Spy Your Mate

Project Report — 054327 Wireless Internet | 054323 Internet of Things

Davide Giacomini
Polytechnic University of Milan (PoliMi)
Deparment of Electronics, Information and Bioengineering
Milan (MI), Italy
August 21, 2022

***Abstract*—After a global pandemic, video streaming calls have become the normality in every situation. Even though the major providers usually encrypt the traffic, some information can be extracted by the packets that are sent between the parties through the internet. By aggregating the packets of a conversation and analyzing some of their features through a ML algorithm, we can infer if somebody is present in front of the camera or there is nobody.**

I. Introduction

In this report I am going to illustrate the work done for the joint project of Wireless Internet and Internet of Things.

I was required to extract some information from a video call with a provider of my choice. In particular, I was requested to analyze the traffic generated by a webcam call and, from such traffic, identify whether a person or only the background is recorded in the video. I was asked to capture the network traffic and filter all the video call packets. Subsequently, I had to extract some important features from the captured packets, and use those features to train a model of Machine Learning. Finally, I had to predict the presence of a person in the background using the trained model and compare the prediction with the ground truth. For the comparison, I had to use a Node-RED chart.

You can find all my code on GitHub[1]. It has three folders that divide everything between Node-RED, Python and Latex (for this report).

II. Method

The first step was to choose a software for network traffic, hence I chose Wireshark[2]. I recorded three vide calls. In the first one I was alone, in the other two I was with another person (different each time). The captures duration is more or less between 300 seconds and 420 seconds. For the video calls I used my laptop (Asus) and my smartphone (Samsung Galaxy S10). The provider I used was Zoom[3], as I am comfortable with it.

I needed both the IP addresses and the UDP ports to filter the packets of the video call. In Wireshark there is a convenient window under `Statistics -> Conversations` which shows you the conversations based on the chosen protocol. Clicking on `UDP`, I ordered the conversations in order to highlight the one with the highest number of packets (and bytes) transferred. It was easy to see which one is the actual video call, because there is a difference of several orders of magnitude in number of bytes exchanged. The addresses and the ports that I extracted from Wireshark can be seen at Table I.

A. Python Code

All the Python code that I will refer to in this section can be found on GitHub under the folder `python/`.

After extracting the addresses, I used them to filter out only the packets regarding the conversation between the parties. I used PyShark[4] to manipulate the captures. The Python scripts are used to extract the packets, organize them in chunks of half a second, and save the chunks and their features on a csv file, later used for training or testing. The scripts are called in order by the `prepare_*.py` scripts. I divided them for convenience during debugging.

The training and testing is done by two scripts called `script_ML_*.py`. The cross validation is only used to understand if the features chosen are good to make an accurate prediction, and the second script is used to actually predict the results. I used the second capture (`capture2_test.csv`) to test the model and the other two captures to train it.

I incrementally added features to get a model each time more accurate, and I finally ended up with 12 features. Each of them represents a chunk of half a second packets. I am going to describe them below. Notice that each point represents three features, i.e. inbound packets, outbound packets and all packets:

- Average packet interval: the average time between one packet and the next.
- Average packet size: the average packet size in bytes.
- Packets number: the total number of packets.
- Bit rate: the number of bytes per half a second.

I used two different algorithms to train the model: logistic regression and random forest. The predictions are saved as `csv` files to be later used in Node-RED.

B. Node-RED Flow

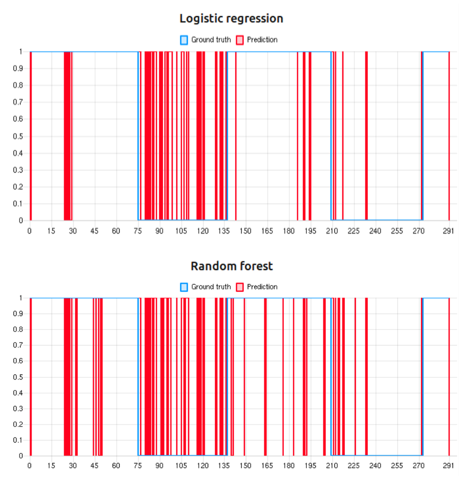
The Node-RED flow is available on GitHub under `NodeRed/nodered_flow.txt`.

[1]github.com/davide-giacomini/spy_your_mate
[2]wireshark.org
[3]zoom.us

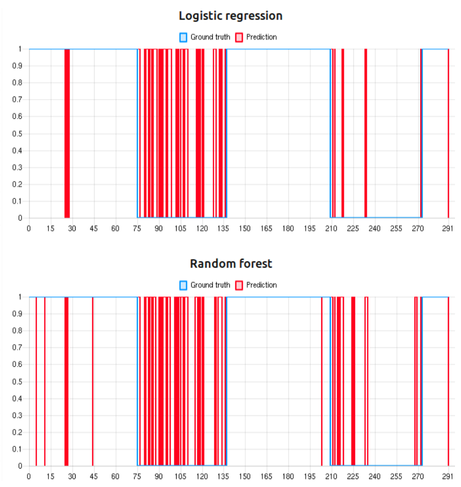
[4]pypi.org/project/pyshark/

	Zoom IP Address	Zoom UDP Port	Local IP Address	Local UDP Port
Capture 1	149.137.11.203	8801	192.168.1.15	60235
Capture 2	149.137.11.142	8801	192.168.1.15	51541
Capture 3	149.137.10.223	8801	192.168.1.15	52409

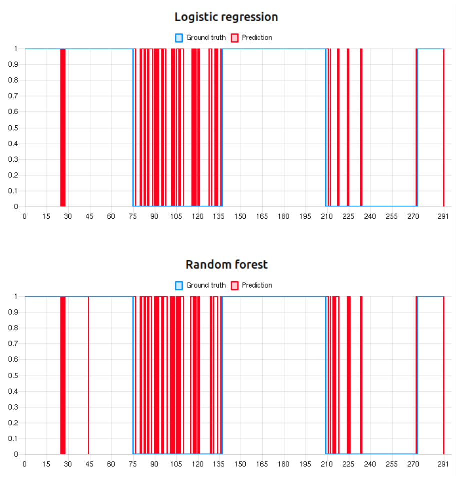
TABLE I: Addresses for each capture.



(a) First Attempt



(b) Second Attempt



(c) Last Attempt

Fig. 1: Node-RED charts for comparison with ground truth

In Node-RED I was required to compare the ground truth with the actual prediction done in Python.

In order to do that, I read with Node-RED two different csv files (the capture file exported in csv and the prediction of python) and compared the results with the Chart of Node-RED. For comparing the results, I had to filter out the ground truth taking only the multiples of half a second. The prediction file, instead, had already been prepared with Python.

C. *ML Training and Testing*

For the training and the testing of models (both logistic regression and random forest), I divided the work in two main parts. The former one is the cross validation, to understand more or less the accuracy of my predictions, and the latter one is the actual training and prediction with the features I chose during the cross validation. Both the cross validation and the actual training and prediction take as input the `capture*.csv` files that I prepared with PyShark. I used `scikit-learn`[5] for the ML methods.

The cross validation file (`script_ML_cross_validation.py`) takes as input the csv files thought for the training part (I chose capture 1 and 3 for the training part) and applies a K fold cross validation with $k = 5$. I set the maximum training set size to the 80% of the training data, hence the minimum validation set is 20% of the training data. I am basing the terminology on the slide number 5 from the set of slides [1]. After the cross validation, a learning curve shows the training and validation accuracy in function of the number of samples (training size).

[5]scikit-learn.org/stable/

After the cross vaildation, we can proceed with the actual training of the model and the subsequent prediction with the trained model. The steps are two: the training and then the testing. For the training, as before, the file takes as input the csv training files and trains the two models (logistic regression and random forest) with those two files. Then, for the testing, the file takes as input the csv file of capture 2 and predicts the presence or absence of a person in front of the webcam. The prediction is then exported in a csv file for Node-RED and is also compared to the ground truth, showing the confusion matrix and the accuracy of the prediction.

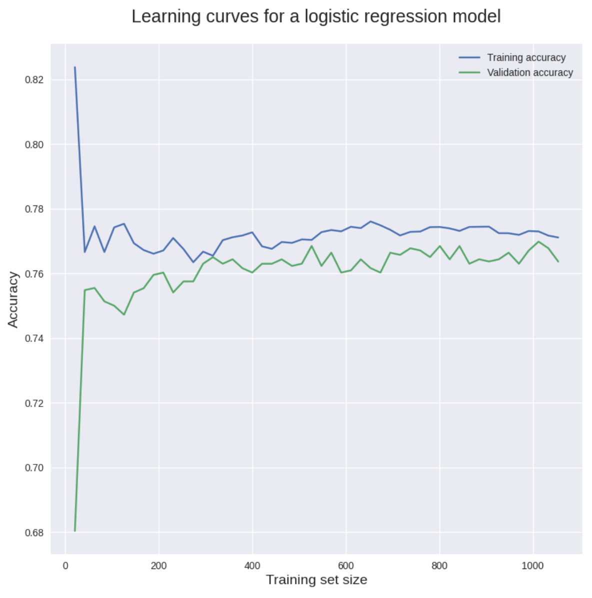
III. Experimental Results

I ran several trials, but I will report here only the ones that I found interesting for the sake of the document.

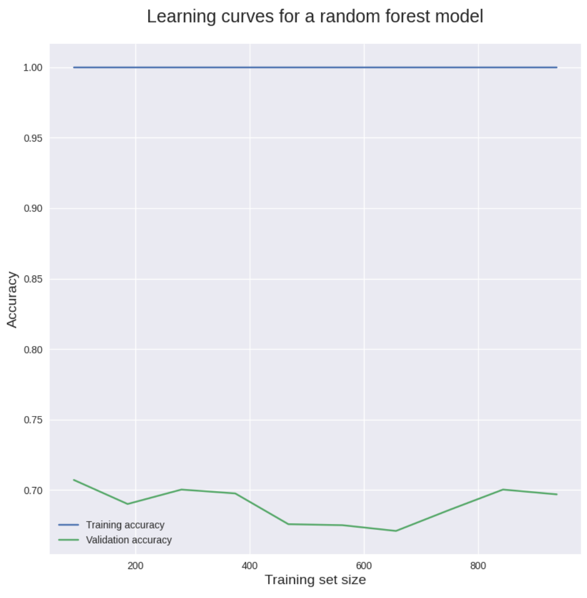
A. *First attempt*

I started with five features: average packets interval, average packets size, inbound and outbound packets number, total packets number.

1) Cross validation results: The learning curve of the cross validation for both models can be seen at Fig. 2. Notice that the training curve of the random forest is always equal to 100% accuracy, and that is not exactly the best behaviour required (Fig. 2a). Nonetheless, the validation accuracy is quite consistent with the testing results, which I will show later. In particular, the validation accuracy stands at around 70%. On the other hand, the logistic regression curve is quite

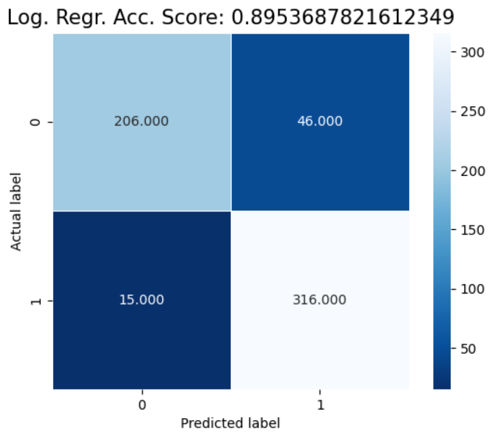


(a) Logistic Regression Curve

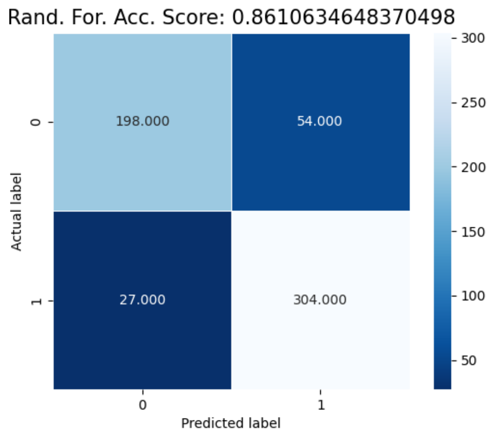


(b) Random Forest Curve

Fig. 2: Learning curves in the first attempt



(a) Logistic Regression Confusion Matrix



(b) Random Forest Confusion Matrix

Fig. 3: Confusion matrices in the first attempt

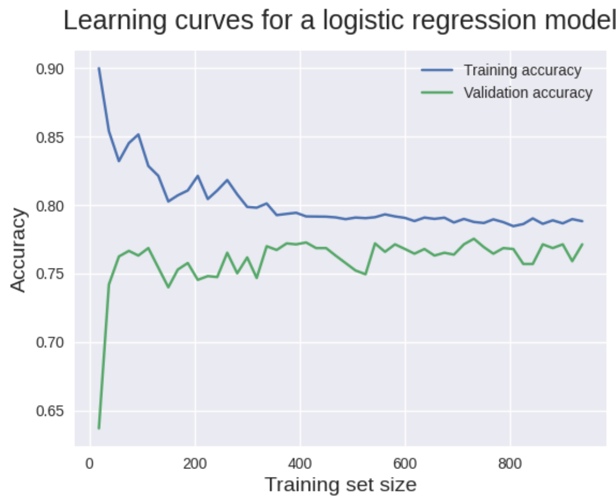
"normal" (Fig. 2b). The training accuracy is about 77% and the validation accuracy is about 76%.

2) Training and testing results: After the prediction, the Python script shows a confusion matrix and the accuracy for each model. You can see both confusion matrices and accuracies at Fig. 2. You can notice that the model predicts relatively well when the person is actually in front of the webcam (output 1), but it cannot predict very well when there is only background (output 0). After exporting the predicted results to the csv file, I used Node-RED to read from that file and compare it to the ground truth. The results of the comparison are shown in a Node-RED chart, that I am reporting here at Fig. 1a. As we noticed with the confusion matrices, the models get a better result when there is actually a person in front of the camera. In this precise case, the random forest model is very dirty and not very useful.

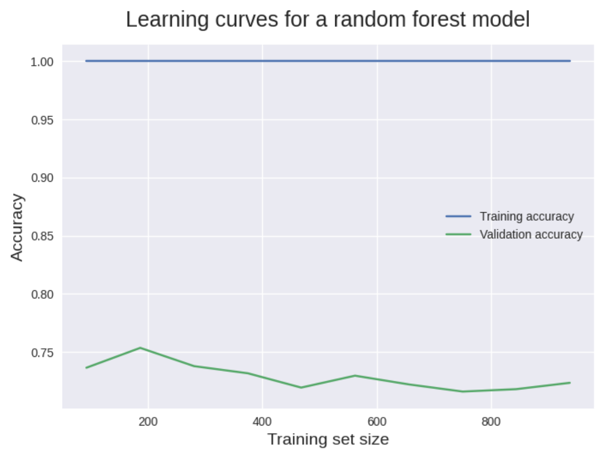
B. Second attempt

For this second attempt, I added some features based on the precedent ones, dividing though inbound and outbound packets. In particular, I added: average inbound and outbound packets interval, average inbound and outbound packets size.

1) Cross validation results: The learning curve here got a little better for the random forest model. As we can see

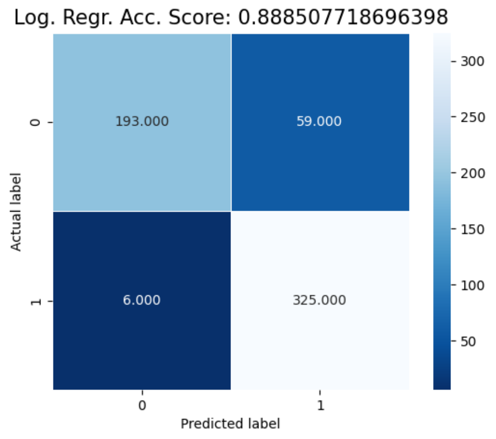


(a) Logistic Regression Curve

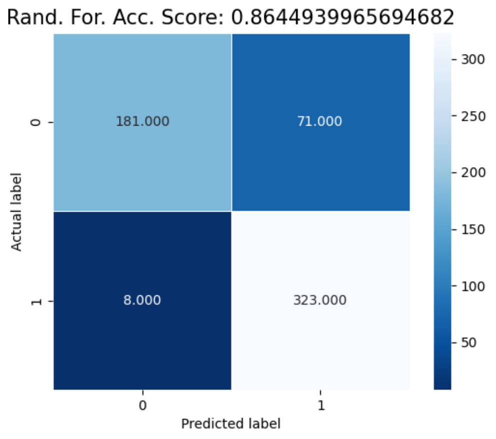


(b) Random Forest Curve

Fig. 4: Learning curves in the second attempt



(a) Logistic Regression Confusion Matrix



(b) Random Forest Confusion Matrix

Fig. 5: Confusion matrices in the second attempt

at Fig. 4, the training curve for the logistic regression model stands at around 78%, the validation curve for the same model stands at around 76% and the validation curve for the other model is around 72% (2% more than the first attempt). As always, the training curve of the random forest model is not informative, as it always stays at 100%.

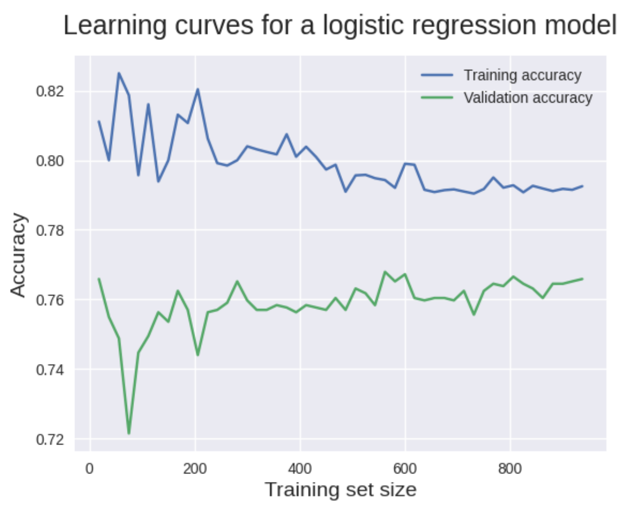
2) Training and testing results: We can see from Fig. 5 how the prediction of the presence of a person has got better again, expecially comparing the matrices to the ones at Fig. 3. Although, the background recognition is still a problem. If we take a look at the chart of Fig. 1b, we can see how now the situation got better expecially for the random forest model. We can basically see that, apart from one or two problems, the prediction of the presence of a person is basically accurate for both models.

C. *Last attempt*

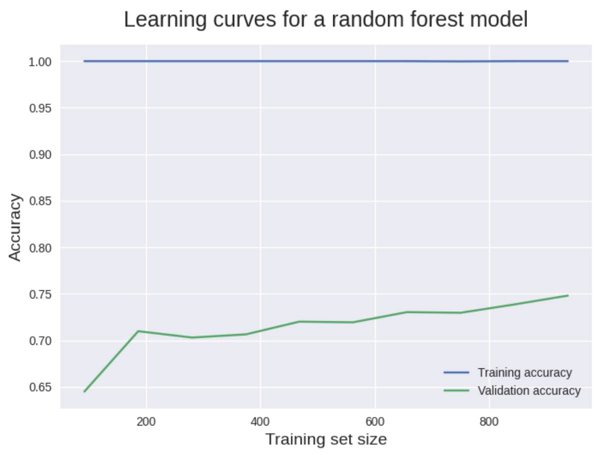
For the last attempt, I added the bit rate for inbound, outbound and all the packets.

1) Cross validation results: I could still increase the validation accuracy for the random forest model. At Fig. 6 we can see the different values. The training and validation curves of the logistic regression model reciprocally stands at around 79% and 76%, and the validation curve of the random forest model is 74%.

2) Training and testing results: The confusion matrices at Fig. 7 shows that there is relatively no difference between the previous attempt, but the charts at Fig. 1c show a very little improvement in the cleanness of the prediction. Unfortunately, although the last background interval is predicted quite well, especially with the logistic regression model, the first one is not predicted at all.

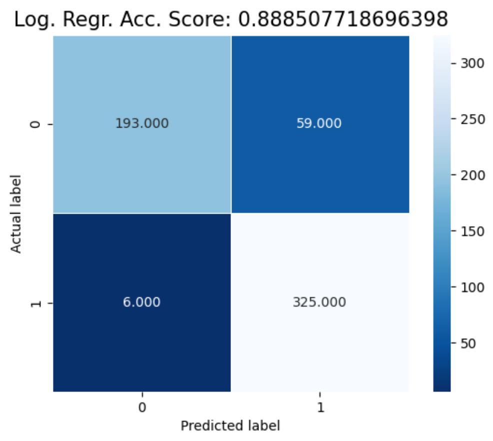


(a) Logistic Regression Curve

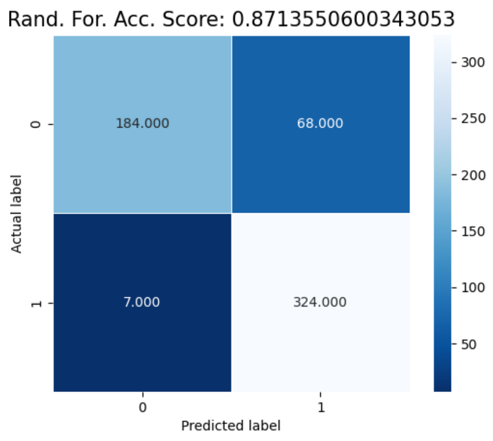


(b) Random Forest Curve

Fig. 6: Learning curves in the last attempt



(a) Logistic Regression Confusion Matrix



(b) Random Forest Confusion Matrix

Fig. 7: Confusion matrices in the last attempt

D. *Failed attempts*

I will not report here the several attempts that are not worth the time of reading, but I wanted to consider a particular case.

I told you that I divided the packets in chunks of half a second, but it is not the only possibility. The rules of the project required to divide the packets in chunks between 100ms and 900ms, hence I also tried to change the interval, for example with 250ms of interval. Although the number of samples was increased (because of the shorter interval), this seemed to be useless from the point of view of training the models. Instead, it seemed to degrade the predictions, especially of the logistic regression model.

IV. Conclusion

I reported here the steps that I have taken to work on the joint project of Wireless Internet and Internet of Things, analyzing the quality of the training and the predictions.

I can conclude here that, based on the results obtained during the tests, even though the communication between two parties is encrypted in Zoom[6], important information can be easily extracted sniffing packets and analyzing some of their features through a Machine Learning algorithm. This conclusion is strong when we consider that those features are never encrypted as they are necessary for a correct protocol procedure in the Internet.

My conclusion can be also extended recognizing a better prediction when the person is actually in front of the camera rather than when there is nobody.

[6]support.zoom.us/hc/en-us/articles/201362723-Encryption-for-Meetings

V. Further Improvements

This work could be improved with a better and bigger dataset, in order to try different combinations of captures. Moreover, I think that if there are some valid features that I missed in my work, it could be a strong improvement even without amplifying the dataset.

Moreover, I was unfortunately alone and I did not try to make a real video call, instead I called myself between laptop and smartphone. Considering the local Wifi network, the only external information the algorithm can extract is the one coming from the Zoom server and not from the other endpoint of the communication. I did however involved other people during the call, in order to have some data with more than one person in front of the camera. I although strongly believe that, overall, this work can be improved with a better environment.

References

[1] Matteo Matteucci. Neural networks training and overfitting, 2022. URL http://chrome.ws.dei.polimi.it/images/8/8a/AN2DL_03_2122_NeuralNetwroksTraining.pdf. Accessed: August 21, 2022.