Università degli studi di Modena e Reggio Emilia

Dipartimento di Ingegneria

Corso di Laurea Magistrale in Ingegneria Informatica

Adversarial Machine Learning per il Rilevamento di Botnet

Relatore:
Prof. Michele Colajanni

Candidato:
Alessandro Aleotti

Correlatore:
Ing. Mirco Marchetti

Anno Accademico 2017/2018

Indice

Todo list

Capitolo 1

Introduzione

In questo capitolo si propongono degli esempi per gli oggetti utilizzati più di frequente in latex: la Sezione 1.1 descrive come scrivere citazioni, la Sezione 1.2 propone degli esempi di oggetti float, la Sezione 1.3 descrive come compilare questo documento.

1.1 Citazioni

Inserisco qualche citazione per mostrare la bibliografia. Per gli articoli accademici è quasi sempre possibile reperire i blocchi da inserire nel file bib da scholar, come ad esempio. Scholar in questo caso è una risorsa/sito online e per questo. Precediamo le citazione da uno spazio indivisibile tramite il carattere ~.

1.2 Oggetti float

Nella Sezione 1.2.1 si propone un esempio di figura float, mentre nella Sezione 1.2.2 si propone un esempio di tabella float.

1.2.1 Figure

La Figura 1.1 è un esempio di figura float.

Figura 1.1: Esempio di figura float in latex.

1.2.2 Tabelle

La Tabella 1.1 è un esempio di tabella.

allineamento centrale	allineamento a sinistra	allineamento a destra
centrale	sinistra	destra

Tabella 1.1: Esempio di tabella float in latex.

1.3 Compilazione

Di seguito il codice da utilizzare per generare il pdf:

```
$ pdflatex main.tex
$ bibtex main.aux
$ pdflatex main.tex
$ pdflatex main.tex
```

Capitolo 2

Stato dell'arte

In questo capitolo si propongono degli esempi per gli oggetti utilizzati più di frequente in latex: la Sezione 1.1 descrive come scrivere citazioni, la Sezione 1.2 propone degli esempi di oggetti float, la Sezione 1.3 descrive come compilare questo documento.

Capitolo 3

Progetto

In questo capitolo si propone il progetto realizzato per raggiungere gli obiettivi preposti: si è partiti dalla realizzazione di un classificatore basato su *Random Forest* per poi passare ad una versione più elaborata, utilizzando una rete neurale. Il passo successivo ha riguardato la creazione di una *Generative Adversarial Network* a partire da un *Autoencoder*.

3.1 Classificatore Random Forest

La prima fase di questo studio è stata quella di implementare un classificatore in grado di separare efficacemente domini *DGA* da domini non malevoli basandosi unicamente sulle caratteristiche linguistiche dei domini: infatti, ad un esame preliminare, i domini *DGA* presentano caratteristiche ben differenti da semplici frasi o parole che solitamente compongono i domini reali.

Si è scelto di utilizzare Random Forest in quanto ritenuto il più adatto al caso in esame. L'algoritmo è stato inoltre messo a confronto con *Support Vector Machine* e *Naive-Bayes*.

All'interno del classificatore *Random Forest* [**?**], ogni albero dell'insieme è costruito a partire da un campione estratto con sostituzione dal *training set*. In aggiunta, al momento della divisione del nodo durante la costruzione di un albero, la divisione scelta non è più la migliore soluzione tra tutte le *features*. Al suo posto, la divisone che

indicare SVM e GNB solo nei risultati?

viene scelta è la migliore divisione all'interno di un *subset* casuale tra tutte le *features*. Come risultato di questa casualità, il *bias* della foresta di solito aumenta leggermente (rispetto al *bias* di un singolo albero non casuale) ma, a causa della media, la sua varianza diminuisce, di solito compensando l'aumento di *bias*, quindi dando un modello generale migliore.

3.1.1 Dataset

I *dataset* di *training* e *testing* sono stati ricavati due fonti differenti: per quel che riguarda i domini reali si è fatto riferimento alla classifica dei domini più visitati al mondo fornita da *Alexa Internet Inc.* [1] , per un totale di 1 milione di siti realmente esistenti; mentre grazie al repository fornito da [2] è stato possibile ottenere un *dataset* esaustivo di esempi *DGA* da diverse famiglie di *malware*.

mostrare esempio diversi DGA

A partire da tale *dataset* combinato si è proceduto alla creazione di un classificatore binario che fosse in grado di distinguere domini reali da domini generati algoritmicamente.

Il passo seguente stato creare una serie di *features* che fossero in grado di descrivere le caratteristiche linguistiche dei domini presi in esame.

descrivere distribuzione caratteri Alexa vs dga

Per raggiungere tale obiettivo si è fatto riferimento a ricerche già esistenti [3] [4] [5] [6]. Di seguito viene illustrato l'insieme di tali *features*:

3.1.2 Features

- **Rapporto tra caratteri significativi**. Modella il rapporto dei caratteri della stringa p che formano una parola significativa all'interno del dizionario Inglese. Un valore basso indica la presenza di algoritmi automatici. In dettaglio, si divide p in n sotto-parole significative w_i di almeno 3 caratteri: $|wi| \geq 3$ cercando di lasciare fuori meno caratteri possibili:

$$R(d) = R(p) = \frac{max(\sum_{i=1}^{n} |wi|)}{|p|}$$

Se $p = \text{facebook}$, $R(p) = \frac{(|\text{face}|+|\text{book}|)}{8} = 1$ allora il dominio è composto completamente da parole significative, mentre $p = \text{pub03str}$, $R(p) = \frac{|\text{pub}|}{8} = 0.375$.

- **Punteggio di normalità degli n-grammi**: Questa classe di *features* modella la pronunciabilità di un nome di dominio rispetto la lingua Inglese. Più la combinazione di fonemi del dominio è presente all'interno del Dizionario Inglese più tale dominio è pronunciabile. Domini con un basso numero di tali combinazioni sono probabilmente generati algoritmicamente. Il calcolo avviene estraendo lo n-gramma di p di lunghezza $n \in \{1, 2, 3\}$ e contando il numero di occorrenze di tale n-gramma all'interno del Dizionario Inglese. Tali *features* sono quindi parametriche rispetto ad n:

$$S_n(d) = S_n(p) \ = \frac{\sum_{\text{n-gramma t in p}} count(t)}{|p| - n + 1}$$

dove $count(t)$ sono le occorrenze dello n-gramma nel dizionario. Ad esempio $S_2(facebook) = fa_{109} + ac_{343} + ce_{438} + eb_{29} + bo_{118} + oo_{114} + ok_{45} = 170.8$

- **Rapporto tra caratteri numerici** Questa *feature* rappresenta il rapporto tra i caratteri numerici presenti all'interno del nome di dominio rispetto la lunghezza totale della parola. Molte famiglie di *malware* utilizzano *DGA* che generano domini tramite una distribuzione uniforme di caratteri alfabetici minuscoli e numeri, questo porta a domini generati algoritmicamente che presentano una maggior presenza di numeri al loro interno rispetto ai domini reali.

- **Rapporto tra vocali e consonanti** Questa *feature* modella il rapporto tra vocali e consonanti all'interno del nome di dominio.

- **Lunghezza del nome di dominio** Questa *feature* calcola la lunghezza del dominio. Molte famiglie di *malware* utilizzano *DGA* che generano domini di lunghezza costante, generalmente molto lunghi rispetto ai domini reali.

L'implementazione di tali *features* ha permesso di ottenere un *dataset* in grado di modellare le caratteristiche linguistiche dei nomi di dominio mostrati al capitolo 3.1.1. Da tale spunto è partita la fase iniziale di *testing*

3.1.3 Output

L'obiettivo di tale classificatore è quello di riuscire a separare in maniera efficace i domini reali da quelli generati algoritmicamente. A tale proposito ??? Durante la fase di sperimentazione il classificatore si è rivelato efficace rispetto la maggior parte delle famiglie di *DGA*. Il caso particolare della famiglia *suppobox* [7] ha messo in particolare difficoltà il classificatore in quanto tale algoritmo genera domini in maniera pseudo-casuale, concatenando due parole a partire da un *subset* del dizionario inglese di 384 parole. Tale caratteristica fa si che le *features* linguistiche estratte da questa famiglia di *malware* siano molto simili a quelle presenti nei domini reali.

ampliare questa sezione e chiedere a Marchetti

A partire da questo risultato si scelto di procedere con la progettazione di un classificatore neurale in grado di superare tale problematica.

indicare il capitolo coi risultati di Random Forest

3.2 Classificatore Neurale

Questo classificatore neurale nasce con l'intento di superare le difficoltà incontrate dal precedente classificatore basato su *Random Forest*, utilizzando le caratteristiche delle reti neurali, in grado di estrarre *features* a partire dai dati grezzi. Si è scelto di partire dall'architettura di tipo *Multilayer Perceptron* con l'obiettivo di ottenere risultati migliori rispetto al caso mostrato nella sezione precedente.

I passi del progetto sono stati la codificazione dei domini in valori numerici, l'individuazione di una architettura ottimale per classificare i dati in esame ed un'ultima fase di *tuning* degli iperparametri della rete neurale.

mostrare esempio suppobox

3.2.1 Input

A partire dal *dataset* creato per il precedente caso, si è deciso di convertire direttamente i nomi di dominio alfanumerici in vettori numerici, mappati secondo il dizionario di tutti i caratteri ammessi [8] (lettere minuscole a-z, numeri 0-9, tratto d'unione "-").

??????????

L'obiettivo è quello di fornire al classificatore neurale in questione una rappresentazione il più possibile aderente ai dati reali, senza l'ausilio di *features* ingegnerizzate a priori, lasciando così la libertà alla rete neurale di estrarre le caratteristiche più appropriate per la distinzione dei domini. Come scelta progettuale si è deciso di limitare la dimensione dei domini a 15 caratteri per ognuno, in modo da ottenere un *dataset* di dimensioni fissate e sopperire alle differenti lunghezze di ogni dominio tramite un semplice *padding* di zeri all'inizio di ogni stringa codificata.

Assieme ai dati codificati è stato generato un vettore di *target* nel quale viene indicato da 0 o da 1 se il dominio in esame è di tipo reale o generato algoritmicamente. L'obiettivo quindi è di attuare un classificatore binario in grado di prevedere correttamente a quale categoria appartiene un dominio esaminato

3.2.2 Composizione Interna

L'architettura scelta in prima fase è stata quella del *Multilayer Perceptron (abbr. MLP)*, una tipologia di rete neurale *feedforward* tipicamente formata da almeno tre livelli di nodi. Ad esclusione del livello di *input* i livelli del MLP utilizzano funzioni di attivazione non lineari che permettono di eseguire distinzioni tra dati non linearmente separabili. Considerando una rete formata da m neuroni, se si considera d come numero di input, si avrà il seguente output

$$y_j = y\left(\sum_{i=0}^{d} w_{ji}x_i\right)$$

nel quale x_i sono gli input e w_{ji} sono i pesi di ogni input combinati con ogni output.

aggiungere ed estendere la teoria MLP nel capitolo

Nel caso in esame è stata utilizzata per i livelli interni la funzione di attivazione *Rectifier Linear Unit* (ReLU) [9] definita dalla funzione

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

dove x rappresenta l'*input* del neurone. I vantaggi di tale funzione sono una migliorata *performance* rispetto ad altre funzioni similari come *tanh* e *sigmoid* per quel che riguarda la convergenza della discesa stocastica del gradiente.

Per quel che riguarda la funzione di attivazione del livello di *output* si è scelta la funzione *sigmoidea*, definita dalla formula

$$P(t) = \frac{1}{1 + e^{-t}}$$

inserire grafo sigmoid

La struttura del *MLP* in esame è stata raggiunta dopo una serie di test pratici in cui si sono messi a confronto tre modelli differenti di per numero di neuroni all'interno degli *hidden layer*:

- un modello ridotto composto da un layer di input con un numero di neuroni pari alla dimensione delle stringhe codificate, un layer intermedio di dimensione dimezzata rispetto al precedente ed il layer finale di uscita di dimensione 1 per attuare la classificazione binaria, oggetto di studio.
- un modello allargato composto da un layer di input con un numero di neuroni pari alla dimensione delle stringhe codificate, due layer intermedi di dimensioni moltiplicate di diversi ordini rispetto al layer iniziale ed un layer finale di dimensione 1.
- un modello intermedio composto da un layer di input con un numero di neuroni pari alla dimensione delle stringhe codificate, un layer intermedio di dimensione 128, un layer di dimensione minore a 64 ed un layer finale di dimensione 1.

I tre modelli messi a confronto hanno mostrato risultati simili, tuttavia il modello intermedio si è dimostrato più performante, con un costo computazionale irrisorio rispetto al modello allargato.

3.1

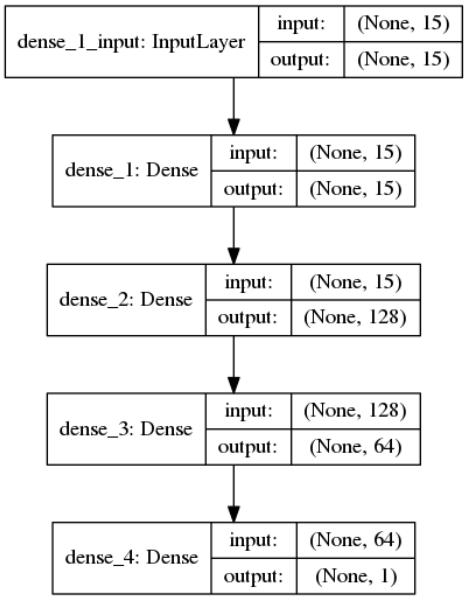


Figura 3.1: Grafico del modello intermedio. Escluso il primo layer di input, si notino gli *hidden layer* dense2 e dense3 di dimensioni rispettivamente 128 e 64

3.2.3 Output

3.3 Realizzazione Adversarial Learning

3.3.1 Input

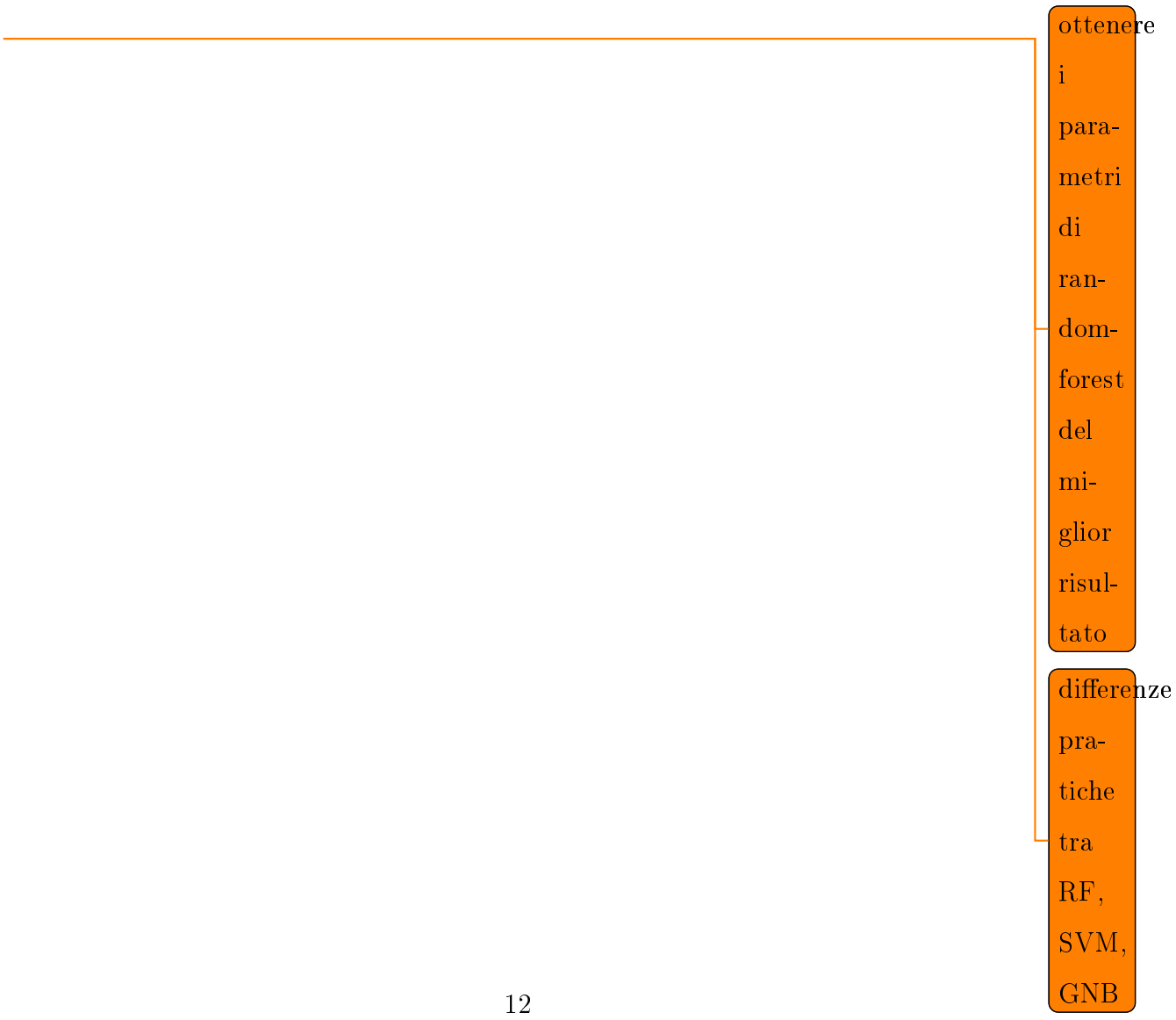
3.3.2 Composizione Interna

3.3.3 Output

Capitolo 4

Implementazione

4.1 Classificatore Random Forest



4.1.1 Input

4.1.2 Composizione Interna

4.1.3 Output

4.2 Classificatore Neurale

4.2.1 Input

4.2.2 Composizione Interna

4.2.3 Output

4.3 Realizzazione Adversarial Learning

4.3.1 Input

4.3.2 Composizione Interna

4.3.3 Output

Capitolo 5

Risultati

In questo capitolo si propongono degli esempi per gli oggetti utilizzati più di frequente in latex: la Sezione 1.1 descrive come scrivere citazioni, la Sezione 1.2 propone degli esempi di oggetti float, la Sezione 1.3 descrive come compilare questo documento.

Capitolo 6

Conclusioni

In questo capitolo si propongono degli esempi per gli oggetti utilizzati più di frequente in latex: la Sezione 1.1 descrive come scrivere citazioni, la Sezione 1.2 propone degli esempi di oggetti float, la Sezione 1.3 descrive come compilare questo documento.

Bibliografia

[1] Amazon, "Alexa." `https://www.alexa.com/`, visited in Sep. 2017.

[2] A. Abakumov, "Dga." `https://github.com/andrewaeva/DGA`, visited in Sep. 2017.

[3] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon, "From throw-away traffic to bots: Detecting the rise of dga-based malware," in *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, (Bellevue, WA), pp. 491–506, USENIX, 2012.

[4] S. Yadav, A. K. K. Reddy, A. N. Reddy, and S. Ranjan, "Detecting algorithmically generated malicious domain names," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, (New York, NY, USA), pp. 48–61, ACM, 2010.

[5] S. Yadav, A. K. K. Reddy, A. L. N. Reddy, and S. Ranjan, "Detecting algorithmically generated domain-flux attacks with dns traffic analysis," *IEEE/ACM Trans. Netw.*, vol. 20, pp. 1663–1677, Oct. 2012.

[6] S. Schiavoni, F. Maggi, L. Cavallaro, and S. Zanero, *Phoenix: DGA-Based Botnet Tracking and Intelligence*, pp. 192–211. Cham: Springer International Publishing, 2014.

[7] J. Geffner, "End-to-end analysis of a domain generating algorithm malware family," *Black Hat USA*, vol. 2013, 2013.

[8] ICANN. `https://www.icann.org/`.

[9] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (G. Gordon, D. Dunson, and M. Dudík, eds.), vol. 15 of *Proceedings of Machine Learning Research*, (Fort Lauderdale, FL, USA), pp. 315–323, PMLR, 11–13 Apr 2011.