Tesina di

Intelligent and Secure Network

Corso di Laurea in Ingegneria Informatica e Robotica – A.A. 2022-2023

Dipartimento di Ingegneria

docente

Prof. Mauro Femminella

Preparazione ed analisi di un dataset di traffico TCP da terminali wireless/wired

studenti

Alex Ardelean alexnicolae.ardelean@studenti.unipg.it

Data ultimo aggiornamento: 22 agosto 2023

0. Indice

1. Introduzione

Il progetto consiste nella preparazione ed analisi di un dataset di traffico TCP da terminali wireless/wired.

Per quanto riguarda la preparazione del dataset si è usato il tool TShark e alcuni script python. TShark è un programma a linea di comando open-source utilizzato per l'analisi del traffico di rete. Fa parte della suite di strumenti Wireshark, che è una delle piattaforme più popolari per l'analisi dei protocolli di rete. TShark è un'alternativa a Wireshark che può essere eseguita senza una GUI ed è particolarmente utile quando è necessario eseguire l'analisi di traffico da una shell o da script. Esso è in grado di catturare e analizzare il traffico di rete in tempo reale o lavorare con file di cattura esistenti. In particolare esso permette di usare un ampio numero di filtri per il calcolo delle statistiche specificando durata e intervallo di cattura. Queste funzionalità sono risultate particolarmente utili, dato che le features considerate sono: latenza, numero di ritrasmissioni e banda. Tuttavia, l'output restituito da tshark non è direttamente utilizzabile per l'addestramento di un modello di machine learning, si è perciò fatto di uso di uno script python per ripulire i dati.

Una volta ottenuto il dataset si è seguito un approccio data-driven per ottenere un modello in grado di classificare il traffico nelle due classi di interesse (wired/wireless).

Si sono infine valutate le prestazioni del modello addestrato su traffico catturato in tempo reale.

Si è ipotizzato che, un modello di classificazione del traffico wireless/wired, potrebbe essere utile per la gestione e monitoraggio della rete, ad esempio:

- Ottimizzazione delle prestazioni: comprendere la tipologia di traffico su una rete può aiutare a ottimizzare le prestazioni. Ad esempio, se si rileva un traffi-

co wireless intenso in un'area specifica, potrebbe essere necessario aggiungere più punti di accesso Wi-Fi per gestire il carico.

- Rilevamento di intrusioni: i modelli di classificazione possono contribuire a rilevare attività sospette o intrusioni nella rete. Se un numero insolitamente elevato di dispositivi viene classificato come "wireless" in un'area in cui ci si aspetta principalmente traffico cablato, potrebbe indicare un possibile tentativo di accesso non autorizzato.
- Monitoraggio del traffico di rete: un modello di classificazione può aiutare a tenere traccia del flusso di traffico di rete e a identificare i diversi tipi di connessioni utilizzate. Questo può essere utile per scopi di registrazione e audit.
- Ricerca e analisi di mercato: le aziende che forniscono apparecchiature di rete potrebbero utilizzare un modello di classificazione per raccogliere dati sulla distribuzione del traffico wireless/cablato tra gli utenti finali. Questi dati possono essere utilizzati per analisi di mercato e sviluppo di prodotti.
- Ottimizzazione delle politiche di sicurezza: identificare quale tipo di traffico è predominante in diverse parti della rete può aiutare nella definizione e nell'ottimizzazione delle politiche di sicurezza. Ad esempio, potrebbe essere necessario applicare controlli di sicurezza più rigorosi al traffico wireless rispetto a quello cablato.

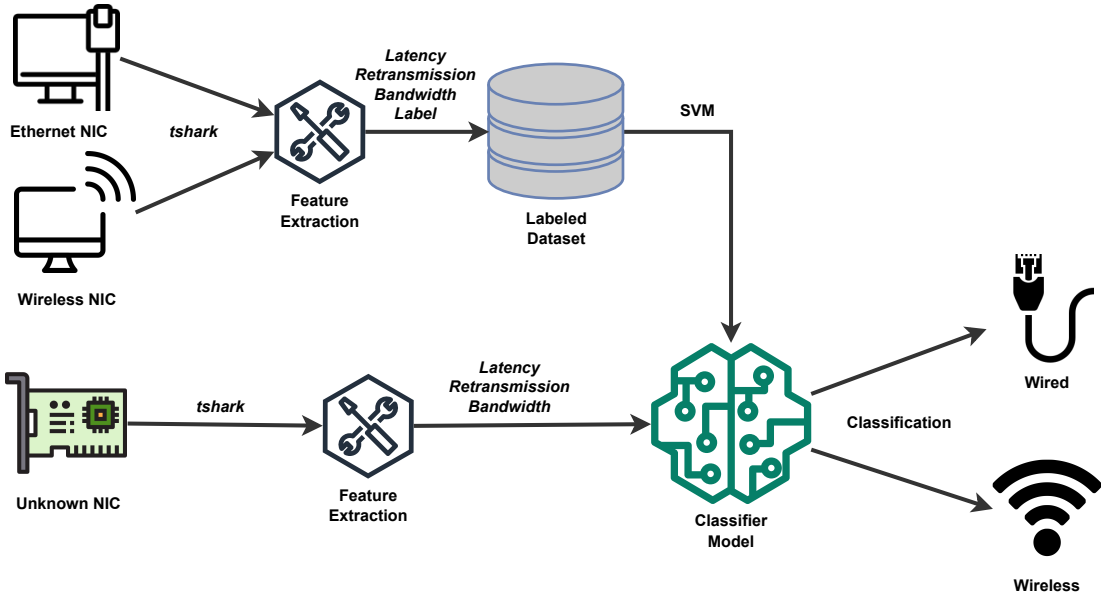

Figura 1.1: Approccio utilizzato

2. Preparazione del dataset

2.1 Descrizione delle features

Le 3 features considerate sono:

- Latenza: ritardo che si verifica tra l'invio di un pacchetto e la sua ricezione. Le reti cablate solitamente offrono latenze più basse rispetto alle reti wireless a causa della loro maggiore affidabilità e velocità di trasmissione. Le reti wireless possono subire ritardi dovuti a interferenze, congestionamenti e altre variabili ambientali. L'analisi della latenza può quindi rivelare differenze significative tra i due tipi di connessioni.
- Numero di ritrasmissioni: si riferiscono al processo in cui dati o informazioni vengono inviati nuovamente da un dispositivo o nodo di rete al mittente a causa di un errore o di una mancata consegna durante la trasmissione iniziale. In sostanza, è un tentativo di correggere o completare la trasmissione fallita. Le reti wireless sono soggette a interferenze e attenuazioni del segnale, il che può portare a una maggiore frequenza di pacchetti persi o danneggiati rispetto alle reti cablate più stabili. Di conseguenza, le reti wireless spesso richiedono più ritrasmissioni di pacchetti per garantire che i dati vengano trasmessi correttamente. Questo comportamento può essere catturato dal numero di ritrasmissioni. Le reti cablate, d'altra parte, tendono a presentare meno ritrasmissioni a causa della maggiore stabilità del segnale.
- Banda: si riferisce alla quantità di dati che possono essere trasmessi attraverso una connessione in un determinato intervallo di tempo. Le reti cablate tendono ad avere una banda più elevata rispetto alle reti wireless. Le differenze nella larghezza di banda possono influenzare la velocità di trasmissione

dei dati e la capacità complessiva della rete. Pertanto, misurare la banda può contribuire a distinguere tra connessioni cablate e wireless.

2.2 Cattura del traffico tramite TShark

Al fine di ottenere un dataset del tipo

```
| latenza | nr. ritrasmissioni | banda | label |
```

si è utilizzato il tool TShark. É infatti possibile utilizzare TShark per realizzare delle sonde software in grado sia di catturare il traffico sia di fornire statistiche su metriche specifiche. Si è fatto uso delle seguenti opzioni:

- `-i <interface>` per specificare l'interfaccia su cui eseguire la cattura;
- `-a duration:<duration>` per specificare la durata della cattura;
- `-q`: per sopprimere la visualizzazione dei pacchetti catturati;
- `-z`: per utilizzare filtri per il calcolo delle statistiche;

In particolare il costrutto

```
-z io,stat,interval[,filter][,filter][,filter]...
```

è risultato particolarmente utile dato che permette di calcolare statistiche sul traffico descritto dati filtri nell'intervallo di tempo `interval`. Le statistiche vengono calcolate su un numero di sottointervalli pari a $\frac{duration}{interval}$.

Per generare statistiche relative alle 3 features di interesse si sono quindi utilizzati i seguenti filtri:

- `AVG(tcp.analysis.ack_rtt)tcp.analysis.ack_rtt`;
- `COUNT(tcp.analysis.retransmission)tcp.analysis.retransmission`;
- `BYTE`;

Per avere una stima affidabile della latenza si è quindi fatto uso dell'RTT. Dato che TCP è un protocollo affidabile che garantisce la consegna dei pacchetti, ogni volta che si trasmette un pacchetto, ne viene salvata una copia nel buffer, e ci resta finché non si riceve l'ACK, dopodiché viene cancellato. Quando un pacchetto viene inviato, il mittente fa partire un timer. Se un pacchetto si perde oppure l'ACK non arriva in tempo, prima che il timer scada, il mittente ritrasmette il pacchetto di nuovo. Il timer è chiamato RTO (Retransmission Time Out) che raddoppia ad ogni ritrasmissione. Per determinare il RTO si utilizza il RTT (Round Trip Time) cioè il

tempo di percorrenza a due vie di un pacchetto per tornare al mittente sotto forma di ACK. Una media dell'RTT garantisce una buona stima della latenza.

Per catturare il traffico si è quindi utilizzato un comando del tipo:

```
tshark -i <interface> -a duration:600 -q -z
"io,stat,1,
AVG(tcp.analysis.ack_rtt)tcp.analysis.ack_rtt,
COUNT(tcp.analysis.retransmission)tcp.analysis.retransmission,
BYTES"
> nomeFile
```

si sono quindi catturate statistiche per una durata di 600 s utilizzando un intervallo di 1 s. Tale comando è stato eseguito indipendentemente con un terminale wireless e con uno wired. In entrambi i casi, prima di avviare la cattura del traffico, si è avviato il download dell'immagine ISO dell'ultima versione LTS di Ubuntu.

Tale comando, in entrambi i casi, ha fornito un output del tipo:

```
================================================================================
| IO Statistics                                                                |
|                                                                              |
| Duration: 600.312506 secs                                                    |
| Interval:   1 secs                                                           |
|                                                                              |
| Col 1: AVG(tcp.analysis.ack_rtt)tcp.analysis.ack_rtt                         |
|     2: COUNT(tcp.analysis.retransmission)tcp.analysis.retransmission         |
|     3: BYTES                                                                 |
|------------------------------------------------------------------------------|
|            |1         |2      |3         |                                  |
| Interval   |    AVG   | COUNT |   BYTES  |                                  |
|------------------------------------------|                                  |
|   0 <>   1 | 0.000019 |     0 | 12526266 |                                  |
|   1 <>   2 | 0.000026 |     0 | 12524238 |                                  |
|   2 <>   3 | 0.000017 |     0 | 12524730 |                                  |
|   3 <>   4 | 0.000017 |     0 | 12524730 |                                  |
|   4 <>   5 | 0.000020 |     0 | 12520524 |                                  |
|   5 <>   6 | 0.000407 |    15 | 12565303 |                                  |
      ...          ...      ...       ...
| 596 <> 597 | 0.000024 |     0 | 12524670 |                                  |
| 597 <> 598 | 0.000016 |     0 | 12523812 |                                  |
| 598 <> 599 | 0.000013 |     0 | 12524730 |                                  |
| 599 <> 600 | 0.000012 |     0 | 12524616 |                                  |
```

```
| 600 <> Dur | 0.000012 |      0 |   3915228 |                                |
=================================================================================
```

che è stato salvato nei file `statisticheWired.txt` e `statisticheWireless.txt`.

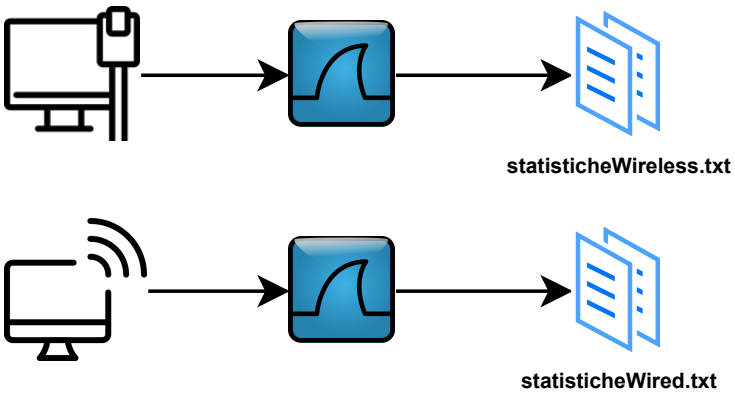

Figura 2.1: Cattura delle statistiche tramite tshark

2.3 Estrazione delle features

Un output del tipo precedentemente visto non è adatto per essere processato da una libreria di machine learning. Per questo motivo si è utilizzato il seguente script python:

```
import sys
import re
import csv

def getDataFromStatistics(filename):
    with open(filename, 'r') as file:
        content = file.read().replace(" ", "")
        content = re.sub('[^A-Za-z0-9|\n<.]+', '', content)
    lines = content.split('\n')

    data = [l for l in lines if "<" in l]
    return data

if __name__ == '__main__':
    arguments = sys.argv[1:]

    wiredStats = arguments[0]
    wirelessStats = arguments[1]
```

```
    wiredData = getDataFromStatistics(wiredStats)
    wirelessData = getDataFromStatistics(wirelessStats)

    with open(f"WirelessWiredData.csv", mode='w', newline='') as
                                        csv_file:
        csv_writer = csv.writer(csv_file)
        csv_writer.writerow(["latency","retransmission","bandwidth
                                          ","label"])

        for l in wiredData:
            linspl = l.split("|")
            [latency, retransmission, bandwidth] = linspl[2],
                                                linspl[3], linspl[4]
            csv_writer.writerow([latency, retransmission,
                                                bandwidth, 'wired'])

        for l in wirelessData:
            linspl = l.split("|")
            [latency, retransmission, bandwidth] = linspl[2],
                                                linspl[3], linspl[4]
            csv_writer.writerow([latency, retransmission,
                                                bandwidth, 'wireless'
                                                ])
```

Tale script prevede il passaggio di due argomenti, corrispondenti ai file di testo precedentemente citati. Prevede quindi la seguente sintassi per poter essere utilizzato:

`python createDatasetFromStats <fileStatisticheWired> <fileStatisticheWireless>`

Tale script produce in output un file `WirelessWiredData.csv` del tipo:

```
latency,retransmission,bandwidth,label
0.000019,0,12526266,wired
0.000026,0,12524238,wired
0.000017,0,12524730,wired
0.000017,0,12524730,wired
0.000020,0,12520524,wired
  ...   ...   ...   ...
0.013583,298,2185006,wireless
0.002872,123,1363116,wireless
0.005966,391,1808764,wireless
0.008923,153,2060142,wireless
0.000042,0,528420,wireless
```

che può essere utilizzato per l'addestramento di modelli di machine learning.

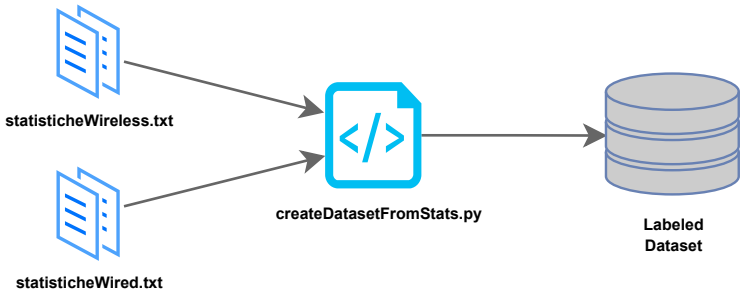

Figura 2.2: Creazione del dataset a partire dalle statistiche

3. Addestramento

Al fine di addestrare un modello in grado di prevedere la classe (wired/wireless) di appartenenza del traffico a partire dalle 3 feature (latenza, numero di ritrasmissioni e banda) si sono utilizzati due algoritmi di machine learning: logistic regression e SVM.

3.1 Logistic Regression

Logistic Regression è un algoritmo di machine learning utilizzato per la classificazione binaria, ma può essere facilmente esteso a problemi multiclasse. É possibile ricavare logistic regression utilizzando il framework Generalized Linear Models oppure a partire da Linear Regression. Nel secondo caso infatti è sufficiente osservare che Linear Regression utilizza una funzione di ipotesi del tipo:

$$h_{\underline{w}}(\underline{x}, \underline{w}) = \sum_{j=0}^{D-1} w_j \phi_j(\underline{x}) = \underline{w}^T \underline{\phi}(\underline{x})$$

Per ottenere un algoritmo di classificazione binaria è quindi necessario effettuare due considerazioni:

- l'output di linear regression $h_{\underline{w}}(\underline{x}, \underline{w}) = \underline{w}^T \underline{x}$ è continuo e assume valori in $]-\infty, \infty[$, mentre in un problema di classificazione binaria si hanno solo due valori $\{0, 1\}$;
- linear regression fornisce stime puntuali, nella classificazione si vuole $p(t^{(i)} = j|\underline{x}^{(i)})$;

Il primo problema può essere risolto imponendo una soglia γ tale che

$$\begin{cases} \text{se } \underline{w}^T \underline{x} > \gamma & \Rightarrow t = 1 \\ \text{se } \underline{w}^T \underline{x} \leq \gamma & \Rightarrow t = 0 \end{cases}$$

Per risolvere anche il secondo problema, invece di utilizzare una soglia, si utilizza una funzione non lineare f tale che

$$f(\underline{w}^T \underline{x}) \in [0, 1]$$

è quindi possibile dire che se $f(\underline{w}^T \underline{x}) \geq 0.5$ si ha $t = 1$ altrimenti si ha $t = 0$.

In logistic regressione la scelta della funzione non lineare ricade sulla funzione sigmoide

$$g(z) = \frac{1}{1 + e^{-z}} = \frac{e^z}{1 + e^z}$$

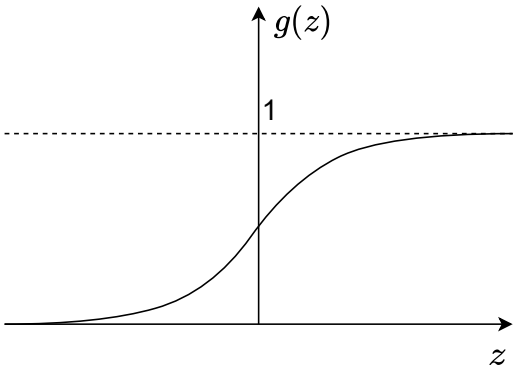

Figura 3.1: Funzione sigomoide

La funzione di ipotesi diventa quindi

$$h_{\underline{w}}(\underline{x}) = g(\underline{w}^T \underline{x}) = \frac{1}{1 + e^{-\underline{w}^T \underline{x}}}$$

che non consente di ottenere una soluzione in forma chiusa. Con il framework MLE è tuttavia possibile ricavare una soluzione iterativa del tipo

$$\underline{w} \leftarrow \underline{w} + \alpha \nabla_{\underline{w}} l(\underline{w})$$

dove

$$\frac{\partial l(\underline{w})}{\partial w_j} = \sum_{i=0}^{N-1} \left(t^{(i)} - h_{\underline{w}}(\underline{x}^{(i)}) \right) x_j^{(i)}$$

Tutto ciò è implementato nel modulo python `sklearn.linear_model`. Addestrando un modello tramite logistic regression si è quindi ottenuto un modello con prestazioni riassunte nella confusion matrix in figura 3.2. Per l'addestramento si è creato una griglia per il parametro di regolarizzazione, andando così ad individuare il migliore in un insieme di possibile valori.

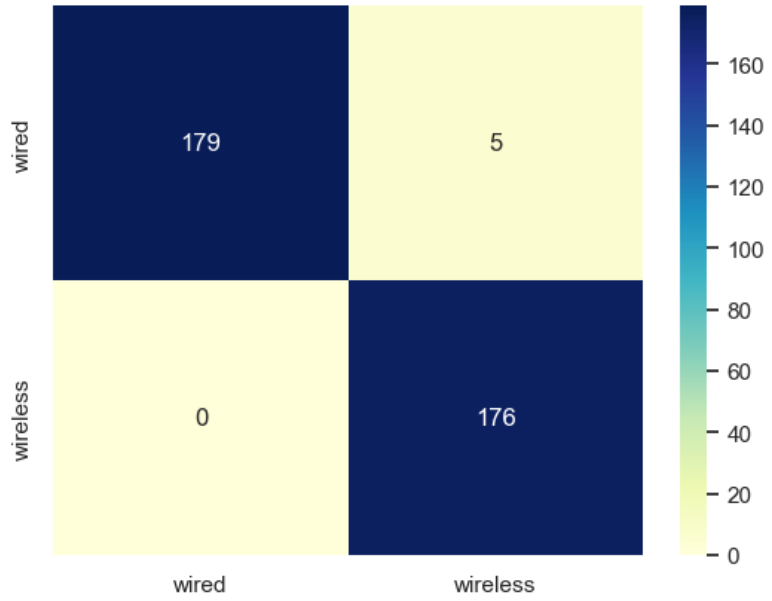

Figura 3.2: Confusion matrix ottenuta con un modello addestrato con Logistic Regression

Risulta quindi che tutti i 176 campioni wireless del test set sono stati classificati correttamente, mentre 5 campioni wired del test set sono stati missclassificati, ottenendo comunque una accuratezza pari a 0.986.

3.2 Support Vector Machine

SVM è un algoritmo di ML supervisionato, esso tende a massimizzare i vettori di supporto, ovvero i punti delle due fazioni che sono più vicini all'iperpiano separatore, andando così ad individuare il miglior iperpiano separatore.

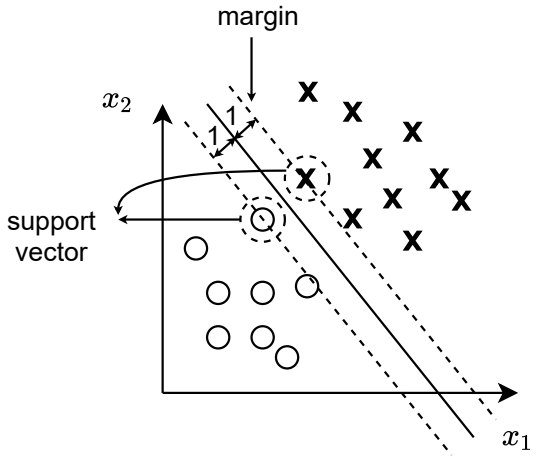

Figura 3.3: Vettori di supporto

Anche in questo caso il modulo python `sklearn.svm` implementa l'algoritmo SVM. Si è perciò sfruttato tale modulo per addestrare il modello. Si è inoltre utilizzato una griglia di ricerca degli iperparametri migliori utilizzando tre diversi tipi di kernel: lineare, RBF e polinomiale. Il kernel consente di trasformare lo spazio delle caratteristiche originale in uno spazio di dimensione superiore (potenzialmente infinita), dove i dati potrebbero diventare linearmente separabili.

Addestrando un modello tramite SVM si è quindi ottenuto un modello con prestazioni riassunte nella confusion matrix in figura 3.4.

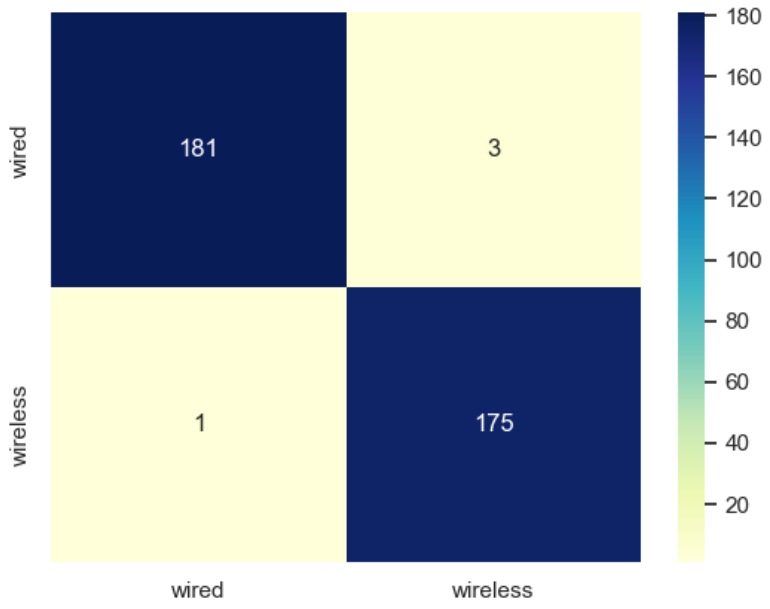

Figura 3.4: Confusion matrix ottenuta con un modello addestato con SVM

Risulta quindi che tutti un solo campione wireless e 3 campioni wired facenti parte del test set sono stati missclassificati.

3.3 Impatto delle singole features

Al fine di valutare l'importanza delle singole features nella classificazione si sono addestrati 6 modelli: 3 utilizzando una singola features e 3 utilizzando 2 features nelle 3 possibile combinazioni. Le prestazioni di tali modelli sono riassunte nella confusion matrix in figura 3.5.

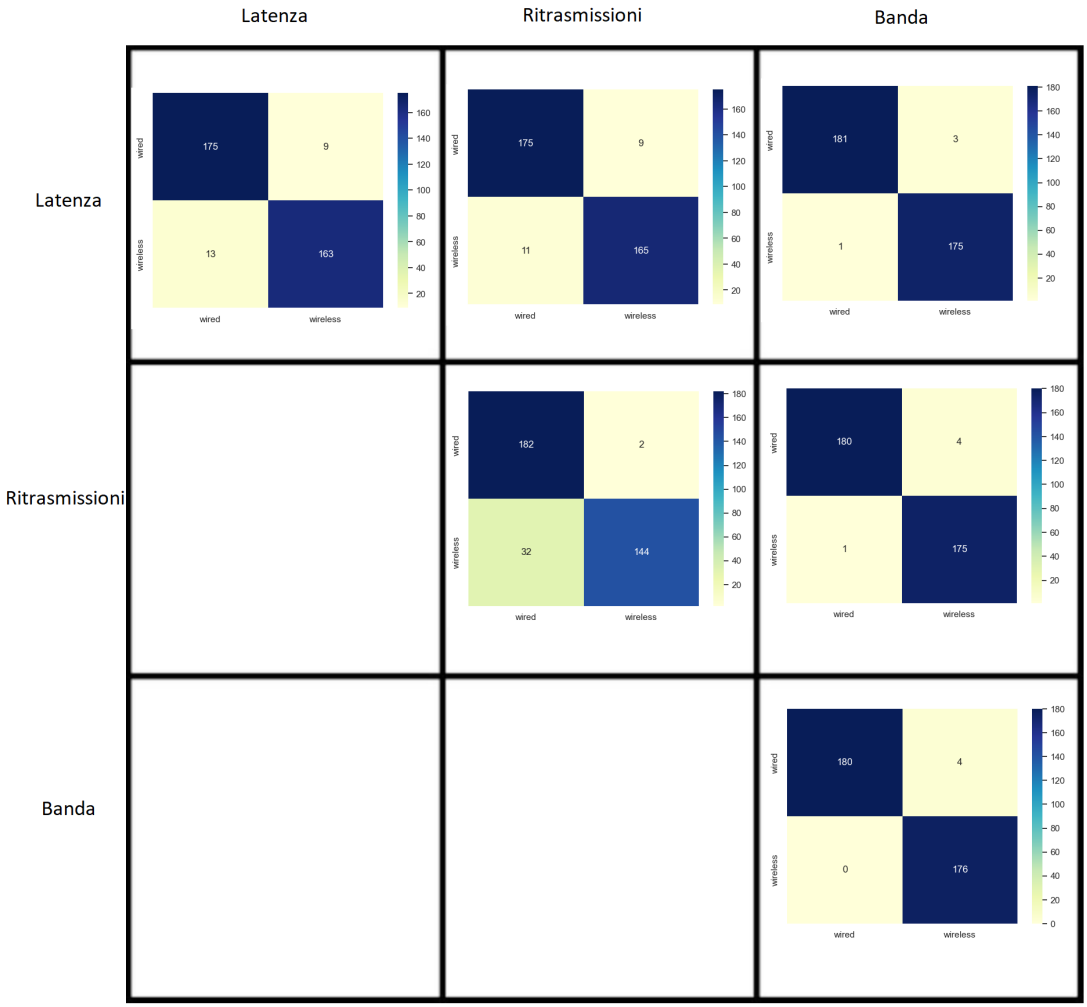

Figura 3.5: Confusion matrix per i 6 modelli addestrati

Dall'analisi dell'impatto delle singole features si è evidenziato che il modello addestrato con la sola features riguardante la banda presenta prestazioni analoghe a quello con tutte le features, mentre utilizzando le altre due features (anche usate insieme) portano ad un modello con prestazioni peggiori, capace comunque di classificare correttamente la maggior parte del traffico.

4. Test su traffico in tempo reale

Al fine di valutare le performance del modello addestrato con SVM si è utilizzato uno script python che, in loop esegue un comando tshark che cattura le statistiche sul traffico precedentemente viste, in un intervallo di 1 secondo. Infine fornisce in output la percentuale di campioni classificati come wired/wireless dal modello precedentemente addestrato.

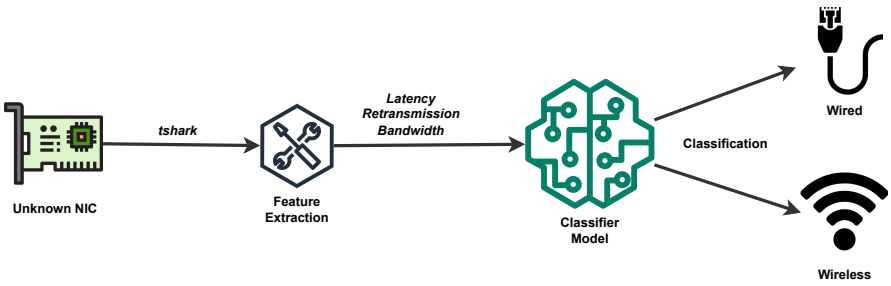

Figura 4.1: Test su traffico in tempo reale

Di seguito lo script python:

```
import subprocess
import re
from joblib import load
import numpy as np
import warnings
warnings.filterwarnings("ignore")

estimator = load("bestModelSVM.joblib")
scaler = load('scaler.pkl')
map = {
    0: "Wired",
```

```
    1: "Wireless"
}

# Selezione dell'interfaccia su cui catturare
result = subprocess.run(['tshark', '-D'], stdout=subprocess.PIPE,
                                  stderr=subprocess.PIPE, text=True
                                  )
print(result.stdout)
interfaccia = input("Select interface: ")

# Avvia la cattura sull'interfaccia
command = [
    'tshark', '-i', interfaccia, '-a', f'duration:1',
    '-q', '-p', '-z',
    'io,stat,1,AVG(tcp.analysis.ack_rtt)tcp.analysis.ack_rtt,COUNT
                                  (tcp.analysis.retransmission)
                                  tcp.analysis.retransmission,
                                  BYTES'
]

duration = 600
wiredSamples = 0
wirelessSamples = 0
# Cattura i dati ogni secondo ed effettua una previsione
for i in range(duration):
    print(i)
    result = subprocess.run(command, stdout=subprocess.PIPE,
                                  stderr=subprocess.PIPE, text=
                                  True)

    if result.returncode == 0:
        content = result.stdout
        content = content.replace(" ", "")
        content = re.sub('[^A-Za-z0-9|\n<.]+', '', content)
        lines = content.split('\n')
        data = [l for l in lines if "<" in l]
        linspl = data[0].split("|")
        [latency, retransmission, bandwidth] = float(linspl[2]),
                                      int(linspl[3]), int(
                                      linspl[4])
        print([latency, retransmission, bandwidth] )

        new_data = np.array([latency, retransmission, bandwidth]).
                                      reshape(1, -1)
        new_data_normalized = scaler.transform(new_data)

        label = map[int(estimator.predict(new_data_normalized))]
        print(label)
```

```
        if label == "Wireless":
            wirelessSamples += 1
        else:
            wiredSamples += 1
    else:
        print(f"Unable to monitor on interface {interfaccia}")
        break

print(f"Wireless sample: {wirelessSamples/duration*100} %")
print(f"Wired sample: {wiredSamples/duration*100} %")
```

In condizioni analoghe (posizione relativa PC-router) a quelle utilizzate per la cattura del dataset, il test effettuato sul traffico in tempo reale sembra funzionare perfettamente, con una percentuale di previsioni corrette comparabile a quella ottenuta sul test set (risulatati riassunti in figura 4.2). Modificando le condizioni tuttavia le performance tendono a degenerare, riuscendo comunque a classificare correttamente la maggior parte del traffico (84% del traffico classificato correttamente).

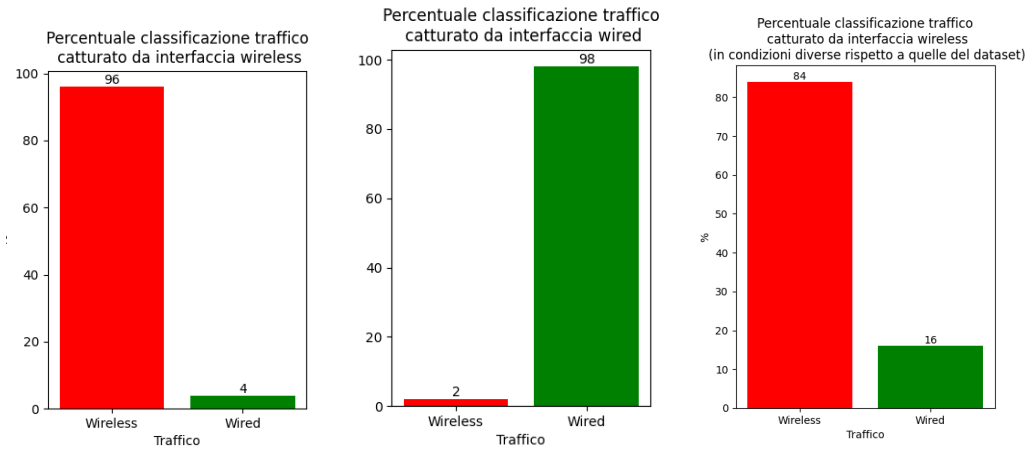

Figura 4.2: Risultati del test su traffico in tempo reale

5. Conclusione

5.1 Impatto della cattura su CPU e RAM

Si è infine effettuata una analisi dell'impatto della cattura sulla CPU e la RAM del dipositivo. Per fare ciò si è sfruttato il comando `top` e il seguente script python:

```
import subprocess
import time
import csv

command = ['top', '-b', '-n', '1']

with open("stats.csv", mode="w", newline='') as csv_file:
  csv_writer = csv.writer(csv_file)
  csv_writer.writerow(["VIRT", "CPU", "RAM"])
  for i in range(600):
    result = subprocess.run(command, stdout=subprocess.PIPE,
                                stderr=subprocess.PIPE, text=True
                                )
    if result.returncode == 0:
      content = result.stdout
      #print(content)
      for line in content.split("\n"):
        if "tshark" in line:
          cols = line.split()
          virt = cols[4]
          cpu = cols[8]
          ram = cols[9]
          csv_writer.writerow([virt, cpu, ram])
          print(i)
          print(virt)
          print(cpu)
```

```
            print(ram)
    else:
      print("Error")
      break
    time.sleep(1)
```

Tale script permette di isolare l'output di interesse dal comando `top` e creare il file `stas.csv`. Tali dati sono riassunti nei grafici in figura 5.1 e 5.2.

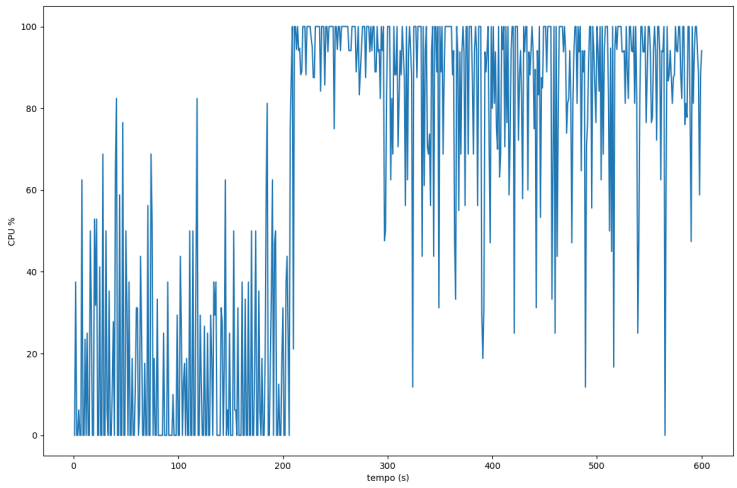

Figura 5.1: Percentuale di CPU utilizzata nella cattura

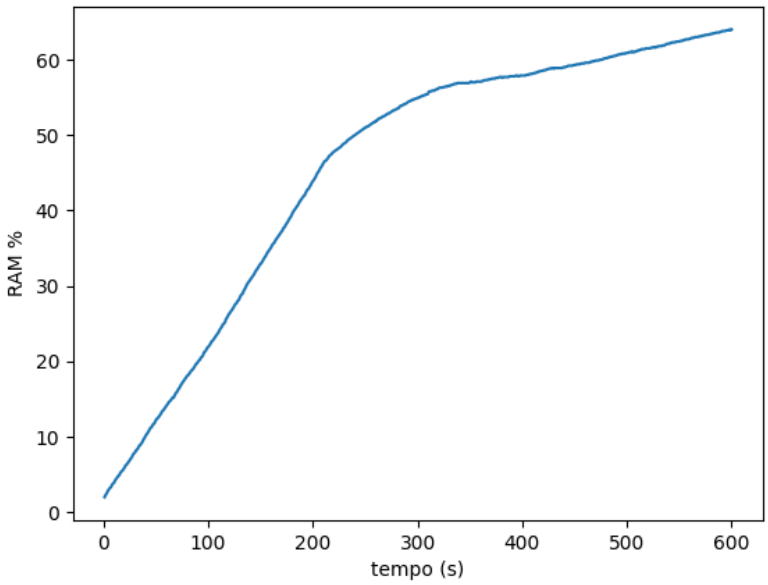

Figura 5.2: Percentuale di RAM utilizzata nella cattura

Il grafico riguardante il consumo della RAM è lineare a tratti e permette di individuare due fasi:

1. si raggiunge il 50% di RAM nei primi 200 secondi con una crescita lineare;
2. nei successivi 400 secondi si ha una crescita dal 50% al 60% sempre lineare;

Nella prima fase il consumo di CPU risulta essere relativamente basso, con picchi sporadici, nella seconda fase invece il consumo di CPU risulta essere notevolmente superiore. Si suppone che tale comportamento è dovuto all'esaurimento della memoria fisica con conseguente spostamento di pagine nella memoria swap.

5.2 Considerazioni finali

Si vogliono infine fare alcune considerazioni finali sulle features utilizzate. Si è infatti visto che l'addestramento di modelli utilizzando la sola features relativa alla larghezza di banda ha portato ad ottenere prestazioni identiche a quelle ottenute con tutte le features. Tuttavia si è anche visto che le altre due features sono comunque in grado di effettuare una buona classificazione. In particolare si vuole evidenziare il fatto che la larghezza di banda potrebbe essere una features particolarmente dipendente dalla rete analizzata. Quindi nonostante tale features

permetta di ottenere modelli con prestazioni migliori, tali modelli potrebbero non essere esportabili su altre reti. Inoltre per ottenere il dataset si è posizionato il terminale lontano dall'access point, con conseguente diminuzione della larghezza di banda. Per ottenere un modello migliore si sarebbe potuto catturare del traffico wireless andando a variare la distanza del terminale dall'access point.