INSTITUTO SUPERIOR TÉCNICO

TRAFFIC ENGINEERING

METI

Lab Report VI

Automatic Classification

2018/2019

Group 6
André Mendes - 78079
Filipe Fernandes - 78083

1 Introduction

The goal of this laboratory guide is to get comfortable with machine learning techniques. Particularly using a Naïve Bayes binary classifier to detect unfair bandwidth usage, caused by excessive peer-to-peer (p2p) connections.

For this work we used traffic data previously collected. Where each flow is classified as p2p or not p2p. This data was collected under controlled conditions and its structure is represented on the following table with sample data.

There are two file types, labeled (p2p or not p2p) and unlabeled (without p2p column).

Source IP	**# Simultaneous connections per IP**	**AVG bandwidth used by each IP**	**AVG packet size**	**Time of the day**	**Label**
192.168.5.0	50	10	300	8-16	not p2p
192.168.4.0	50	5	900	8-16	p2p
192.168.3.0	50	5	500	0-8	not p2p
...	...	...	...	...	...

Table 1: Structure of input files

We aim to develop an algorithm that allows us to predict for each entry of the previous unlabeled table if the traffic is p2p or not p2p. Besides that we will also evaluate the accuracy of our algorithm. We will develop our solution using *Python3*.

On the next sections, we will explain in detail the solution we developed and analyze its results.

2 Solution developed

In this sections we will explain our solution, the theory behind it and the code developed.

Naïve Bayes approximation is based on Bayes' theorem but it assumes that events are independent between them. Bayes theorem is represented below.

Bayes' theorem:

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B|A) * P(A)}{P(B)}$$

(where A and B are events and $P(B) \neq 0$)

Our solution can be divided in two major phases. Training phase and Testing phase. In the first one we estimate the probability of the events by analyzing a percentage of random entries of the labeled file. On the testing phase we use the probabilities calculated to predict if the remaining traffic flows are p2p or not.

On the training phase, for example if A is a certain **IP** and B is **p2p** or **not p2p** we use the Bayes' theorem to calculate for each connection knowing if it's p2p or not p2p what is the probability that the **IP's** connection is a certain one.

Knowing the results for each parameter of each line we predict if a certain connection is p2p or not. For example let's assume each connection (k) has three parameters: Parameter A = IP, Parameter B = Bandwidth, Parameter C = Time. With the results given to us by the Bayes' theorem we may calculate the following probabilities:

$$P(p2p|k) = [P(A|p2p) * P(B|p2p) * P(C|p2p) * P(p2p)]$$

$$P(notp2p|k) = [P(A|notp2p) * P(B|notp2p) * P(C|notp2p) * P(notp2p)]$$

For each connection if the first probability is higher the algorithm predicts that the connection is **p2p**, otherwise predicts that he connection is **not p2p**.

3 Results obtained

On the next table we display the error percentage for each file with a given training set percentage. Given that the training phase always trains with different random entries for each execution of our code the results are slightly different.

	1-labeled.dat	**2-labeled.dat**	**3-labeled.dat**	**4-labeled.dat**
70%:	25.26%	10.23%	0.0%	16.2%
False positive:	0.0%	0.23%	0.0%	0.07%
False negative:	25.26%	10%	0.0%	16.13%
50%:	24.56%	6.74%	0.0%	14.36%
False positive:	0.0%	0.36%	0.0%	0.14%
False negative:	24.56%	6.38%	0.0%	14.22%
30%:	24.93%	10.09%	0.09%	12.69%
False positive:	0.0%	0.0%	0.0%	0.36%
False negative:	24.93%	10.09%	0.09%	12.33%

Table 2: Error rate (rounded to two decimal units) for each file with a given training set percentage

Regarding the unlabeled files, on the next table we present how many p2p traffic flow were identified for each input file. Using the corresponding labeled file in training phase.

	1-unlabeled.dat	**2-unlabeled.dat**	**3-unlabeled.dat**	**4-unlabeled.dat**
(P2P, Not P2P)	(0, 5000)	(245, 4755)	(824, 9176)	(11, 4989)

Table 3: Number of P2P flows identified on each file

4 Conclusions

This work gave us the opportunity to implement a binary Naïve Bayes classifier.

The results obtained show that this machine learning technique has some flaws. Because for small or inconsistent samples, the output isn't accurate.

We also noticed that our results diverge at each simulation/execution of our developed code. To get more accurate results we should for each percentage calculate the average of, for instance, 100 simulations. We weren't able to do that because its execution would take several hours.

Annex:

A

Annex A - lab6.py

```
import pandas as pd
import argparse
import random

files = ["1-labeled.dat", "2-labeled.dat", "3-labeled.dat", "4-
   labeled.dat"]
files_u = ["1-unlabeled.dat", "2-unlabeled.dat", "3-unlabeled.dat", "
   4-unlabeled.dat"]


def get_parser():
    parser = argparse.ArgumentParser(description="""Traffic
       Engineering P2P classification tool""")
    parser.add_argument('-e', '--error', dest='error', help="""
       specify training percentage, ex: 0.7""",
                        action="store")
    parser.add_argument('-l', '--label', help="""Label the unlabeled
       files""",
                        action="store_true")
    return parser


def gen_random(start, end, num):
    res = []

    for j in range(num):
        val = random.randint(start, end)
        while val in res:
            val = random.randint(start, end)
        res.append(val)

    return res


def index(filename, mode, percentage):
    df = pd.read_csv(filename, header=None)
    dic_ips = {}
    list_ip = []
    dic_connection = {}
    list_connection = []
    dic_bandwidth = {}
    list_bandwidth = []
    dic_packet_size = {}
    list_packet_size = []
    dic_time = {}
    list_time = []
```

```
dic_p2p = {}
list_p2p = []

if mode == "label":
    mode_range = range(len(df))
else:
    val = int(percentage*len(df))
    mode_range = gen_random(0, 9999, val)
    all_range = list(range(len(df)))
    missing_range = set(all_range).difference(mode_range)

for n in mode_range:

    ip = df.values[n][0]
    connection = df.values[n][1]
    band = df.values[n][2]
    packet_size = df.values[n][3]
    time = df.values[n][4]
    p2p = df.values[n][5]

    # Populate IP dictionary
    if ip not in list_ip:
        list_ip.append(ip)
        if p2p == "p2p":
            dic_ips.update({ip: (1, 0)})
        else:
            dic_ips.update({ip: (0, 1)})
    else:
        tmp = dic_ips.get(ip)
        if p2p == "p2p":
            tmp = (tmp[0]+1, tmp[1])
            dic_ips.update({ip: tmp})
        else:
            tmp = (tmp[0], tmp[1]+1)
            dic_ips.update({ip: tmp})

    # Populate Connection dictionary
    if connection not in list_connection:
        list_connection.append(connection)
        if p2p == "p2p":
            dic_connection.update({connection: (1, 0)})
        else:
            dic_connection.update({connection: (0, 1)})
    else:
        tmp = dic_connection.get(connection)
        if p2p == "p2p":
            tmp = (tmp[0]+1, tmp[1])
            dic_connection.update({connection: tmp})
        else:
            tmp = (tmp[0], tmp[1]+1)
            dic_connection.update({connection: tmp})
```

```python
# Populate bandwidth dictionary
if band not in list_bandwidth:
    list_bandwidth.append(band)
    if p2p == "p2p":
        dic_bandwidth.update({band: (1, 0)})
    else:
        dic_bandwidth.update({band: (0, 1)})
else:
    tmp = dic_bandwidth.get(band)
    if p2p == "p2p":
        tmp = (tmp[0]+1, tmp[1])
        dic_bandwidth.update({band: tmp})
    else:
        tmp = (tmp[0], tmp[1]+1)
        dic_bandwidth.update({band: tmp})

# Populate packet_size dictionary
if packet_size not in list_packet_size:
    list_packet_size.append(packet_size)
    if p2p == "p2p":
        dic_packet_size.update({packet_size: (1, 0)})
    else:
        dic_packet_size.update({packet_size: (0, 1)})
else:
    tmp = dic_packet_size.get(packet_size)
    if p2p == "p2p":
        tmp = (tmp[0]+1, tmp[1])
        dic_packet_size.update({packet_size: tmp})
    else:
        tmp = (tmp[0], tmp[1]+1)
        dic_packet_size.update({packet_size: tmp})

# Populate time dictionary
if time not in list_time:
    list_time.append(time)
    if p2p == "p2p":
        dic_time.update({time: (1, 0)})
    else:
        dic_time.update({time: (0, 1)})
else:
    tmp = dic_time.get(time)
    if p2p == "p2p":
        tmp = (tmp[0]+1, tmp[1])
        dic_time.update({time: tmp})
    else:
        tmp = (tmp[0], tmp[1]+1)
        dic_time.update({time: tmp})

if p2p not in list_p2p:
    list_p2p.append(p2p)
    if p2p == "p2p":
        dic_p2p.update({"count": (1, 0)})
```

```
            else:
                dic_p2p.update({"count": (0, 1)})
        else:
            tmp = dic_p2p.get("count")
            if p2p == "p2p":
                tmp = (tmp[0] + 1, tmp[1])
                dic_p2p.update({"count": tmp})
            else:
                tmp = (tmp[0], tmp[1]+1)
                dic_p2p.update({"count": tmp})

    if mode == "error":
        return dic_ips, dic_connection, dic_bandwidth,
           dic_packet_size, dic_time, dic_p2p, missing_range
    else:
        return dic_ips, dic_connection, dic_bandwidth,
           dic_packet_size, dic_time, dic_p2p


def get_prob(dic_p2p):
    prob_p2p = dic_p2p.get("count")[0] / (dic_p2p.get("count")[0] +
       dic_p2p.get("count")[1])
    prob_np2p = dic_p2p.get("count")[1] / (dic_p2p.get("count")[0] +
       dic_p2p.get("count")[1])
    return prob_p2p, prob_np2p


def output(filename, mode, missing_range, dic_ips, dic_connection,
   dic_bandwidth, dic_packet_size, dic_time,
          prob_p2p, prob_np2p):
    lines = []
    dfu = pd.read_csv(filename, header=None)

    if mode == "label":
        mode_range = range(len(dfu))
    else:
        error = 0
        nr_times = 0
        mode_range = missing_range

    for n in mode_range:
        ip_u = dfu.values[n][0]
        connection_u = dfu.values[n][1]
        band_u = dfu.values[n][2]
        packet_size_u = dfu.values[n][3]
        time_u = dfu.values[n][4]

        if mode == "error":
            p2p_u = dfu.values[n][5]

        # IP:
        ip_label_ip = dic_ips.get(ip_u)
        if ip_label_ip is not None:
            ip_p2p = ip_label_ip[0]/prob_p2p
            ip_np2p = ip_label_ip[1]/prob_np2p

        # Connection:
        ip_label_connection = dic_connection.get(connection_u)
        if ip_label_connection is not None:
            conn_p2p = ip_label_connection[0]/prob_p2p
            conn_np2p = ip_label_connection[1]/prob_np2p

        # Bandwidth:
        ip_label_band = dic_bandwidth.get(band_u)
        if ip_label_band is not None:
            band_p2p = ip_label_band[0] / prob_p2p
            band_np2p = ip_label_band[1] / prob_np2p

        # Packet_size:
        ip_label_packet = dic_packet_size.get(packet_size_u)
        if ip_label_packet is not None:
            packet_p2p = ip_label_packet[0] / prob_p2p
            packet_np2p = ip_label_packet[1] / prob_np2p
        # Time:
        ip_label_time = dic_time.get(time_u)
        if ip_label_time is not None:
            time_p2p = ip_label_time[0] / prob_p2p
            time_np2p = ip_label_time[1] / prob_np2p

        total_p2p = ip_p2p * conn_p2p * band_p2p * packet_p2p *
           time_p2p * prob_p2p
        total_np2p = ip_np2p * conn_np2p * band_np2p * packet_np2p *
           time_np2p * prob_np2p

    # Label mode
        if mode == "label":

            if total_p2p > total_np2p:
                line = ip_u + "," + str(connection_u) + "," + str(
                   band_u) + "," + str(packet_size_u) + "," + time_u
                   \
                       + "," + "p2p\n"
                lines.append(line)

            if total_p2p < total_np2p:
                line = ip_u + "," + str(connection_u) + "," + str(
                   band_u) + "," + str(packet_size_u) + "," \
                       + time_u + "," + "not␣p2p\n"
                lines.append(line)
        else:
            if total_p2p > total_np2p:
                if p2p_u != "p2p":
                    error = error + 1
            if total_p2p < total_np2p:
                if p2p_u != "not␣p2p":
                    error = error + 1
            nr_times = nr_times + 1

    if mode == "label":
        # Write Output file
        if filename.startswith("1-"):
            f = open("out-1-labeled.dat", "w")
            f.writelines(lines)
            f.close()
        if filename.startswith("2-"):
            f = open("out-2-labeled.dat", "w")
            f.writelines(lines)
            f.close()
        if filename.startswith("3-"):
            f = open("out-3-labeled.dat", "w")
            f.writelines(lines)
            f.close()
        if filename.startswith("4-"):
            f = open("out-4-labeled.dat", "w")
            f.writelines(lines)
            f.close()
    else:
        perc_error = (error / nr_times) * 100
        print("Error␣=", perc_error, "%")


def main():

    parser = get_parser()
    args = vars(parser.parse_args())
    label = args['label']
    error = args['error']

    # Label mode
    if label:
        count = 0
        for filename in files:
            print(filename)
            print(files_u[count])
            dic_ips, dic_connection, dic_bandwidth, dic_packet_size,
               dic_time, dic_p2p = index(filename, "label", 1)
            prob_p2p, prob_np2p = get_prob(dic_p2p)
            output(files_u[count], "label", [], dic_ips,
               dic_connection, dic_bandwidth, dic_packet_size,
               dic_time,
                   prob_p2p, prob_np2p)
            count += 1

    # Error mode
    for n in range(0, 10):
        print("Results␣of␣run:␣%d", n)
        if error is not None:
            for filename in files:
                print(filename)
                dic_ips, dic_connection, dic_bandwidth,
                   dic_packet_size, dic_time, dic_p2p, missing_range
                   = \
                    index(filename, "error", float(error))
                prob_p2p, prob_np2p = get_prob(dic_p2p)
                output(filename, "error", missing_range, dic_ips,
                   dic_connection, dic_bandwidth, dic_packet_size,
                       dic_time, prob_p2p, prob_np2p)

    if not label and error is None:
        parser.print_help()


if __name__ == "__main__":
    main()
```