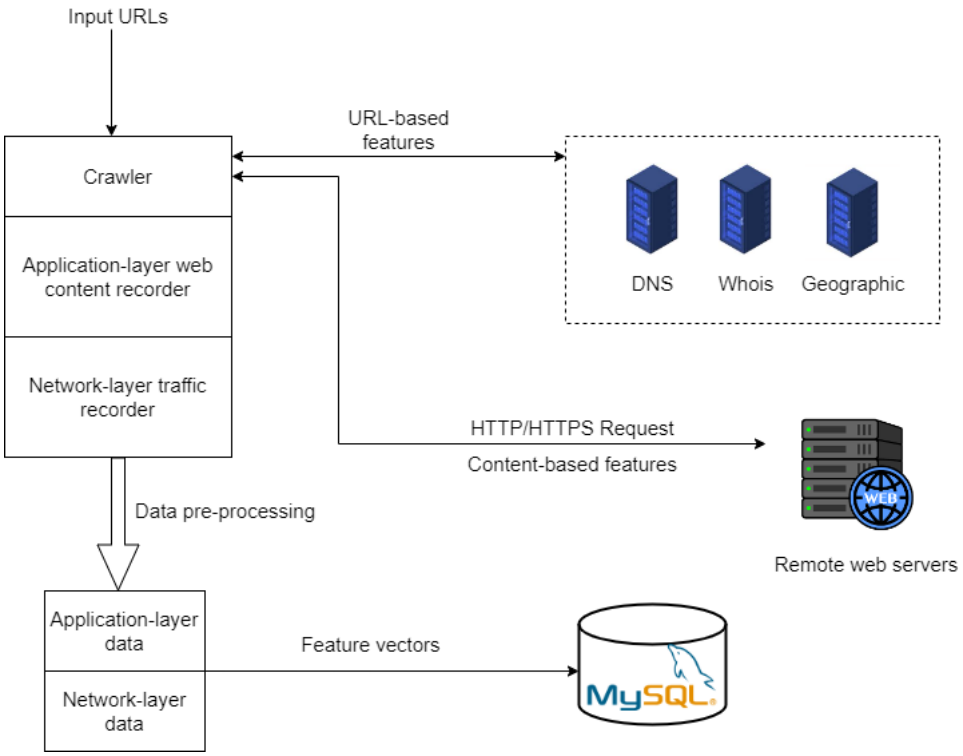

Figura 3.1: Sistema di data collection di Xu [2014]

In totale si ottengono 105 features del livello applicazione, divise in 4 sotto-classi, e 19 features dal livello rete, divise in 3 sotto-classi. Di seguito se ne riportano alcune evidenziando le sotto-classi.

- Feature livello applicazione
  - Feature basate sulle informazioni lessicali dell'URL
    - Lunghezza dell'URL: i dati dimostrano che i siti benigni hanno, in media, URL di 18.23 caratteri, i siti maligni 25,11 caratteri;
    - Numero di caratteri speciali nell'URL: i siti benigni hanno una media di 2.93 caratteri, i maligni di 3.36;
    - Presenza di un indirizzo IP nell'URL: in un URL invece del dominio può essere presente direttamente l'indirizzo IP del server;
  - Feature basate sull'header HTTP
    - Charset: indica anche la lingua del sito, l'etnia degli utenti target e la nazionalità della pagina web;
    - HTTPHeader_server: informazioni sul server web. Dai dati emergono tre web server principali: Apache, Microsoft IIS e nginx;

  - Feature basate sulle informazioni dell'host
    * Data di registrazione e di ultimo aggiornamento del servizio Whois;
    * Redirect interno o esterno: il 21.7% dei siti malevoli e il 16.1% dei siti benigni effettua redirect verso domini differenti da quello di partenza;
  - Feature basate sul contenuto della pagina web
    * Numero di redirect;
    * Numero di risorse esterne richieste durante il caricamento della pagina: immagini, video, audio, file...
    * Lunghezza del contenuto: confronta la consistenza tra la lunghezza del contenuto riportata nell'header HTTP con la lunghezza effettiva. Una lunghezza negativa potrebbe indicare un attacco di tipo buffer overflow;
    * Numero di iframe HTML;
    * Numero di funzioni JavaScript sospette: identifica la presenza di codice offuscato;

- Feature livello rete
  - Feature basate sugli attributi del server
    * Numero di pacchetti TCP inviati al server dal crawler: i siti maligni spesso usano molte risorse che causano più richieste http da inviare al server;
  - Feature basate sulla comunicazione crawler-server
    * Numero di pacchetti TCP con il flag URG (urgente): alcuni attacchi usano questo flag per bypassare i firewall che non sono correttamente configurati;
  - Feature basate sulla comunicazione crawler-DNS
    * Numero di query DNS inviate dal crawler: i siti malevoli richiedono più query DNS, grazie ad un numero di redirect maggiore rispetto ai siti benigni;

3.1.2 Data Pre-Processing e Training

Ad ogni URL vengono associati due vettori, uno è il vettore che contiene le informazioni del livello applicazione e l'altro contiene le informazioni del livello rete.

Il vettore del livello applicazione contiene informazioni come gli header HTTP, informazioni sul codice JavaScript e sui redirect. Il vettore del livello rete invece contiene le informazioni ottenute grazie al packet capture.

Riguardo ai redirect, siccome URL differenti possono avere un numero differente di redirect, i vettori associati possono avere lunghezze differenti. Per facilitare le operazioni di analisi i vettori sono stati aggregati, per le informazioni numeriche si è usata la media aritmetica, per i dati booleani si è usata l'operazione di OR e per le stringhe si è considerato solo l'URL dell'ultimo redirect.

Ad esempio considerando le features: (lunghezza-url, presenza della funzione JavaScript eval(), paese) e considerando i vettori del URL in input, del primo e del secondo redirect: (100, FALSE, US), (200, FALSE, UK) e (300, TRUE, IT) allora il vettore finale sarebbe: (200, TRUE, IT).

I due vettori verranno poi utilizzati per l'addestramento degli algoritmi: Naive Bayes, Logistic regression, SVM e J48. L'addestramento si basa su 4 modi di trattare i due vettori:

- Data-aggregation: dato un URL, si concatena il vettore delle feature estratte dal livello applicazione con il vettore delle feature provenienti dal livello rete;
- OR-aggregation: dato un URL, il modello valuterà separatamente i due vettori per poi applicare un OR logico tra i due risultati. Quindi se il modello valuterà come maligno o il vettore rete o il vettore applicazione, allora l'URL verrà classificato come maligno. Negli altri casi l'URL verrà classificato come benigno;
- AND-aggregation: dato un URL, il modello valuterà separatamente i due vettori per poi applicare un AND logico tra i due risultati. Un URL verrà classificato come maligno solo se i due vettori risulteranno maligni. Negli altri casi l'URL verrà validato come benigno;
- XOR-aggregation: dato un URL, se i due vettori risulteranno maligni allora l'URL verrà classificato come maligno. Se i due vettori risulteranno benigni allora l'URL verrà classificato come benigno. Negli altri casi l'URL verrà classificato considerano l'approccio dinamico.

Per ogni tipo di aggregazione è stato effettuato l'addestramento degli algoritmi senza tecniche di feature selection e poi utilizzando la CfsSubsetEval, InfoGainAttributeEval e Principle Component Analysis.

3.2 Implementazione per il confronto empirico

Di seguito si discute dell'implementazione realizzata per effettuare un confronto empirico con i risultati ottenuti da Xu [2014]. Quindi le tecniche e gli algoritmi saranno gli stessi implementati dal suddetto autore, però realizzati per un dataset diverso. Nella prima sezione viene riportata una breve spiegazione delle feature presenti nel dataset utilizzato. Successivamente si discute di come è stato effettuato il processo di data pre-processing, e del training degli algoritmi sia per il single che per il cross-layer.

3.2.1 Data Collection e data description

Il dataset utilizzato è presente sul sito di data science kaggle[1]. Il dataset è composto da 21 colonne e 1781 righe. Le 21 colonne sono:

- URL: identificativo anonimo dell'URL;
- URL_LENGTH: numero di caratteri dell'URL;
- CHARSET: indica il character set utilizzato dal sito;
- SERVER: indica il server web che ospita il sito;
- CONTENT_LENGTH: indica la taglia dell'header HTTP;
- WHOIS_COUNTRY: indica il paese da cui proviene la risposta HTTP;
- WHOIS_STATEPRO: indica lo stato da cui proviene la risposta HTTP;
- WHOIS_REGDATE: indica la data di registrazione del dominio;
- WHOIS_UPDATE_DATE: indica la data dell'ultimo aggiornamento del servizio Whois;
- TCP_CONVERSATION_EXCHANGE: indica il numero di pacchetti TCP scambiati tra il server e il client honeypot;
- DIST_REMOTE_TCP_PORT: indica il numero di porte rilevate e diverse da TCP;
- REMOTE_IPS: indica il numero di indirizzi IP collegati al client honeypot;
- APP_BYTES: indica il numero di byte traferiti;
- SOURCE_APP_PACKETS: indica il numero di pacchetti inviati dall'honeypot al server;

[1] `https://www.kaggle.com/datasets/xwolf12/malicious-and-benign-websites`

- REMOTE_APP_PACKETS: indica il numero di pacchetti ricevuti dal server;
- APP_PACKETS: numero di pacchetti IP generati durante la comunicazione tra honeypot e server;
- DNS_QUERY_TIMES: numero di pacchetti DNS generati durante la comunicazione tra honeypot e server;
- Type: indica la tipologia di sito web, vale 1 se il sito è malevolo, 0 altrimenti.

3.2.2 Data Pre-Processing

Feature URL

La colonna del dataset che rappresenta la feature "URL" non contiene l'URL del sito ma un identificativo anonimo, quindi non aggiunge alcuna informazione sulla natura del sito web. Di conseguenza si è deciso di rimuovere tale colonna con l'apposita funzione:

```
df.pop('URL')
```

Feature WHOIS_COUNTRY

Su questa feature sono stati rilevati 360 campi vuoti, indicati con il valore "none". Per risolvere questo problema si è deciso di usare la most frequent imputation. Però, prima di ricavare il valore più frequente, è stato applicato il lowercasing per evitare problemi dovuti alle lettere maiuscole e minuscole. Di seguito il codice

```
df["WHOIS_COUNTRY"] = df["WHOIS_COUNTRY"].astype('str').str.lower()
df["WHOIS_COUNTRY"] = df["WHOIS_COUNTRY"].replace(['none'],df['WHOIS_COUNTRY'
 ].value_counts().index.tolist()[0])
```

Feature WHOIS_STATEPRO

La colonna del dataset che rappresenta la feature "WHOIS_STATEPRO" ha due problemi. Il primo problema è la presenza di campi vuoti, indicati con la parola "None", il secondo problema è dato dai differenti modi di indicare un paese. Ad esempio è possibile trovare le due parole: "NY" e "New York" per indicare la città di New York.
Per risolvere il problema delle città si è deciso di usare un dizionario per sostituire le abbreviazioni con i rispettivi nomi estesi. Prima però, per evitare problemi dovuti alle lettere maiuscole e minuscole, si è deciso di trasformare tutta la colonna in minuscolo.

Di seguito il codice con un dizionario parziale, il dizionario originale non viene riportato per una questione di leggibilità.

```
df["WHOIS_STATEPRO"] = df["WHOIS_STATEPRO"].astype('str').str.lower()
states_code_dict = {"qld": "queensland", "nsw": "new south wales", "ab": "alberta
    ", "on": "ontario","al": "alabama","qc": "quebec"}
df["WHOIS_STATEPRO"].replace(states_code_dict, inplace=True)
```

Per risolvere il problema dei campi vuoti si è deciso di usare la most frequent imputation, questa tecnica ha permesso di non violare il vincolo tra il valore della feature WHOIS_STATEPRO e il valore della feature WHOIS_COUNTRY, ad esempio la prima può essere uguale a "new york" solo se la seconda è uguale a "us".
Per estrapolare le città più frequenti è stata usata la seguente riga di codice:

```
df.groupby(["WHOIS_COUNTRY","WHOIS_STATEPRO"]).size().reset_index(name="Time").
    sort_values(by=['WHOIS_COUNTRY'])
```

Analizzando il risultato della funzione *groupby()* è stato creato il dizionario *most_freq_states_none_replace*, che ad ogni stato associa una città, ad esempio *"fr":"paris"* indica che Parigi è la città più frequente quando WHOIS_COUNTRY è uguale a "fr". Una volta creato il dizionario si procede con la sostituzione dei valori "none".

```
    most_freq_states_none_replace = {"no": "rogaland", "lu": "luxembourg", "jp":
    "osaka","fr": "paris"}
    dict_index_state ={}
    for index, row in df.iterrows():
        if (row["WHOIS_COUNTRY"] != "none") & (row["WHOIS_STATEPRO"] == "none"):
            dict_index_state[index] = most_freq_states_none_replace.get(row["
    WHOIS_COUNTRY"])

    new_column = pd.Series(dict_index_state.values(), name='WHOIS_STATEPRO',
    index=dict_index_state.keys())
    df.update(new_column)
```

L'algoritmo itera tutte le righe del dataframe, per ogni riga controlla se WHOIS_COUNTRY è diverso da "none" e se WHOIS_STATEPRO è uguale a "none", in caso affermativo salva in un dizionario l'associazione indice:valore, dove l'indice indica il numero di riga e il valore indica la città da inserire. Finita l'iterazione si procede con aggiornare la colonna WHOIS_STATEPRO utilizzando le associazioni salvate nel dizionario *dict_index_state*.

Pre-processing sulle restanti features

A questo punto le uniche features su cui è necessario applicare tecniche di pre-processing sono: CHARSET, SERVER, WHOIS_UPDATED_DATE e WHOIS_REGDATE.
Le quattro feature presentano diversi campi vuoti, per risolvere il problema si è deciso di usare la KNN imputation. La scelta del KNN è stata una conseguenza del fatto che tra le quattro feature non c'è nessun vincolo, come ad esempio il vincolo tra WHOIS_STATEPRO e WHOIS_COUNTRY, e dal fatto che le colonne con un maggior numero di campi vuoti sono state processate precedentemente.
Per poter applicare il KNN è stato necessario applicare il lowercasing, trasformare i valori "none" nel valore "NaN", trasformare le stringhe in numeri ed effettuare la feature scaling.
Per le prime due operazioni è stato usato il seguente codice:

```
df["WHOIS_REGDATE"] = df["WHOIS_REGDATE"].astype('str').str.lower()
df["WHOIS_UPDATED_DATE"] = df["WHOIS_UPDATED_DATE"].astype('str').str.lower()
df["CHARSET"] = df["CHARSET"].astype('str').str.lower()
df["SERVER"] = df["SERVER"].astype('str').str.lower()

df["WHOIS_REGDATE"].replace('none', np.nan, inplace=True)
df["WHOIS_UPDATED_DATE"].replace('none', np.nan, inplace=True)
df["CHARSET"].replace('none', np.nan, inplace=True)
df["SERVER"].replace('none', np.nan, inplace=True)
```

Per trasformare le stringhe in numeri è stato usato il seguente algoritmo:

```
list_col_str = []
for col in df.select_dtypes(include='object').columns:
    list_col_str.append(col)
df[list_col_str] = df[list_col_str].apply(lambda series: pd.Series(
    LabelEncoder().fit_transform(series[series.notnull()]),
    index=series[series.notnull()].index
))
```

Per ogni colonna da trasformare l'algoritmo applica una funzione che ritorna una *pd.Series*, questa funzione elimina i valori NaN e addestra il LabelEncoder attraverso il metodo *fit_transform*. A questo punto si ottiene un dataset in cui le stringhe sono state sostituite da valori numerici conservando i valori NaN.

Per effettuare la feature scaling e poi addestrare l'algoritmo KNNImputer, e quindi sostituire i valori NaN con dati "reali", sono state usate le seguenti righe di codice:

```
scaler = MinMaxScaler()
df = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
```

```
    imputer = KNNImputer(missing_values=np.nan)
    df = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)
```

Quindi, a questo punto, si ottiene un dataset su cui sono state applicate le fasi di data cleaning e feature scaling.

3.2.3 Training

In questa sezione si discute dell'implemetazione usata per l'addestramento degli algoritmi e per le tecniche di feature selection. Il processo di addestramento è stato progettato seguendo le tecniche e gli algoritmi usati da Xu [2014]. Quindi gli algoritmi usati, sia per il single-layer che per il cross-layer, sono: Naive Bayes, Logistic Regression, SVM e J48. Ogni algoritmo è stato addestrato per la data-aggregation, per l'OR-aggregation e per l'AND-aggregation. Purtroppo non è stato possibile implementare la XOR-aggregation perchè, come detto nel capitolo precedente, la feature "URL" del dataset riporta un codice anonimo e non l'URL del sito. Di conseguenza risulta impossibile utilizzare la classificazione dinamica.

Per ogni tipo di aggregazione sono state applicate le tecniche di feature selection: PCA, CfsSubsetEvaluation e InformationGain. La validazione degli algoritmi è avvenuta usando la cross fold validation e le metriche riportate sono: Accuracy (Acc), Falsi Negativi (FN) e Falsi Positivi (FP).

Single-layer

La classificazione single-layer prevedere l'addestramento degli algoritmi usando solo uno dei due vettori associati ad ogni URL.

Il training degli algoritmi senza applicare nessuna tecnica di feature selection è stato implementato nel seguente modo:

```
    df = pd.read_csv("../dataset/dataset.csv")
    df = cleaning_dataframe(df)

    df_application = df[
        ['URL_LENGTH', 'NUMBER_SPECIAL_CHARACTERS', 'CHARSET', 'SERVER', '
    CONTENT_LENGTH', 'WHOIS_COUNTRY',
         'WHOIS_STATEPRO', 'WHOIS_REGDATE', 'WHOIS_UPDATED_DATE', 'Type']]
    df_network = df[
        ['TCP_CONVERSATION_EXCHANGE', 'DIST_REMOTE_TCP_PORT', 'REMOTE_IPS', '
    APP_BYTES', 'SOURCE_APP_PACKETS',
```

```
        'REMOTE_APP_PACKETS', 'SOURCE_APP_BYTES', 'REMOTE_APP_BYTES', '
    APP_PACKETS', 'DNS_QUERY_TIMES', 'Type']]

    X_application, y_application = data_balancing(df_application)
    X_network, y_network = data_balancing(df_network)

    train_model_single_layer("GAUSSIAN NB", GaussianNB(), X_network,
    X_application, y_network, y_application)
    # train_model_single_layer per gli altri algoritmi
```

Inizialmente si legge il dataset e si crea un *DataFrame* con la funzione *pd.read_csv()*. Nella riga successiva si applica la data cleaning, descritta nel capito precedente, e si procede con il dividere le feature del livello rete con quelle del livello applicazione, rispettativamente in *df_network* e *df_application*.

Infine si applica il data balancing, effettuato con l'algoritmo SMOTE nel seguente modo:

```
def data_balancing(df):
    smote = SMOTE()
    X, y = smote.fit_resample(df[df.columns.values.tolist()[:-1]], df['Type'])
    return X, y
```

e si procede con l'addestramento vero e proprio attraverso la funzione *train_model_single_layer* definita nel seguente modo:

```
    def train_model_single_layer(name_classifier, classifier, X_network,
    X_application, y_network, y_application):
    print("\n\n " + name_classifier + " NETWORK")
    print_metrics(confusion_matrix(y_network, cross_val_predict(classifier,
    X_network, y_network, cv=10)))
    print("\n\n" + name_classifier + " APPLICATION")
    print_metrics(confusion_matrix(y_application, cross_val_predict(classifier,
    X_application, y_application, cv=10)))
```

La funzione riceve in input il classificatore, i dataframe delle features del livello network e application (*X_network e X_application*) e i rispettivi array che rappresentano la variabile target (*y_network e y_application*). La funzione applica poi la cross fold validation sul classificatore, con la funzione *cross_val_predict*, genera la matrice di confusione, con la funzione *confusion_matrix*, e stampa le metriche grazie alla funzione *print_metrics*.

Riguardo al training applicando le tecniche di feature selection CfsSubsetEval e InfoGainAttributeEval è stato usato il linguaggio di programmazione Java sfruttando le API del tool Weka.

La prima tecnica è stata implementata nel seguente modo:

```
Instances instancesNetwork = Utils.getInstances(path);
Remove removeFilter = new Remove();
int[] networkFeatures = {9,10,11,12,13,14,15,16,17,18,19};
//int[] applicationFeatures = {0,1,2,3,4,5,6,7,8,19};
removeFilter.setAttributeIndicesArray(networkFeatures);
removeFilter.setInvertSelection(true);
removeFilter.setInputFormat(instancesNetwork);
Instances instances = Filter.useFilter(instancesNetwork, removeFilter);
```

Inanzitutto si legge il dataset, su cui è stato già applicato il processo di data cleaning e di feature scaling, e si rimuovono le features di livello network lasciando quelle del livello application, o viceversa.

Una volta applicato il filtro si procede con l'algoritmo di selezione.

```
AttributeSelection attributeSelection = new AttributeSelection();
CfsSubsetEval cfsSubsetEval = new CfsSubsetEval();
BestFirst bestFirst = new BestFirst();

attributeSelection.setInputFormat(instances);
attributeSelection.setEvaluator(cfsSubsetEval);
attributeSelection.setSearch(bestFirst);

Instances instancesEval = Filter.useFilter(instances, attributeSelection);
Utils.printCSV("./dataset/subsetEvalDatasetSingleLayerNetwork.csv",
instancesEval);
//Utils.printCSV("./dataset/subsetEvalDatasetSingleLayerApplication.csv",
instancesEval);

instancesEval.enumerateAttributes().asIterator().forEachRemaining(item->
System.out.println(item));
```

Quindi si crea un oggetto *AttributeSelection*, si setta il metodo di ricerca e l'evaluator e si applica il filtro. A questo punto si ottiene un dataset con solo le features che risultano più rilevanti. Quindi infine il dataset viene salvato in un file csv che verrà poi utilizzato per addestrare gli algoritmi e nella console vengono stampati i nomi delle features selezionate.

I dataset prodotti dalla CfsSubsetEval quindi vengono letti e bilanciati grazie alla funzione *data_balancing* per poi addestrare gli algoritmi:

```
df_sub_eval_application = pd.read_csv("../dataset/
subsetEvalDatasetSingleLayerApplication.csv")
df_sub_eval_network = pd.read_csv("../dataset/
subsetEvalDatasetSingleLayerNetwork.csv")
```

```
3
4     X_subset_application, y_subset_application = data_balancing(
      df_sub_eval_application)
5     X_subset_network, y_subset_network = data_balancing(df_sub_eval_network)
6
7     train_model_single_layer("GAUSSIAN NB", GaussianNB(),    X_subset_network,
      X_subset_application, y_subset_network, y_subset_application)
8     # train_model_single_layer per gli altri algoritmi
```

La tecnica InfoGainAttributeEval è stata implementata in modo molto simile alla precedente. Infatti dopo aver applicato il filtro di rimozione features, precedentemente riportato, è stato eseguito il seguente codice:

```
1     AttributeSelection attributeSelection = new AttributeSelection();
2     InfoGainAttributeEval eval = new InfoGainAttributeEval();
3     Ranker ranker = new Ranker();
4
5     ranker.setNumToSelect(numToSelect);
6     attributeSelection.setInputFormat(instances);
7     attributeSelection.setEvaluator(eval);
8     attributeSelection.setSearch(ranker);
9
10    Instances instancesEval = Filter.useFilter(instances, attributeSelection);
11    Utils.printCSV("./dataset/infoGainDatasetSingleLayerNetwork.csv",
      instancesEval);
12    //Utils.printCSV("./dataset/infoGainDatasetSingleLayerApplication.csv",
      instancesEval);
13    instancesEval.enumerateAttributes().asIterator().forEachRemaining(item->
      System.out.println(item));
```

Quindi dopo aver creato un oggetto di tipo *InfoGainAttributeEval* e un oggetto di tipo *Ranker* si setta il numero di features da conservare. Questo numero è impostato a cinque per le features di livello rete e a sette per il livello application, questi numeri sono stati scelti in base ai risultati della tecnica CfsSubsetEval (tenendo in considerazione anche i risultati cross-layer descritti successivamente), ai risultati ottenuti da Xu [2014] e da considerazioni fatte osservando il variare delle metriche di valutazione in base al numero di features selezionate.

Nelle righe successive si setta l'evaluator e il metodo di ricerca, si applica il filtro, si salva il nuovo dataset in un file csv e si stampano i nomi delle features nella console.

I dataset prodotti quindi vengono letti, bilanciati e usati per addestrare gli algoritmi:

```
1     df_info_gain_application = pd.read_csv("../dataset/
      infoGainDatasetSingleLayerApplication.csv")
```

```
    df_info_gain_network = pd.read_csv("../dataset/
    infoGainDatasetSingleLayerNetwork.csv")

    X_info_gain_application, y_info_gain_application = data_balancing(
    df_info_gain_application)
    X_info_gain_network, y_info_gain_network = data_balancing(
    df_info_gain_network)

    train_model_single_layer("GAUSSIAN NB", GaussianNB(), X_info_gain_network,
    X_info_gain_application, y_info_gain_network, y_info_gain_application)
    # train_model_single_layer per gli altri algoritmi
```

Il Principal Component Analysis è stato sviluppato nel seguente modo:

```
    df_pca = pd.read_csv("../dataset/dataset.csv")
    df_pca = cleaning_dataframe(df_pca)

    df_pca_application = df_pca[
        ['URL_LENGTH', 'NUMBER_SPECIAL_CHARACTERS', 'CHARSET', 'SERVER', '
    CONTENT_LENGTH', 'WHOIS_COUNTRY',
         'WHOIS_STATEPRO', 'WHOIS_REGDATE', 'WHOIS_UPDATED_DATE']]
    df_pca_network = df_pca[
        ['TCP_CONVERSATION_EXCHANGE', 'DIST_REMOTE_TCP_PORT', 'REMOTE_IPS', '
    APP_BYTES', 'SOURCE_APP_PACKETS',
         'REMOTE_APP_PACKETS', 'SOURCE_APP_BYTES', 'REMOTE_APP_BYTES', '
    APP_PACKETS', 'DNS_QUERY_TIMES']]

    y = df_pca['Type'].astype('int')

    pca_application = PCA(n_components=7)
    df_pca_application = pd.DataFrame(pca_application.fit_transform(
    df_pca_application))

    pca_network = PCA(n_components=5)
    df_pca_network = pd.DataFrame(pca_network.fit_transform(df_pca_network))

    smote = SMOTE()
    X_pca_application, y_pca_application = smote.fit_resample(df_pca_application,
     y)

    smote = SMOTE()
    X_pca_network, y_pca_network = smote.fit_resample(df_pca_network, y)

```

```
    train_model_single_layer("GAUSSIAN NB", GaussianNB(), X_pca_network,
    X_pca_application, y_pca_network, y_pca_application)
    # train_model_single_layer per gli altri algoritmi
```

Dopo aver letto il dataset e aver chiamato la funzione *cleaning_dataframe*, si divide il dataframe in base alle features: nel dataframe *df_pca_network* vengono conservate solo le features del livello network e in *df_pca_application* quelle del livello applicazione. Quindi si applica il PCA su ogni dataframe per poi bilanciarli con SMOTE. Infine si procede con l'addestramento.

Cross-layer

La classificazione cross-layer si basa sull'usare entrambi i vettori associati ad ogni URL. I vettori vengono combinati in tre modi: data-aggregation, AND-aggregatione e OR-aggregation, spiegati nei capitoli precedenti.

Il training senza nessuna tecnica di feature selection è stato implementato attraverso il seguente codice:

```
    df = pd.read_csv("../dataset/dataset.csv")
    df = cleaning_dataframe(df)

    X, y = data_balancing(df)

    train_model_data_aggregation("J48", DecisionTreeClassifier(), X, y)
    # train_model_data_aggregation per gli altri algoritmi

    application_df = X[
        ['URL_LENGTH', 'NUMBER_SPECIAL_CHARACTERS', 'CHARSET', 'SERVER', '
    CONTENT_LENGTH', 'WHOIS_COUNTRY',
         'WHOIS_STATEPRO', 'WHOIS_REGDATE', 'WHOIS_UPDATED_DATE']]
    network_df = X[
        ['TCP_CONVERSATION_EXCHANGE', 'DIST_REMOTE_TCP_PORT', 'REMOTE_IPS', '
    APP_BYTES', 'SOURCE_APP_PACKETS',
         'REMOTE_APP_PACKETS', 'SOURCE_APP_BYTES', 'REMOTE_APP_BYTES', '
    APP_PACKETS', 'DNS_QUERY_TIMES']]

    train_model_and_or_aggregation("J48", DecisionTreeClassifier(),
    application_df, network_df, y)
    # train_model_and_or_aggregation per gli altri algoritmi
```

Come primo passo si legge il dataset e si applica il processo di data cleaning e feature scaling grazie alla funzione *cleaning_dataframe*, per poi applicare il data balancing. E quindi proce-

dere con l'addestramento usando la data-aggregation. La funzione *train_model_data_aggregation* è definita nel seguente modo:

```
def train_model_data_aggregation(name_classifier, classifier, X, y):
    print("\n" + name_classifier + " data-aggregation" + "-" * 60)
    print_metrics(confusion_matrix(y, cross_val_predict(classifier, X, y, cv=10))
    )
```

La funzione applica la cross-fold validation sul classificatore, genera la matrice di confusione e stampa le metriche grazie alla funzione *print_metrics*.

A questo punto si divide il dataframe in *network_df*, con solo le features del livello rete, e *application_df*, con solo le features del livello applicazione. Infine si addestrano gli algoritmi secondo la logica AND-aggregation e OR-aggregation grazie alla funzione *train_model_and_or_aggregation* definita nel seguente modo;

```
def train_model_and_or_aggregation(name_classifier, classifier, application_df,
    network_df, y):
    print("\n" + name_classifier + "-" * 60)
    pred_or_agg_application = cross_val_predict(classifier, application_df, y, cv
    =10)
    pred_or_agg_network = cross_val_predict(classifier, network_df, y, cv=10)
    or_agg = or_aggregation(pred_or_agg_application, pred_or_agg_network)
    and_agg = and_aggregation(pred_or_agg_application, pred_or_agg_network)
    print(name_classifier + "OR-aggregation" + "-" * 10)
    print_metrics(confusion_matrix(y, or_agg))
    print(name_classifier + "AND-aggregation" + "-" * 10)
    print_metrics(confusion_matrix(y, and_agg))
```

La funzione riceve in input il nome del classificatore, il classificatore, il dataframe delle features del livello rete (*network_df*) e quello delle features applicazione (*application_df*) e la colonna della variabile target. Attraverso la funzione *cross_val_predict* si valuta l'algoritmo sul dataframe *network_df* e sul dataframe *application_df* separatamente. Nelle righe successive viene utilizzata la funzione *or_aggregation*, per unire le predizioni secondo la logica OR-aggregation, e la funzione *and_aggregation*, per unire le predice secondo la logica AND-aggregation. Come operazione finale vengono stampate le metriche di valutazione.

Le funzioni *and_aggregation* e *or_aggregation* ricevono in input due array: uno contiene le predizioni dell'algoritmo addestrato usando i vettori delle features del livello rete e l'altro contiene le predizioni dello stesso algoritmo addestrato però usando i vettori delle features del livello applicazione. Dopo aver invocato la funzione *check*, che ha lo scopo di verificare la

correttezza dei parametri, creano un terzo array in cui la casella i-esima corrisponde all'AND (oppure OR) della casella i-esima dei due array in input. Di seguito il codice:

```
def or_aggregation(application_pred, network_pred):
    check(application_pred, network_pred)
    or_agg = np.array([])
    for i in range(len(application_pred)):
        or_agg = np.append(or_agg, application_pred[i] or network_pred[i])

    return or_agg


def and_aggregation(application_pred, network_pred):
    check(application_pred, network_pred)
    and_agg = np.array([])
    for i in range(len(application_pred)):
        and_agg = np.append(and_agg, application_pred[i] and network_pred[i])

    return and_agg
```

L'implementazione della tecnica CfsSubsetEval si è basata sull'utilizzo delle API di Weka:

```
    Instances instances = Utils.getInstances("./dataset/
    datasetDataCleaningScalingFinal.arff");

    AttributeSelection attributeSelection = new AttributeSelection();
    CfsSubsetEval cfsSubsetEval = new CfsSubsetEval();
    BestFirst bestFirst = new BestFirst();

    attributeSelection.setInputFormat(instances);
    attributeSelection.setEvaluator(cfsSubsetEval);
    attributeSelection.setSearch(bestFirst);

    Instances instancesEval = Filter.useFilter(instances, attributeSelection);
    Utils.printCSV("./dataset/datasetSubsetEval.csv",instancesEval);

    instancesEval.enumerateAttributes().asIterator().forEachRemaining(item->
    System.out.println(item));
```

Innanzitutto viene letto il dataset su cui è già stato eseguito il processo di data cleaning e feature scaling, vengono istanziati gli oggetti necessari e settati l'evaluator e il metodo di ricerca. Gli ultimi tre step consistono nell'applicare il filtro, salvare il nuovo dataset e

stampare a video il nome delle feature selezionate. Per il training, il nuovo dataset viene utilizzato nel seguente modo:

```
df_subset_eval = pd.read_csv("../dataset/datasetSubsetEval.csv")

X_eval, y_eval = data_balancing(df_subset_eval)

train_model_data_aggregation("J48", DecisionTreeClassifier(), X_eval, y_eval)
# train_model_data_aggregation per gli altri algoritmi

application_df = X_eval[
['CONTENT_LENGTH', 'WHOIS_COUNTRY', 'WHOIS_STATEPRO', 'WHOIS_REGDATE', '
WHOIS_UPDATED_DATE']]

network_df = X_eval[['DIST_REMOTE_TCP_PORT', 'REMOTE_APP_PACKETS']]

train_model_and_or_aggregation("J48", DecisionTreeClassifier(),
application_df, network_df, y_eval)
# train_model_and_or_aggregation per gli altri algoritmi
```

Quindi viene letto il dataset, lo si bilancia e si addestrano gli algoritmi per la data-aggregation. Infine si dividono le due tipologie di features e si procede con l'addestramento per l'AND-aggregation e per l'OR-aggregation.

Per la tecnica InfoGainAttributeEval è stato sviluppato un procedimento molto simile alla tecnica precedente, di seguito il codice:

```
Instances instances = Utils.getInstances("./dataset/
datasetDataCleaningScalingFinal.arff");

AttributeSelection attributeSelection = new AttributeSelection();
InfoGainAttributeEval eval = new InfoGainAttributeEval();
Ranker ranker = new Ranker();

ranker.setNumToSelect(7);
attributeSelection.setInputFormat(instances);
attributeSelection.setEvaluator(eval);
attributeSelection.setSearch(ranker);

Instances instancesEval = Filter.useFilter(instances, attributeSelection);
Utils.printCSV("./dataset/infoGainDataset.csv",instancesEval);

```

```
15    instancesEval.enumerateAttributes().asIterator().forEachRemaining(item->
      System.out.println(item));
```

Dopo aver letto il dataset e istanziato gli oggetti necessari, si setta il numero delle feature da conservare, l'evaluator e il metodo di ricerca. Nelle righe successive si applica il filtro, si salva il nuovo dataset e si stampano a video i nomi delle features selezionate.

L'ultima tecnica usata è la Principal Component Analysis, di seguito il codice:

```
1     df_pca = pd.read_csv("../dataset/dataset.csv")
2     df_pca = cleaning_dataframe(df_pca)
3
4     df_pca_application = df_pca[
5         ['URL_LENGTH', 'NUMBER_SPECIAL_CHARACTERS', 'CHARSET', 'SERVER', '
      CONTENT_LENGTH', 'WHOIS_COUNTRY',
6          'WHOIS_STATEPRO', 'WHOIS_REGDATE', 'WHOIS_UPDATED_DATE']]
7     df_pca_network = df_pca[
8         ['TCP_CONVERSATION_EXCHANGE', 'DIST_REMOTE_TCP_PORT', 'REMOTE_IPS', '
      APP_BYTES', 'SOURCE_APP_PACKETS',
9          'REMOTE_APP_PACKETS', 'SOURCE_APP_BYTES', 'REMOTE_APP_BYTES', '
      APP_PACKETS', 'DNS_QUERY_TIMES']]
10
11    y = df_pca['Type'].astype('int')
12
13    pca_application = PCA(n_components=7)
14    df_pca_application = pd.DataFrame(pca_application.fit_transform(
      df_pca_application))
15
16    pca_network = PCA(n_components=2)
17    df_pca_network = pd.DataFrame(pca_network.fit_transform(df_pca_network))
18
19    smote = SMOTE()
20    X_pca, y_pca = smote.fit_resample(pd.concat([df_pca_application,
      df_pca_network], axis=1), y)
21
22    train_model_data_aggregation("J48", DecisionTreeClassifier(), X_pca, y_pca)
23    # train_model_data_aggregation per gli altri algoritmi
24
25    X_pca_application = X_pca.iloc[:, :7]
26    X_pca_network = X_pca.iloc[:, -2:]
27
28    train_model_and_or_aggregation("J48", DecisionTreeClassifier(),
      X_pca_application, X_pca_network, y_pca)
```

```
29    # train_model_and_or_aggregation per gli altri algoritmi
```

L'implementazione inizia con il leggere il dataset e applicare la funzione *cleaning_dataframe*, per poi dividere le features in due dataframe e su ognuno applicare il PCA. Nelle righe successive si effettua il data balancing con SMOTE e la funzione *pd.concat* per concatenare il dataset delle features del livello rete con quelle del livello applicazione. Quindi si procede con il training per la data-aggregation. Prima di effettuare il training per l'AND e l'OR-aggreagtion si divide nuovamente il dataset in due parti e viene chiamata la funzione *train_model_and_or_aggregation*.

3.3 Differenze tra le due implementazioni

In questa sezione si descrivono le differenze tra le due implementazioni, in particolare le differenze tra il dataset utilizzato da Xu [2014] e il dataset utilizzato dall'implementazione di confronto.

Nell'implementazione di Xu [2014] è stato usato un dataset costruito proprio dall'autore grazie a un crawler. Nell'implementazione di confronto invece è stato usato un dataset presente sul sito di data science kaggle, quindi un dasaset diverso. Di seguito vengono riportate le principali differenze tra i due dataset:

- Il dataset utilizzato da Xu [2014] conta 124 features totali, di cui 105 appartenenti al livello applicazione e 19 al livello rete. Contro le 21 colonne del dataset utilizzato nella seconda implementazione, di cui una contiene la variabile target (colonna "Type") e una non fornisce informazioni utili (colonna "URL"), e quindi rimangono nove features del livello applicazione e dieci del livello rete;
- Il dataset usato per il confronto empirico ha solo 1781 istanze, inoltre è fortemente sbilanciato, infatti conta 1565 istanze etichettate con zero (siti web benigni) e 216 con uno (siti web maligni);
- Inoltre la descrizione del dataset trovato su kaggle non è molto esaustiva. Questo ha impedito un processo di data cleaning più accurato soprattutto riguardo le features del livello rete.

Riguardo le tecniche di feature selection e gli algoritmi utilizzati dall'implementazione di confronto sono gli stessi utilizzati da Xu [2014]. Quindi i modi di aggregare i due vettori associati ad ogni URL sono: data-aggregation, OR-aggregation e AND-aggregation. La XOR-aggregation non è stata implementata perché nel dataset utilizzato per il confronto empirico non è presente l'URL del sito, e quindi risulta impossibile utilizzare l'approccio dinamico. Anche gli algoritmi utilizzati sono gli stessi: Naive Bayes, Logistic Regression, SVM e J48. Così come le tecniche di feature selection: CfsSubsetEvaluation, InformationGain e Principle Component Analysis. L'implementazione di confronto ha seguito la pipeline di Xu [2014], infatti ogni algoritmo è stato addestrato per ogni tipo di aggregazione in combinazione con ogni tipo di feature selection.

CAPITOLO 4

Analisi dei Risultati

All'interno di questo capitolo si discute dei risultati ottenuti da Xu [2014] e dei risultati ottenuti dall'implementazione realizzata per effettuare il confronto empirico. Quindi successivamente si prosegue con il discutere le differenze e le uguaglianze tra i risultati.

4.1 Analisi dei Risultati ottenuti da Xu [2014]

L'addestramento e poi la validazione dei modelli, secondo l'approccio single layer, è avvenuto su due dataset separati. Di seguito si riporta la tabella con i risultati ottenuti.

Feature	Naive Bayes			Logistic Regression			SVM			J48		
selection	Acc	FN	FP	Acc	FN	FP	Acc	FN	FP	Acc	FN	FP
Application-layer												
nessuna	51.26	11.02	59.27	90.55	22.99	5.69	85.65	55.50	3.06	96.39	6.09	2.93
PCA	67.75	9.99	38.47	91.49	20.52	5.16	89.46	30.03	5.18	95.66	9.53	2.89
Subset	77.96	35.31	18.16	86.86	37.89	6.28	84.68	51.67	5.27	93.58	15.07	3.99
InfoGain	71.70	19.67	30.66	84.89	43.85	7.09	83.73	52.07	6.36	94.73	12.14	3.39
Network-layer												
nessuna	51.76	0.79	61.64	90.12	21.53	6.63	86.91	24.44	9.98	95.16	9.12	3.67
PCA	67.76	4.01	40.27	87.45	30.65	7.52	85.85	32.95	9.34	89.90	22.58	6.60
Subset	70.18	0.62	38.03	88.14	25.62	8.06	86.53	25.39	10.18	92.41	14.58	5.65
InfoGain	55.53	0.82	56.80	86.75	29.78	8.64	82.82	40.87	10.56	92.85	15.44	4.85

Tabella 4.1: Risultati single-layer di Xu [2014]

Per l'addestramento avvenuto solo sui dati del livello applicazione ci sono due osservazioni da fare:

- L'algoritmo J48 è più efficace degli altri tre modelli, indipendentemente dalla feature selection. Purtroppo però J48 può presentare una percentuale alta di falsi negativi;
- La feature selection implica una diminuzione dell'accuracy e un incremento dei falsi negativi, anche per l'algoritmo J48. Quindi effettuare la feature selection sui dati del livello applicazione non è una buona scelta.

Per il livello rete ci sono tre osservazioni:

- J48 risulta più efficace degli altri algoritmi, indipendentemente dalla feature selection. In maniera analoga all'osservazione precedente, J48 presenta una percentuale di falsi negativi considerevole.
- Anche in questo caso la feature selection danneggia l'accuracy.
- J48 per il network-layer presenta metriche peggiori, di qualche punto percentuale, rispetto a J48 per l'application-layer

Dall'addestramento e dalla validazione effettuati seguendo l'approccio cross-layer, si sono ottenuti i seguenti risultati:

Layer	Feature selection	Naive Bayes			Logistic Regression			SVM			J48		
		Acc	FN	FP	Acc	FN	FP	Acc	FN	FP	Acc	FN	FP
Cross-layer (data-aggregation)	nessuna	55.24	7.96	55.10	96.86	7.94	1.78	94.56	21.22	1.11	99.17	2.28	0.42
	PCA	72.08	4.12	34.65	97.58	5.74	1.48	96.01	9.33	2.49	98.80	3.00	0.69
	Subset	80.39	1.40	24.72	94.56	13.66	3.12	93.29	15.57	4.24	98.33	4.24	0.94
	InfoGain	73.14	1.34	34.06	90.70	22.26	5.69	88.29	26.56	7.57	97.36	6.05	1.68
Cross-layer (OR-aggregation)	nessuna	40.28	0.16	76.43	91.56	6.11	9.10	88.51	7.85	12.54	97.10	0.05	3.70
	PCA	41.58	0.21	74.70	90.03	7.99	10.52	88.34	19.30	9.91	94.25	1.27	7.01
	Subset	57.66	0.06	54.16	88.49	11.46	11.55	86.95	14.15	12.77	94.26	2.61	6.62
	InfoGain	45.27	0.15	70.05	87.34	12.07	12.85	85.26	18.14	13.80	95.12	1.62	5.79
Cross-layer (AND-aggregation)	nessuna	79.09	8.26	24.50	92.52	33.53	0.20	90.33	44.21	0.14	97.88	9.78	0
	PCA	79.91	12.42	22.35	90.43	43.24	0.19	85.64	66.75	0	94.52	24.99	0.03
	Subset	88.18	17.35	10.24	88.98	49.66	0.30	86.73	60.51	0.20	95.44	20.50	0.11
	InfoGain	83.71	14.26	16.88	87.62	55.77	0.29	84.31	71.17	0.26	95.49	20.68	0.02
Cross-layer (XOR-aggregation)	nessuna	80.86	0.16	24.50	98.51	6.11	0.20	98.18	7.85	0.14	99.98	0.05	0
	PCA	82.55	0.21	22.35	98.10	7.99	0.19	96.05	19.30	0	99.69	1.27	0.03
	Subset	91.99	0.06	10.24	97.27	11.46	0.30	96.75	14.15	0.20	99.34	2.61	0.11
	InfoGain	86.80	0.15	16.88	97.14	12.07	0.29	95.82	18.14	0.26	99.63	1.62	0.02

Tabella 4.2: Risultati cross-layer di Xu [2014]

Dal confronto dei risultati cross e single layer si ottiene che:

- La classificazione fatta con J48 e con l'aggregazione dei due vettori è più efficace di J48 usato con l'approccio single-layer.
- J48 è più efficace degli altri tre algoritmi con o senza feature selection, di conseguenza nei punti successivi si farà riferimento a questo algoritmo.
- J48 addestrato con l'aggregazione OR ha meno falsi negativi rispetto a J48 addestrato con la data-aggregation, però ha meno accuracy e più falsi positivi. Invece la AND-aggregation comporta meno accuracy della data aggregation e alte percentuali di falsi negativi. La XOR-aggregation presenta le metriche migliori.
- Anche con l'approccio cross-layer la feature selection implica un abbassamento dell'accuracy.

4.2 Analisi dei Risultati ottenuti dall'implementazione

I risultati single-layer ottenuti dall'implementazione realizzata per il confronto empirico sono riportati nella seguente tabella:

Feature	Naive Bayes			Logistic Regression			SVM			J48		
selection	Acc	FN	FP	Acc	FN	FP	Acc	FN	FP	Acc	FN	FP
Application-layer												
nessuna	70.22	9.32	50.22	70.47	32.01	27.02	86.74	13.67	12.84	91.11	3.96	13.80
PCA	62.90	41.59	32.58	57.82	51.24	33.09	88.53	11.18	11.75	90.89	5.68	12.52
Subset	69.16	3.06	58.59	70.21	34.82	25.17	77.47	18.01	27.02	92.93	5.87	8.24
InfoGain	71.94	5.62	50.47	62.42	44.98	30.15	83.25	15.84	17.63	93.01	4.47	9.52
Network-layer												
nessuna	65.62	48.30	20.44	59.96	22.04	58.01	68.94	43.51	18.59	94.98	4.28	5.75
PCA	56.61	4.21	82.55	59.68	22.36	58.27	69.42	42.87	18.27	94.21	4.79	6.77
Subset	85.14	7.34	22.36	62.65	11.43	63.25	70.95	41.08	16.99	93.54	5.68	7.22
InfoGain	85.01	7.09	22.87	60.63	15.46	63.25	70.92	39.87	18.27	94.44	4.85	6.26

Tabella 4.3: Risultati single-layer

Analizzando i risultati ottenuti dall'application-layer si ottiene che:

- L'algoritmo J48 presenta un accuracy più alta, ma soprattutto falsi negativi e falsi positivi molto inferiori rispetto agli altri algoritmi;

- Bisogna sottolineare che le metriche riportate nella colonna "Naive Bayes" sono state ottenute dall'addestramento dell'algoritmo GaussianNB, che è risultato migliore rispetto le altre tipologie di Naive Bayes. Nonostante questo i falsi positivi sono comunque alti, e nel caso del PCA risultano alti anche i falsi negativi;

- L'algoritmo J48, con la CfsSubsetEval e l'InfoGainAttributeEval, rivela due punti percentuali in più di accuracy e qualche punto percentuale in meno di falsi positivi e falsi negativi, rispetto al PCA e al non applicare nessuna tecnica di feature selection.

Analizzando i risultati ottenuti dal network-layer si ottiene che:

- Anche in questo caso l'algoritmo con migliore accuracy e falsi positivi e falsi negativi minori è il J48.

- Le metriche riportate nella prima colonna sono le metriche ottenute dal ComplementNB, tranne per il PCA. Infatti per quest'ultimo non è stato possibile testare altre tipologie oltre il GaussianNB, a causa dei valori negativi che può generare la tecnica PCA. Infatti si può osservare che il ComplementNB implica metriche migliori rispetto al GaussianNB.

- L'algoritmo J48 per il network-layer presenta metriche migliori, di qualche punto percentuale, rispetto all'addestramento application-layer.

I risultati cross-layer ottenuti dall'implementazione realizzata per il confronto empirico sono riportati nella seguente tabella:

Layer	Feature selection	Naive Bayes			Logistic Regression			SVM			J48		
		Acc	FN	FP	Acc	FN	FP	Acc	FN	FP	Acc	FN	FP
Cross-layer (data-aggregation)	nessuna	66.90	2.85	61.36	75.55	22.93	25.70	89.64	9.16	10.73	94.33	2.93	7.40
	PCA	68.72	29.13	33.36	66.86	24.98	27.28	85.62	15.13	13.61	93.80	3.83	8.56
	Subset	73.73	2.68	49.84	71.92	32.88	23.15	74.95	15.05	31.20	95.75	3.64	4.85
	InfoGain	67.18	6.26	59.36	67.12	44.85	20.89	77.09	12.84	32.97	94.12	4.74	7.70
Cross-layer (OR-aggregation)	nessuna	76.32	5.40	42.04	61.78	6.07	70.35	63.43	0.76	72.90	87.79	0.76	23.64
	PCA	58.51	8.43	74.37	56.73	12.41	73.11	56.66	1.06	86.07	89.88	1.27	18.05
	Subset	70.79	4.28	53.72	63.67	3.51	69.00	56.26	0	87.47	92.74	1.21	13.29
	InfoGain	65.65	1.40	67.28	58.30	10.78	70.95	52.52	0	94.95	91.15	1.21	16.48
Cross-layer (AND-aggregation)	nessuna	75.44	12.28	33.41	67.14	50.46	1.56	90.89	11.56	8.64	95.34	8.07	1.33
	PCA	74.28	35.14	16.29	67.95	47.36	18.92	82.01	23.06	12.90	91.66	15.52	1.15
	Subset	76.51	5.55	40.50	73.16	39.18	15.39	78.32	14.76	26.58	92.23	14.82	0.44
	InfoGain	70.06	8.56	51.30	68.88	44.18	16.03	79.04	12.90	29.00	91.85	15.52	0.76

Tabella 4.4: Risultati cross-layer

Dal confronto dei risultati cross e single layer, e dall'analisi dei risultati cross-layer di ottiene che:

- J48 presenta le metriche migliori rispetto agli altri algoritmi, in tutti i casi.
- J48 con la data-aggregation presenta le metriche migliori rispetto all'AND-aggregation e OR-aggregation, molto simili a J48 network-layer.
- La OR-aggregation per J48 implica un aumento dei falsi positivi, invece, l'AND-aggregation implica un auemnto dei falsi positivi.
- Riguardo al Naive Bayes, le metriche riportate nella riga del PCA sono state ottenute dal GaussianNB, siccome non è stato possibile testare altre tipologie per il problema descritto precedentemente. Per le altre tipologie di feature selection risulta che il GaussianNB ottiene metriche migliori con la data-aggregation e l'AND-aggregation. Invece per la OR-aggregation risulta migliore il ComplementNB. Nonostante questo, il Naive Bayes presenta falsi positivi alti.

4.3 Confronto dei Risultati

Questo capitolo ha lo scopo di riportare le uguaglianze e le differenze tra i risultati descritti nei due capitoli precedenti.

La prima cosa che si può notare è che in entrambi i risultati il miglior algoritmo, sia in termini di cross-layer che single-layer, è l'albero decisionale J48. Infatti anche se con alcuni tipi di aggregazione presenta falsi positivi e falsi negativi considerevoli, come ad esempio si può vedere nella tabella 4.2 per la AND-aggregation, rimane comunque l'algoritmo con le migliori metriche.

Un'ulteriore uguaglianza tra i risultati di J48 si presenta con l'AND-aggregation infatti, come si può osservare dalla tabella 4.2 e dalla 4.4, con questo tipo di aggregazione si ottiene un incremento dei falsi negativi. Nei risultati di Xu [2014] i falsi negativi subiscono un incremento di circa venti punti percentuali, nei risultati di confronto di circa quindici punti percentuali. Cosa che non è vera per l'OR-aggregation, siccome nei primi risultati i falsi positivi e negativi rimangono stabili, invece nei risultati di confronto si nota un incrementi dei falsi positivi.

L'ottimalità di J48, rispetto agli altri algoritmi, potrebbe essere giustificata dal fatto che, costruendo i rami dell'albero, riesce meglio ad apprendere i pattern nascosti nei dati. Cosa

che magari non riescono a fare gli algoritmi probabilistici, come il Naive Bayes, o algoritmi che cercano di dividere le istanze con degli iperpiani, come SVM.

Infatti osservando i risultati di Naive Bayes, si osserva una percentuale di falsi positivi alta, in alcuni casi superiore al 70%. E in altri casi, una accuracy che scende fino al 40%. Come si può osservare dalla seguente tabella, questi risultati valgono per entrambe le implementazioni.

Layer	Feature selection	Naive Bayes (imp. di Xu [2014])			Naive Bayes (imp. di confronto)		
		Acc	FN	FP	Acc	FN	FP
Cross-layer (data-aggregation)	nessuna	55.24	7.96	55.10	66.90	2.85	61.36
	PCA	72.08	4.12	34.65	68.72	29.13	33.36
	Subset	80.39	1.40	24.72	73.73	2.68	49.84
	InfoGain	73.14	1.34	34.06	67.18	6.26	59.36
Cross-layer (OR-aggregation)	nessuna	40.28	0.16	76.43	76.32	5.40	42.04
	PCA	41.58	0.21	74.70	58.51	8.43	74.37
	Subset	57.66	0.06	54.16	70.79	4.28	53.72
	InfoGain	45.27	0.15	70.05	65.65	1.40	67.28
Cross-layer (AND-aggregation)	nessuna	79.09	8.26	24.50	75.44	12.28	33.41
	PCA	79.91	12.42	22.35	74.28	35.14	16.29
	Subset	88.18	17.35	10.24	76.51	5.55	40.50
	InfoGain	83.71	14.26	16.88	70.06	8.56	51.30
Single-layer (Application-layer)	nessuna	51.26	11.02	59.27	70.22	9.32	50.22
	PCA	67.75	9.99	38.47	62.90	41.59	32.58
	Subset	77.96	35.31	18.16	69.16	3.06	58.59
	InfoGain	71.70	19.67	30.66	71.94	5.62	50.47
Single-layer (Network-layer)	nessuna	51.76	0.79	61.64	65.62	48.30	20.44
	PCA	67.76	4.01	40.27	56.61	4.21	82.55
	Subset	70.18	0.62	38.03	85.14	7.34	22.36
	InfoGain	55.53	0.82	56.80	85.01	7.09	22.87

Tabella 4.5: Tabella di confronto per i risultati di Naive Bayes

Anche se le metriche ottenute da Xu [2014], riguardo gli algoritmi SVM e Logistic Regression, non sono ottimali, le metriche ottenute dagli stessi algoritmi dell'implementazione di confronto sono peggiori.

Questo potrebbe essere dovuto al dataset utilizzato. Infatti oltre alle notevoli differenze tra il dataset di Xu [2014] e il dataset dell'implementazione di confronto, descritte nel capitolo 3.3, osservando le istanze classificate in modo errato da SVM con l'OR-aggregation, risulta che il 93% degli errori è dovuto alla classificazione errata del vettore delle features di livello rete. La percentuale scende al 63% per l'algoritmo Logistic regression.

CAPITOLO 5

Conclusioni e Sviluppi Futuri

Dal lavoro svolto e dai risultati analizzati, risulta che l'idea di utilizzare i dati ricavati dal livello applicazione, dal livello rete e da servizi come DNS e Whois, porta sicuramente allo sviluppo di una tecnica in grado di classificare i siti web in modo molto accurato. Infatti, la presenza di più fonti da cui estrarre dati, aumenta la probabilità di classificare correttamente anche i siti malevoli progettati in modo tale da condurre in errore le tecniche che utilizzano solo l'URL, solo i contenuti della pagina web o solo il livello applicazione in generale. Questo però, stando ai dati ottenuti, si verifica solo usando l'algoritmo J48, inoltre sfruttando la XOR-aggregation, quindi combinando l'approccio dinamico con il cross-layer, si ottengono falsi positivi e falsi negativi quasi nulli.

Purtroppo l'implementazione di confronto non ha potuto verificare i risultati della XOR-aggregation, perché per usare l'approccio dinamico bisogna conoscere l'URL sito web, che non è presente nel dataset usato. Infatti le maggiori difficoltà riscontrate durante l'implementazione di confronto sono dovute al dataset: siccome risulta molto sbilanciato, non ha molte istanze, non è presente una descrizione esaustiva delle features e soprattutto è l'unico dataset presente nei maggiori siti web di data science ad avere i dati del livello rete e applicazione.

Di conseguenza, per gli sviluppi futuri, potrebbe essere interessante sviluppare un sistema in grado di costruire altri dataset che tengano in considerazione i due livelli, e quindi, oltre a metterli a completa disposizione della comunità scientifica, si potrebbe ripetere il lavoro svolto usando altri algoritmi e tecniche. Con lo scopo di verificare o contestare i risultati descritti nei capitoli precedenti.

Ringraziamenti

Ringrazio il professore Fabio Palomba che mi ha guidato con grande professionalità e dedizione nello sviluppo di questa tesi, senza mai farmi mancare il suo sostegno e i suoi preziosi consigli.

Ringrazio mio padre Giuseppe, mia madre Rosetta e mio fratello Stefano per avermi supportato.

Desidero ringraziare tutti i miei amici che sono stati sempre al mio fianco. In particolare Alessandro Aquino, Catello Staiano, Eduardo Scarpa, Umberto Della Monica, Antonio Romano, Carmine Leo, Alfredo Cannavaro e Silvio Venturino.

Bibliografia

Davide Canali, Marco Cova, Giovanni Vigna, and Christopher Kruegel. Prophiler: a fast filter for the large-scale detection of malicious web pages. In *Proceedings of the 20th international conference on World wide web*, pages 197–206, 2011. (Citato a pagina 5)

Cho Do Xuan, Hoa Dinh Nguyen, Tisenko Victor Nikolaevich, et al. Malicious url detection based on machine learning. *International Journal of Advanced Computer Science and Applications*, 11(1), 2020. (Citato alle pagine v, 8, 9 e 10)

Vikramaditya Jakkula. Tutorial on support vector machine (svm). *School of EECS, Washington State University*, 37(2.5):3, 2006. (Citato a pagina 7)

Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. Beyond blacklists: learning to detect malicious web sites from suspicious urls. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1245–1254, 2009. (Citato a pagina 5)

Rami M Mohammad, Fadi Thabtah, and Lee McCluskey. Tutorial and critical analysis of phishing websites methods. *Computer Science Review*, 17:1–24, 2015. (Citato a pagina 5)

Moheeb Rajab, Lucas Ballard, Nav Jagpal, Panayiotis Mavrommatis, Daisuke Nojiri, Niels Provos, and Ludwig Schmidt. Trends in circumventing web-malware detection. *Google, Google Technical Report*, 2011. (Citato a pagina 5)

Routhu Srinivasa Rao, Tatti Vaishnavi, and Alwyn Roshan Pais. Catchphish: detection of phishing websites by inspecting urls. *Journal of Ambient Intelligence and Humanized Computing*, 11(2):813–825, 2020. (Citato a pagina 10)

Toshiki Shibahara, Yuta Takata, Mitsuaki Akiyama, Takeshi Yagi, and Takeshi Yada. Detecting malicious websites by integrating malicious, benign, and compromised redirection subgraph similarities. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 655–664. IEEE, 2017. (Citato alle pagine v, 9, 10 e 12)

Li Xu. *Detecting and characterizing malicious websites*. The University of Texas at San Antonio, 2014. (Citato alle pagine i, iii, iv, v, 2, 3, 13, 14, 15, 16, 17, 21, 24, 32, 33, 34, 37, 38 e 39)

Wen Zhang, Yu-Xin Ding, Yan Tang, and Bin Zhao. Malicious web page detection based on on-line learning algorithm. In *2011 International Conference on Machine Learning and Cybernetics*, volume 4, pages 1914–1919. IEEE, 2011. (Citato alle pagine v, 8, 9 e 10)