Alessio Rocchi 7014405

UNIVERSITÀ DEGLI STUDI FIRENZE

Relazione dell'elaborato di Intelligenza Artificiale

Testo dell'elaborato: "Intrusion Detection"

In questo esercizio si utilizzano implementazioni disponibili di Decision Tree e Random Forest (p.es. scikit-learn in Python o Weka in Java) al problema dell'intrusion detection.

Prefazione:

Nell'elaborato si utilizzano i due data frame: "kddcup.data" e "kddcup.data_10_percent".

Questi data frame sono stati utilizzati nel "The Third International Knowledge Discovery and Data Mining Tools Competition" che è stata tenuta in combinazione con il KDD99 "The Fifth International Conference on Knowledge Discovery and Data Mining".

La competizione consisteva nel costruire un modello predittivo che potesse distinguere fra "buone" connessioni e "cattive" connessioni.

Svolgimento:

Durante lo svolgimento di questo elaborato è stata usato la libreria *scikit-learn* in Python.

Il primo passo per addestrare il nostro modello predittivo di tipo Random Forest è di importare i due data frame, grazie alla funzione *read_cvs* della libreria *pandas.*

Essendo in formato unlabeled, sono state rinominate le colonne, che con i nomi corrispondono alle caratteristiche di ciascuna connessione.

Esempio di stampa delle prime 5 righe del data frame:

	duration	protocol_type	service	flag	src_bytes	dest_bytes	land	wrong_fragment	urgent	hot	...
0	0	tcp	http	SF	239	486	0	0	0	0	...
1	0	tcp	http	SF	235	1337	0	0	0	0	...
2	0	tcp	http	SF	219	1337	0	0	0	0	...
3	0	tcp	http	SF	217	2032	0	0	0	0	...
4	0	tcp	http	SF	217	2032	0	0	0	0	...

1) Preprocessing:

Prima di addestrare il modello è stato fatto preprocessing sui dati per trasformare le variabili categoriche in qualcosa che il nostro modello Random Forest possa classificare. Infatti, le variabili presenti nelle colonne "protocol_type", "service", "flag" e "conn_type" non sono classificabili.

È stato diviso il data frame in due parti:

- X: che contiene tutte i features delle connessioni, apparte il loro tipo contenuto in "conn_type"
- Y: che contiene solo la colonna "conn_type"

X viene processato usando la funzione *make_column_transformer* in congiunzione con il *OneHotEncoder*:

Mentre il *make_column_transformer* si occupa di creare l'oggetto *column_transformer* che grazie alla funzione *fit_trasform* trasformerà le colonne indicate con il metodo del *OneHotEncoder*.

Alla fine di questo processo ogni variabile categorica sarà codificata in un array del tipo:

V = <0,0... 0,1,0... 0,0>

Questo metodo è usato per far sì che le variabili categoriche siano più lontane possibili l'una dall'altra in modo che il Random Forest possa classificarle meglio.

Per la Y è stato usato un metodo più semplice dato che non ci serve che i dati siano il più distanti possibili per migliorare l'efficienza. È stato quindi usato il *LabelEncoder*, un oggetto che fa lo stesso lavoro del *OneHotEncoder*, ma invece di creare un vettore ad ogni variabile categorica assegna un valore numerico.

Dopo aver creato il *LabelEncoder* usiamo la sua funzione *fit_transform* su Y.

2) Addestramento Classificatore:

Gli insiemi creati al punto precedente devono essere suddivisi un'altra volta, grazie alla funzione *train_test_split* , per creare:

- X_train: parte del data frame con le *features* delle connessioni che verrà usato per addestrare il classificatore.
- Y_train: parte del data frame che contiene i tipi di connessione corrispondenti alle features in X_train.
- X_test: la parte rimanente di X che ci serve a verificare l'accuratezza del nostro classificatore.
- Y_test: la parte rimanente di Y che ci serve a verificare l'accuratezza del nostro classificatore.

è stato considerato uno split del 10%, tra train e test, per entrambi i data frame.

In seguito alla divisione abbiamo addestrato il nostro classificatore di tipo Random Forest con la sua funzione *fit* con parametri X_train e Y_train.

3) Risultati:

Per verificare l'efficacia del nostro classificatore è stata usata la funzione *score* a cui passiamo X_test e Y_test come parametri in ingresso e ci risponderà con la precisione del modello da noi creato.

Per semplicità consideriamo sempre una foresta con 10 alberi, quindi con 10 stimatori.

Facendo vari test abbiamo visto che la percentuale di accuratezza dell'albero addestrato sul data frame "kddcup99.data_10_percent" si aggira su valori di:

99,98 % ± 0,01 %

Anche cambiando il criterio di suddivisione da “entropy” a “gini” il risultato si aggira sempre sulle stesse cifre.

Mentre per il data frame “kddcup99.data” la percentuale di accuratezza si aggira su valori del:

99,995 % ± 0,005 %

4) Analisi dei risultati:

Il data frame “kddcup99.data” da’ risultati migliori di quelli ottenuti utilizzando il data frame “kddcup.data_10_percent”.

Infatti, nel primo data frame abbiamo 4898430 connessioni il cui 10% sono 489843 connessioni, mentre nel secondo caso le connessioni in tutto il data frame sono solo 489843 il cui 10% è circa 48984 connessioni.

Quindi gli alberi formati sul primo data frame sono più precisi perché hanno a disposizione una maggiore quantità di dati per imparare a riconoscere le connessioni.

Seguendo poi la ricerca:

“*Selecting Features for Intrusion Detection: A Feature Relevance Analysis on KDD 99 Intrusion Detection Datasets*” di H. Güneş Kayacık, A. Nur Zincir-Heywood, Malcolm I. from Heywood Dalhousie University.

Ci accorgiamo del perché l’accuratezza del nostro classificatore è così alta:

Definiamo prima il concetto di **“Information Gain”**:

Sia “S” un set di campioni del training set con le loro etichette corrispondenti. Supponiamo che ci siano m classi e che il training set contenga “s_i” campioni della classe “I” e che “s” sia il numero totale di campioni nel training set. Allora l’informazione per classificare in dato campione si può calcolare come:

$$I(s_1,s_2,\ldots,s_m) = -\sum_{i=1}^{m} \frac{s_i}{s} \log_2\left(\frac{s_i}{s}\right)$$

Sia anche F un feature con valori { *f1, f2, …, fv* } possiamo dividere il training set in v subsets { *S1, S2, …, Sv* } dove Sj è il subset che ha valore fj per il feature F. Inoltre, sia Sj l'insieme che contiene gli s_{ij} campioni della classe I. L'Entropia del feature F si calcola come:

$$E(F) = \sum_{j=1}^{v} \frac{s_{1j} + \ldots + s_{mj}}{s} \times I(s_{1j},\ldots,s_{mj})$$

Allora l’Information Gain si ottiene come:

$$Gain(F) = I(s_1,\ldots,s_m) - E(F)$$

Questo valore indica quanto una determinata caratteristica influisca nella classificazione di un determinato tipo di connessione.

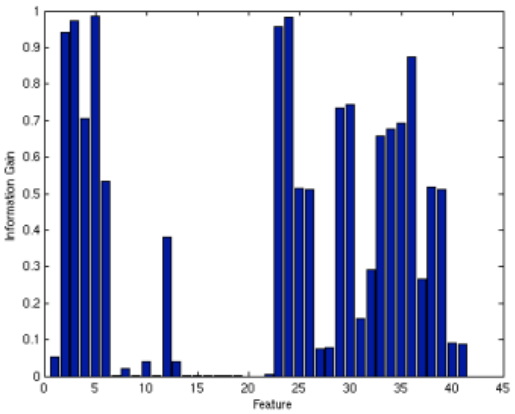
Dai risultati della ricerca svolta da H. Güneş Kayacık, A. Nur Zincir-Heywood, Malcolm I. otteniamo la seguente tabella delle caratteristiche più rilevanti per alcune classi:

Class Label	Relevant Features
normal	1, 6, 12, 15, 16, 17, 18, 19, 31, 32, 37
smurf	2, 3, 5, 23, 24, 27, 28, 36, 40, 41
neptune	4, 25, 26, 29, 30, 33, 34, 35, 38, 39
land	7
teardrop	8
ftp_write	9
back	10, 13
guess_pwd	11
buffer_overflow	14
warezclient	22

Da questa tabella ci accorgiamo come le classi più facili da individuare sono le classi “normal”, “smurf” e “neptune” dato che dipendono da un maggior numero di caratteristiche. Queste tre connessioni costituiscono inoltre circa il 99.07% dei due data frame. Questi due fattori fanno sì che il nostro classificatore abbia una precisione molto elevata sui dati contenuti in X_test.

Sempre basandoci su questa ricerca possiamo ottimizzare i tempi di addestramento del Random Forest e quelli di preprocessing scartando le caratteristiche meno rilevanti:

Infatti, osservando la tabella:



Vediamo come le caratteristiche 20 e 21 sono completamente irrilevanti e quanto poco impatto abbiano i feature 13, 15, 17, 22 e 40 per la classificazione.

Se scartiamo le colonne corrispondenti a quelle caratteristiche dal data frame come parte del preprocessing, otteniamo come media di accuratezza, considerando sempre 10 alberi:

99,995 % ± 0,005 %

Cioè la stessa del classificatore che viene addestrato su tutti i dati.

Così facendo diminuiamo il tempo di preprocessing di un fattore di circa 3,71 e il tempo di addestramento di ogni singolo albero di un fattore X:

- X=1,95 per un solo albero.
- X<1,95 ad aumentare di n dove n è il numero degli alberi.