Spy your mate – Project #3

In the following project, we were requested to analyse the traffic generated by a webcam during an ongoing video call in order to identify, from such traffic, whether a person was in front of the camera or only the background was recorded in the video.

In the upcoming paragraphs, we'll discuss how we structured our work and analysed the obtained information to complete the request.

The whole project was carried out on Jupyter Notebook, because of its ease of use given the possibility to work step by step with singular cells rather than with a whole python program.

1. Capturing traffic

First, we needed to set up a video call between two different devices and to "sniff" the traffic generated by said video call. This task was carried out by starting a "Zoom meeting" between a pc (MacBook Pro) and a phone (iPhone), meanwhile the video traffic was captured by exploiting the Wireshark software. With the aim of obtaining a simpler and clearer capture, both the pc and the phone were connected to the home Wi-Fi network, and every other device located in the house was either turned off or switched-on airplane mode, as to free the Wi-Fi channel and give more (if not all) the bandwidth to the technologies involved in the call.

According to the project's specifications, we alternated period of times (specifically of 1 minute duration) with and without the person in front of the camera, hence only the background was recorded in the latter. After a few attempts, we managed to acquire a satisfactory dataset to carry on the project. The videocall lasted around 10 minutes, while the capture itself lasted exactly 602.849661 seconds (when the last packet is registered), meaning circa 10:04 minutes.

2. Defining the datasets

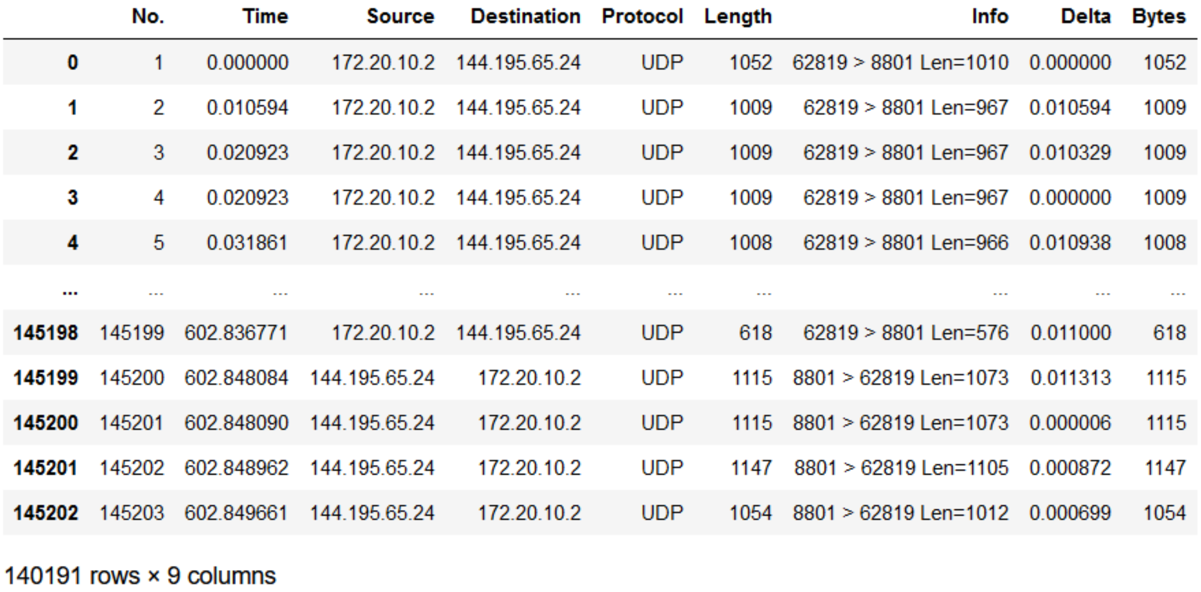
At the beginning of the traffic capture, we started a stopwatch to keep track of the exact times in which the user changed its position, attempting to keep that interval fixed to one minute. With this information we were able to create a ground truth table, which would be extremely useful later in the work, to procure classification metrics about the algorithm performance (that information has been added in an updated version of the capture csv file).

Once the capturing process was over, we first analysed the data frame focusing on the UDP: the ones to expect out of a live audio-video streaming, which is a time-sensitive application, so dropping packets is preferable to waiting for packets delayed due to retransmission (TCP).

Nonetheless, it was nearly impossible to detect noticeable differences between any of them with a naked eye.

We analysed the capture's frames importing the "Pandas" library's tools (a proper Data Analysis library) to perform our tasks.
Starting from the capture csv file, filtered of all non-UDP packets, we put our attention on the main parameters already available thanks to Wireshark, namely:

- the source and the destination IP address --> "Source" and "Destination"
- the packets' size (in bytes) --> "Lenght"
- the interval between packets (in seconds) --> "Delta"

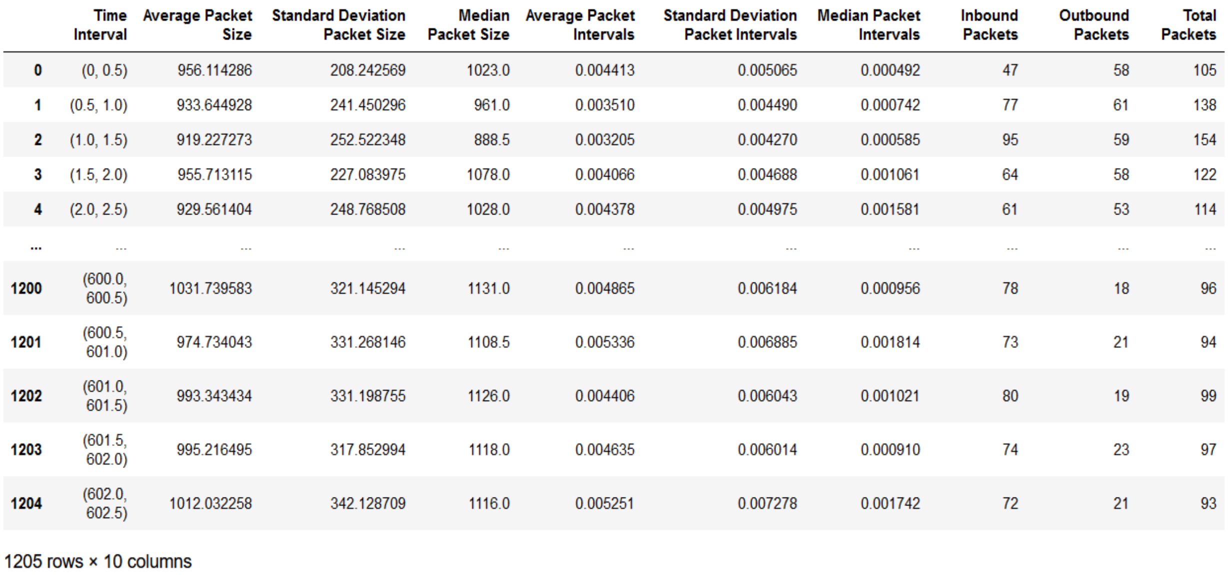


	No.	Time	Source	Destination	Protocol	Length	Info	Delta	Bytes
0	1	0.000000	172.20.10.2	144.195.65.24	UDP	1052	62819 > 8801 Len=1010	0.000000	1052
1	2	0.010594	172.20.10.2	144.195.65.24	UDP	1009	62819 > 8801 Len=967	0.010594	1009
2	3	0.020923	172.20.10.2	144.195.65.24	UDP	1009	62819 > 8801 Len=967	0.010329	1009
3	4	0.020923	172.20.10.2	144.195.65.24	UDP	1009	62819 > 8801 Len=967	0.000000	1009
4	5	0.031861	172.20.10.2	144.195.65.24	UDP	1008	62819 > 8801 Len=966	0.010938	1008
...	...	...	...	...	...	...	...	...	...
145198	145199	602.836771	172.20.10.2	144.195.65.24	UDP	618	62819 > 8801 Len=576	0.011000	618
145199	145200	602.848084	144.195.65.24	172.20.10.2	UDP	1115	8801 > 62819 Len=1073	0.011313	1115
145200	145201	602.848090	144.195.65.24	172.20.10.2	UDP	1115	8801 > 62819 Len=1073	0.000006	1115
145201	145202	602.848962	144.195.65.24	172.20.10.2	UDP	1147	8801 > 62819 Len=1105	0.000872	1147
145202	145203	602.849661	144.195.65.24	172.20.10.2	UDP	1054	8801 > 62819 Len=1012	0.000699	1054

140191 rows × 9 columns

Fig 1: The capture file from Wireshark converted into a csv.

Then, we proceeded to discretise time in intervals of 500ms and to retrieve more in-depth statistical information for each above-mentioned interval: we iterated the computation of median, mean and standard deviation, through the rows of the filtered dataframe, selecting the columns "Lenght" and "Delta". Furthermore, we extracted the number of inbound and outbound packet, thanks to the source and destination IP addresses.
The obtained result was the "Dataset_iotwi" csv file (shown in Fig 2), composed of the following parameters: the time interval, the average packet, with its standard deviation and its median, the packet intervals average, with its standard deviation and its median as well, the number of inbound packets, the number of outbound packets, and finally the total of the packets exchanged.



	Time Interval	Average Packet Size	Standard Deviation Packet Size	Median Packet Size	Average Packet Intervals	Standard Deviation Packet Intervals	Median Packet Intervals	Inbound Packets	Outbound Packets	Total Packets
0	(0, 0.5)	956.114286	208.242569	1023.0	0.004413	0.005065	0.000492	47	58	105
1	(0.5, 1.0)	933.644928	241.450296	961.0	0.003510	0.004490	0.000742	77	61	138
2	(1.0, 1.5)	919.227273	252.522348	888.5	0.003205	0.004270	0.000585	95	59	154
3	(1.5, 2.0)	955.713115	227.083975	1078.0	0.004066	0.004688	0.001061	64	58	122
4	(2.0, 2.5)	929.561404	248.768508	1028.0	0.004378	0.004975	0.001581	61	53	114
...	...	...	...	...	...	...	...	...	...	...
1200	(600.0, 600.5)	1031.739583	321.145294	1131.0	0.004865	0.006184	0.000956	78	18	96
1201	(600.5, 601.0)	974.734043	331.268146	1108.5	0.005336	0.006885	0.001814	73	21	94
1202	(601.0, 601.5)	993.343434	331.198755	1126.0	0.004406	0.006043	0.001021	80	19	99
1203	(601.5, 602.0)	995.216495	317.852994	1118.0	0.004635	0.006014	0.000910	74	23	97
1204	(602.0, 602.5)	1012.032258	342.128709	1116.0	0.005251	0.007278	0.001742	72	21	93

1205 rows × 10 columns

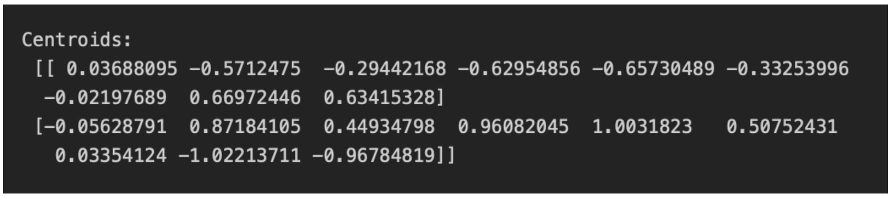
Fig 2: The output csv file from “dataset.ipybn” code.

3. *Data analysis*

The actual analysis of the dataset was done exploiting the Scikit.learn (or “sk.learn”) library, a machine learning library which features various algorithms, among which we have chosen to use K-Means, Random Forests and the K-Neighbours. At first, we exclusively relied on the K-Means algorithm for our project, however we ended up deciding to try the others aiming to obtain better scores in terms of accuracy and precision (performance metrics that will be later discussed).

- *K-Means Classifier*

After importing the csv file in the “k-means.ipynb” code, we proceeded with the necessary pre-processing requirements, which are performed by the “StandardScaler” command and the “scaler.fit_transform()”, in order to standardize the dataset features removing the mean and scaling to unit variance. Then, we used the algorithm, choosing k=2, since our goal was to analyse a binary event, and 200 iterations, because that’s the number we found to be appropriate, through trial and error, for the maximum accuracy and precision score obtainable. The corresponding clusters’ centroids calculated (with “kmeans_model.cluster_centers_”) were the ones in Fig 3.



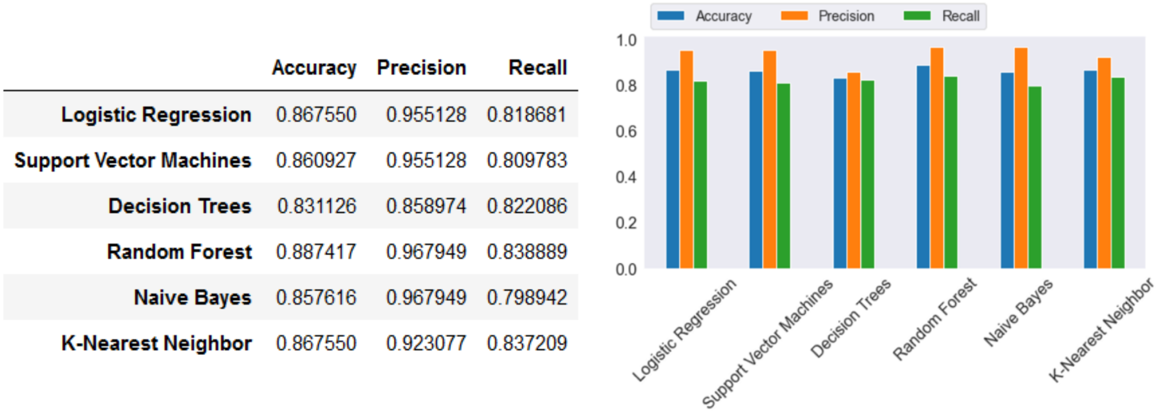
```
Centroids:
 [[ 0.03688095 -0.5712475  -0.29442168 -0.62954856 -0.65730489 -0.33253996
  -0.02197689  0.66972446  0.63415328]
 [-0.05628791  0.87184105  0.44934798  0.96082045  1.0031823   0.50752431
   0.03354124 -1.02213711 -0.96784819]]
```

Fig 3: The k-means centroids for 2 clusters and 200 iterations.

After this crucial step, each row of dataset, corresponding to a time interval, was assigned to a cluster, either 0 or 1, with the code “kmeans_model.labels_”: these results are available in the “Cluster.csv” file. Utilizing the “sklearn.metrics” library, we were able to evaluate the performance of the K-Means in our scenario, so we plotted the confusion matrix and printed the classification report, which was saved as “Metrics_report.csv”, regardless, both will be shown in the following section of the paper.

- *Random Forest Classifier & K-Neighbours Classifier*

In the file “other_ML_algorithms.ipynb”, we tested different ML algorithms at the same time, in this way, we were able to select the most promising ones and test them singularly. For this part, we followed a different procedure compared to before: we split the dataset into two parts, and the 75% of it was put into the training of the algorithm, the remaining 25% into testing it. We listed the algorithms we wanted to use in “models” and trained each of them with the code “models[key].fit(X_train, y_train)”, where y_train was the corresponding portion of the truth table. The results obtained were defined as “predictions”, and again they are arrays of 0 and 1 referring either to the absence or the presence of the person in front of the camera during the video call. The calculations of the accuracy, precision and recall scores were iterated through all the algorithms, so to acquire an overview of the test executions. The values obtained are displayed below in Fig 4.



	Accuracy	Precision	Recall
Logistic Regression	0.867550	0.955128	0.818681
Support Vector Machines	0.860927	0.955128	0.809783
Decision Trees	0.831126	0.858974	0.822086
Random Forest	0.887417	0.967949	0.838889
Naive Bayes	0.857616	0.967949	0.798942
K-Nearest Neighbor	0.867550	0.923077	0.837209

Fig 4: On the left, the evaluation parameters of the models displayed in a table format. On the right, same values in a bar chart format.

From this point, we selected Random Forest and K-Neighbours Classifiers for further analysis, using the code in the “any_ML_algorithm.ipynb”, which was created as a simple template where ML algorithms could be interchanged easily (just by editing the model choice).
In both cases, we found that splitting the dataset into 80% for training and 20% for testing (with “random_state = 85”), we were able to obtain even higher metrics scores than before, and the results will be now commented in the related section.
Moreover, we also plotted their learning curves, as well as we represented the scalability of the model and the performance.

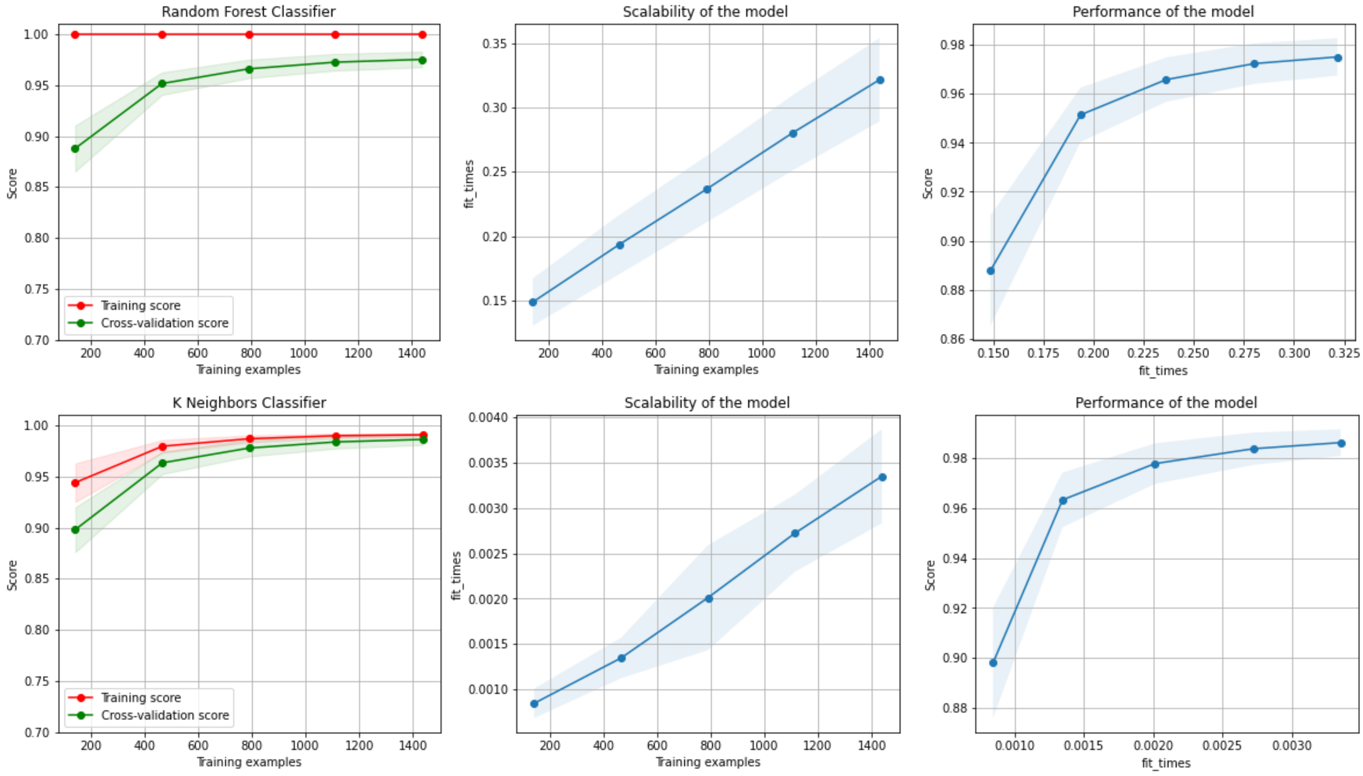


Fig 6: Learning curves, scalability, and performance of the two models, graphs described below.

Referring to the first column of graphs, we can see clearly that the training score is the maximum for the Random Forest Classifier and still around the maximum for the K-Neighbors. In the latter, the validation score could be increased with more training samples. Meanwhile, the plots in the second column show the times required by the models to train, with various sizes of training dataset. Finally, the ones in the third column reveal how much time was required to train the models for each training sizes.

4. Results and comments

Before commenting results and evaluating performances, we represented the ground truth data with NodeRed (see Fig 7), as suggested by the project description, and later used this blueprint to compare the results obtained, graphing them on top of it. To represent our data, we exploited the "function" node available in NodeRed, defining over the x axis the various timestamps, and over the y axis the respective values, namely 1 if the person was present or 0 if it wasn't.

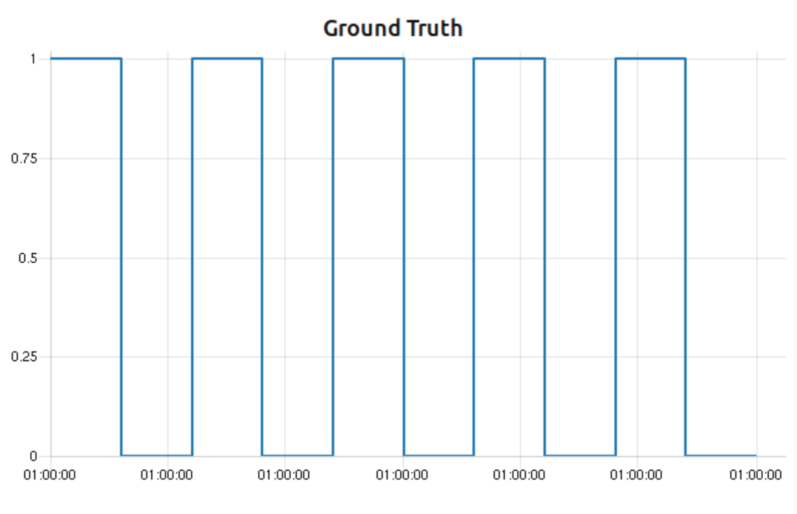


Fig 7: Ground truth obtained through NodeRed.

The following graph shows the results obtained by K-Means, where dark blue lines represent the ground truth data, while the ones in light blue are the values computed by the algorithm. As one may see, the results were satisfactory, but not as good as we would have hoped.

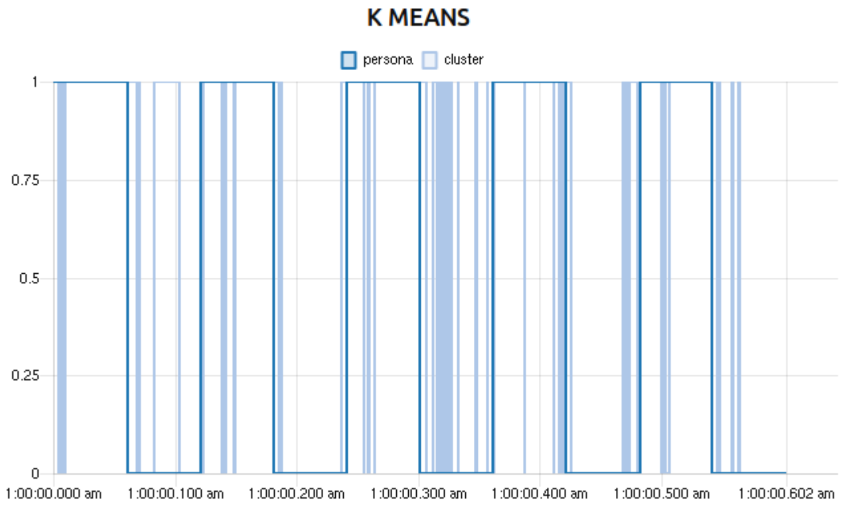


Fig 8: The outcome of the K-Means Classifier clustering overlapped on the ground truth.

According to the metrics, this first algorithm was shown to be 83.8% accurate and 94.6% precise. To better analyse what went wrong, we extracted the confusion matrix (Fig 9) related to the dataset which was composed of:

- True Positive(TP) = 565
- False Positive(FP) = 163
- True Negative(TN) = 445
- False Negative(FN) = 32

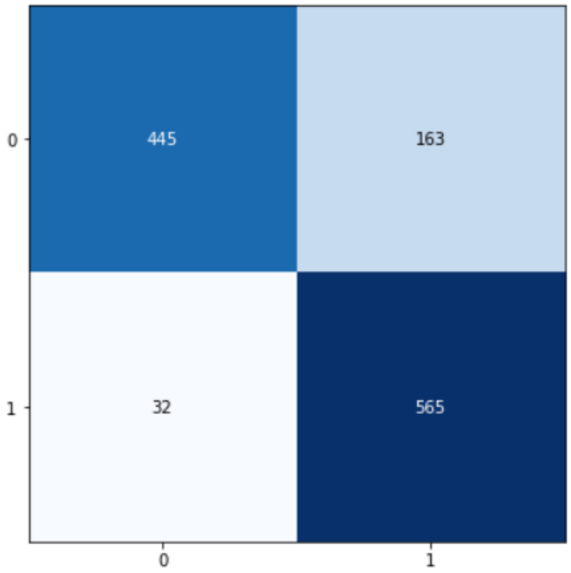
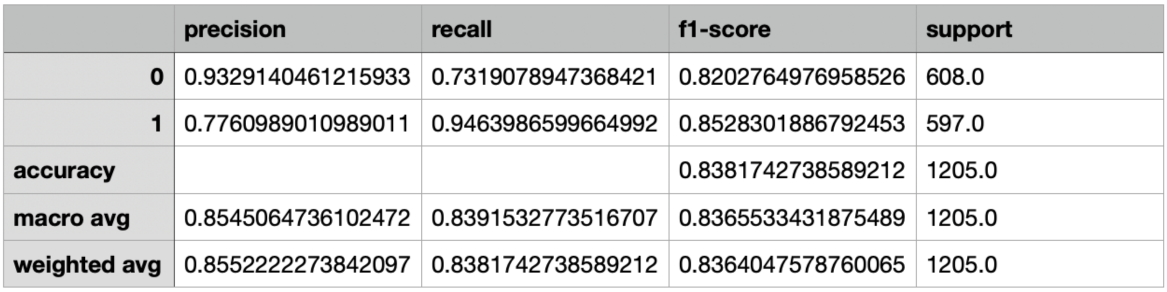


Fig 9: The confusion matrix of K-Means.

Those numbers confirmed the errors already noticed in the NodeRed chart, expressing that the K-Means Classifier worked quite fine when clustering the zeroes (binary definition of "false"), but revealed a tendency to predict false ones (binary definition of "true").



	precision	recall	f1-score	support
0	0.9329140461215933	0.7319078947368421	0.8202764976958526	608.0
1	0.7760989010989011	0.9463986599664992	0.8528301886792453	597.0
accuracy			0.8381742738589212	1205.0
macro avg	0.8545064736102472	0.8391532773516707	0.8365533431875489	1205.0
weighted avg	0.8552222273842097	0.8381742738589212	0.8364047578760065	1205.0

Fig 10: The metrics classification report for K-Means.

In the Fig 10 above, the metrics classification report is exhibited, with the following parameters, namely:

- Accuracy it the ratio of correctly predicted observation to the total observations.
- Precision is the ratio of correctly predicted positive observations to the total predicted positive observations.
- Recall is how many of the true positives were recalled (found).
- F1 Score is the weighted average of Precision and Recall.
- Macro Average computes F1 for each label and returns the average without considering the proportion for each label in the dataset.
- Weighted Average computes F1 for each label and returns the average considering the proportion for each label in the dataset.
- Support is the number of actual occurrences of the class in the specified dataset.

Moving to the Random Forest Classifiers results, the binary classification obtained an accuracy of 92,1% and precision of 93%. Below, the confusion matrix and its relevant numbers:

- True Positive(TP) = 107
- False Positive(FP) = 8
- True Negative(TN) = 115
- False Negative(FN) = 11

With the recall = 0.915 and the F1 score = 0,923.

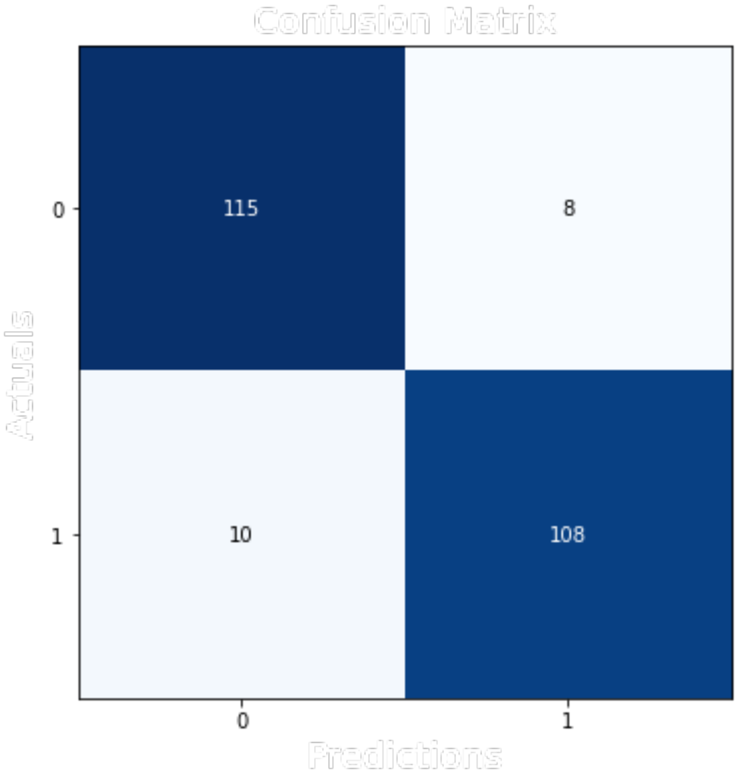


Fig 11: The confusion matrix of Random Forest.

Finally, the achievements of the K-Neighbors algorithm, 91,3% accurate and 93,7% precise:

- True Positive(TP) = 104
- False Positive(FP) = 7
- True Negative(TN) = 116
- False Negative(FN) = 14

With the recall = 0,881 and the F1 score = 0,908.

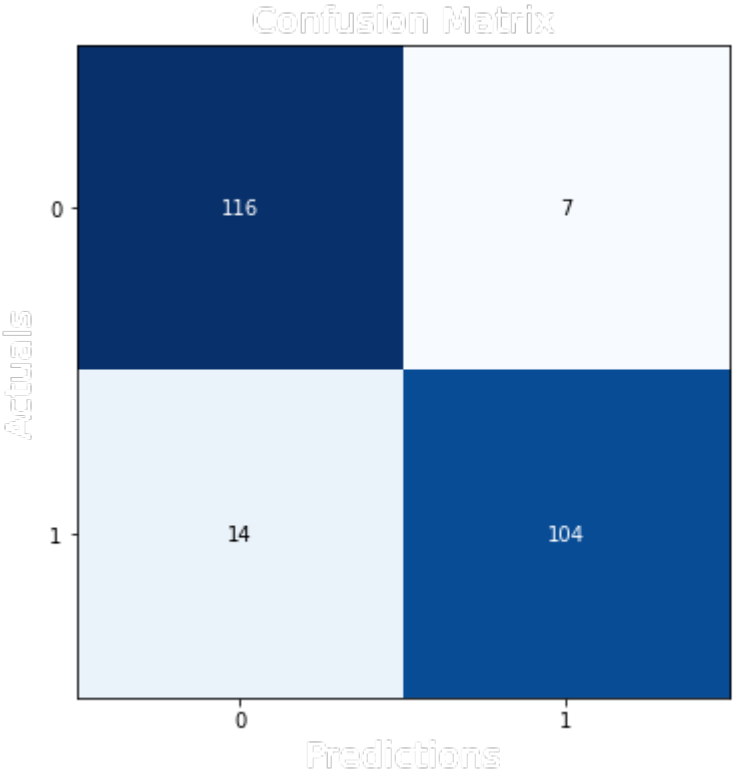


Fig 12: The confusion matrix of K-Neighbors.

In conclusion, taking into consideration the results found in the various cases, it’s clear how Random Forest and the K-Neighbors should work the best in this project scenario. Looking more closely, they showed different metrics results.

One may think that if we have high accuracy (Random Forest) then our model is best. Accuracy is, indeed, a great measure, but only when you have symmetric datasets, where values of false positive and false negatives are almost same. Therefore, one must look at other parameters to evaluate the performance of a model. On the other hand, high precision (K-Neighbors) relates to the low false positive rate.

The F1 score takes both false positives and false negatives into account. Intuitively, it is not as easy to understand, but F1 is usually more useful than accuracy, especially when having an uneven class distribution. Accuracy works best if false positives and false negatives have similar cost. If the cost of false positives and false negatives are very different, it's better to look at both precision and recall.

Our case gives no higher weight to a false positive or negative, so in this prospective, the K-Neighbors Classifier comes first. However, if we were to think to a practical application of the project, this outcome could be different: for instance, if the algorithm was supposed to work with a camera inserted in an alarm system, then one could safely say a false negative would have a much higher cost than a false positive, and the Random Forest Classifier (which has a greater F1 score) would be considered more appropriate.

Nevertheless, one thing that must be pointed out is that the longer the call lasted, the better the results we got (we just repeated the early described process but with shorter versions of the data capture file, hence those data are not reported here).

5. Code and files

https://github.com/chiaradraghini/spy_your_mate_project