University of Padua
Department of Information Engineering

Digital Forensics Report
Project 1

Identification of User Actions on Android Apps via Traffic Analysis

Author:
Simone Favaro

Teacher:
Simone Milani

8th July 2020
A.Y. 2019/2020

Contents

Chapter 1

Introduction

Nowadays a lot of internet content is consumed by people by means of smartphone applications. Even though the actual internet traffic is almost always encrypted thanks to security protocols (like SSL/TLS), it is possible to eavesdrop it in order to obtain the actual TCP/IP packets of information exchanged between servers and devices, and their exact length too (among with other similar and useful features).
The main goal of this project is to identify Android application user actions using few supervised machine learning techniques, starting from a dataset of sorted sniffed data payloads belonging to different application actions normally done during app usage.
In order to introduce this framework, few theoretical concepts are reviewed in the very next chapter. After those, it is described the approach used both for the data preprocessing and training/testing phase. Finally, results are shown and analyzed, discussed and confronted with each other and with the ones got in [1].
Moreover, it is fair to note that almost the entirety of this project scope is just to implement a python code which reflects all of the main ideas and concepts contained in [1]: for this matter, the analysis are similar to those made in the afromentioned paper.

Chapter 2

Theoretical backgroung

This chapter is a review of several theorical concepts used for the project accomplishment. It is discussed the structure of the given dataset, a kind of distance used for data clustering, the actual clustering method and few supervised machine learning techniques as well as analysis tools for measure their effectiveness.

2.1 User action dataset

The dataset given in [2] is the result of capturing network traffic from a smartphone, while an external script let it execute a pool of user actions belonging to different applications: in order to increase the data callected, the same script in charge to a specific app is run multiple times.
The data [1] ends up with, is a *.csv* file which stores a different *traffic flow* for each row, that is a time ordered sequence of TPC packets belonging to a specific user action: each action may have one or more flows, and same istances of the same action may be composed of different number of flows.
Concerning flow features, lot of them are used in [1] just to accomplish a first data filtering (like exclude some kind of flows based on domain and packet filtering, packet timeout). The fields which turn out to be useful in this project are the followings:

- *app* is the smartphone application that contains the flow;
- *action* is the user application that contains the flow;
- *sequence* is the metric that tells the times a script is ran for the same app;
- *flow number* is the cardinal number label given to each flow;
- *flow length* is the number of TCP packets the flow is composed;

- *packets_length_total* is the length [byte] of each packet belonging to the flow. The sign tells if the packet is incoming (negative sign) or outgoing (positive sign).

2.2 Time series

Since the *packet_length_total* field is actually a sequence of integers, [1] models it as a set of three time series [3] [1], which are a set of time dipendent data points:

- time series which consider each flow packets;
- time series which consider only incoming flow packets;
- time series which consider only outgoing flow packets.

This way of representing flows is useful, since it is possible to set a distance metric between them, using *Dynamic Time Warping*. Given two time series (also represented by discrete signals):

$$X = (x_1, x_2, ..., x_n)$$

$$Y = (y_1, y_2, ..., y_n)$$

it is possible to compute a cost matrix $C \in \mathbb{R}^{N \times M}$, where each entry $C_{j,k}$ refers to the distance[1] between x_j and y_k. Furthermore, a *warping path* is a sequence of entries, linking [1, 1] to [N, M], by creating a monotonic path of adjacent entries. The path cost is computed by summing all the entries in C belonging to the path itself. Moreover, the warping path which cost less is called *optimal warping path* and it is the measure used in [1] to evaluate distances between time series. In [1] is goes by $DTW(X, Y)$.
In order to compute the DTW optimal path, in this project is used the `fastdtw` python library [5], which include an approximate algorithm that provide a near optimal solution in $O(N)$ both time and memory complexity.

2.3 Agglomerative hierarchical clustering

Agglomerative hierarchical clustering [6] is a bottom-up clustering method, since it starts from a number of clusters equals to the input elements cardinality, and then it refines the guess merging items two by two, until the number of desired clusters is reached.
The merging procedure is straightforward: first all the distances between elements must be computed, then the two items less distant are merged in a single cluster pair, finally the distances from the cluster to all the other points (some of them will be also clusters in subsequent iterations) are computed;

[1] In [1], the distance metric used to compute the entries is Manhattan [4].

and so on.
The function which adresses clusters distance computation is the *linkage criteria*: its existance is necessary for the last part of the procedure, since it is not impossible to directly get distances of two non-point entity.
Note that neither the distance metric nor the linkage criteria are fixed: [1] uses DTW distance metric and the so called *average distance* linkage criteria: given two clusters u and v and the metric DTW, $d()$ is computed in the follong way:

$$d(u,v) = \sum_{\substack{1 \leq i \leq n \\ 1 \leq j \leq m}} \frac{d(u[i], v[j])}{|u| * |v|}$$

In order to compute this kind of clustering, in the project it is used the `scipy.cluster.hierarchy` [7] package, which provides an algorithm of time complexity equals to $O(N^2)$.

2.4 Supervised machine learning techniques

Two different kind of learning models for classification are considered, in order to meet the project goal. They are supervised [8], which means that the training of the classifier is based on a dataset provided with a correct labeling classification[2]. The classifier works in two phases:

- *Training phase* is the computing of the most efficient partitioning, based on the labeled dataset;
- *Testing phase* is the measure of the classifier accuracy[3].

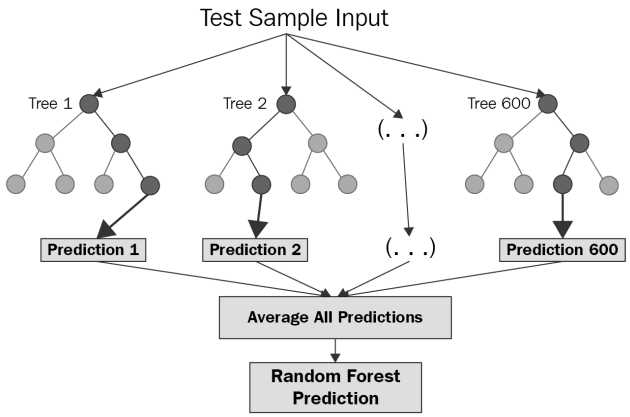
2.4.1 Random forest

Random forest classifier [9] works on the idea that few *weak classifier* decisions combined, generate a *strong classifier* decision. After the model is trained, weak classifiers (also called *decision trees*) take a random partition of the test sample, and output a guess. The model takes the average of all the guesses and outputs the final decision. The visual representation of the process is shown in figure 2.1.

This project uses the `sklearn.ensemble.RandomForestClassifier` [10] package, which provides lot of possibile customisation parameters concerning the training model settings. However, the only one considered[4] is the number of estimators `n_estimators` which sets the decision tree cardinality of the model.

[2] This is in fact the project case, assuming we make use of the dataset we looked into before.
[3] The set of items used for training is different from the one used for testing.
[4] The other parameters were good on the standard setting as far the project needs were concerned.

Figure 2.1: Workflow of a random forest classifier, using 600 weak learners.



2.4.2 Deep neural network

Deep neural network classifier [8] is based on the combination of multiple layers of binary classifiers called *perceptron* or *neurons* [8], which take as input a feature vector and returns a binary output, like a linear separator:

$$c = \begin{cases} 0 & \text{if} \quad \sum_i w_i x_i - b = \mathbf{w}^T \mathbf{x} - b \leq 0 \\ 1 & \text{if} \quad \sum_i w_i x_i - b = \mathbf{w}^T \mathbf{x} - b > 0 \end{cases}$$

The activation function is a function applied to $z = \mathbf{w}^T \mathbf{x} - b$ in order to modify the neuron activation threshold. The functions used in this project are the followings: [8] [11]:

- *identity*, which leave the neuron fire according to the step function;
- *sigmoid*, which is: $\frac{1}{1+e^{-z}}$
- *tanh* which is: $\frac{2}{1+e^{-2z}} - 1$
- *rectified linear unit (relu)* which is: $\begin{cases} 0 & \text{if} \quad z < 0 \\ z & \text{if} \quad z \geq 0 \end{cases}$

This project uses the `sklearn neural_network.MLPClassifier` [11] package, which provide lot of customizable parameters including the `activation` one, that sets the activation function.

2.5 Classification metrics

The following are the definitions of the classification metrics used in this project, in order to analyse classifiers accuracy. These definitions are based on the notion of *True Positives* (TP), *False Positives* (FP) and *False Negatives* (FN) [12].

- Precision = $\frac{TP}{TP+FP}$
- Recall = $\frac{TP}{TP+FN}$
- F-measure = $\frac{2 \cdot TP}{2 \cdot TP+FP+FN}$[5]

Note that these metrics are applicable both on a specific class and the entire model (that will be just the average of computations for each single class) [12].

Finally the *confusion matrix* is an quick way to visually evaluate the performance of each class in a classification model, as far recall metric and class specific FNs are concerned [1]:

- each row and column is labeled with a class;
- the diagonal entry of each row represents the recall value of the specific class on that matrix row;
- the other entries on the same row are the FN ratios in realation to the corresponding class present in the column of the matrix.

[5] that is also equals to $\frac{2 \cdot precision \cdot recall}{precision+recall}$

Chapter 3

Framework

The framewok used in this project is divided in two main parts: the data preprocessing, that is necessary in order to get data ready for the ML computatons, and the actual model training and testing, along with classification results.

3.1 Data preprocessing

The data preprocessing main goal is the convertion of user actions present in the dataset to elements ready to be processed by ML models. The so called elements go by the specific names of *feature vectors* [8]: they are n-dimension vectors $x^T = [x_1, x_2, ..., x_n]^T \in \mathbb{R}^n$ where the domain is called *descriptor space* and each entry value stores information useful for ML classifications [8].
The entire procedure is composed by the following parts:

- *Flows dataset processing* creates a new flow dataset concerning just data belonging to a single application, where flows are expressed in a convinient way;
- *Clustering leaders computing* clusters flows belonging to the prevous dataset and finds a leader for each cluster;
- *Actions dataset processing* creates a dataset similar to the first one, but associating each action to the list of respective flows. It also provides labels for the feature vectors;
- *Feature vectors computing* outputs a feature vector for each user action belonging to the previous dataset.

The following sections further describe flows dataset processing, clustering leaders computing and feature vectors computing.

3.1.1 Flows dataset processing

The scope of this processing is to produce a flows dataset similar to the initial one, but easier to handle for clustering. Firsty, are taken only flows from a specific app and only the *packets_length_total* column. Then a flow processing is computed, extracting for each one of them just the interval that most likely contains information useful for the clustering itself.[1]

3.1.2 Clustering leaders computing

Starting from the flows dataset, it is computed the agglomerative hierarchical clustering, using the DTW metric and the average distance linkage criteria[2]. The optimal number of clusters has already evaluated in [1], so in this project it is chosen according to the afromentioned paper.
Finally, leaders are the most representative flows for each cluster: in order to compute these, it is used again the DTW metric to find the element which minimize the overall distance from all the other cluster flows:

$$\arg\min_{f_i \in C} \left(\sum_{j=1}^{n} dist(f_i, f_j) \right)$$

3.1.3 Feature vectors computing

Features vectors are computed according to the cluster that each action flow belongs to: the vector dimension is exactly the clusters cardinality, and for each action flow it is measured the distance between it and all leaders; clearly the nearer leader represents the flow's cluster.
Since this is repeated for each action flow, at the end of the procedure every one of them is assigned to one cluster. Moreover, every feature vector entry represent a cluster, so each entry stores the number of flows belonging to that cluster.

3.2 Model training and testing

The second part of the framework aims to take feature vectors and trains/tests ML models from these. Since the user action identification is a ML multiclass classification problem, this project makes use of two different ML classifiers: random forest and deep neural network. For each one of them it is applied the some following procedure:

[1] The intervals settings are already discussed in [1], so they are taken for granted. Moreover, [1] uses a weighted version for that (it assigns for each interval a certain weight), while in this project is used just the most weighted interval, in order to lighten further computations.

[2] In [1] it is used a similar metric which contains the interval weights of the three time series type discussed in section 2.2. However, since the computer where the scripts are run has not lot of computational power, this project valued more the cardinality of flows clustered then the metric complexity.

- *Split train and test datasets* divides features vector dataset (and respective labels) in two subsets in order to use the majority of them (70%) for the model training and the rest for the testing (30%);
- *Build classifiers* chooses the best parameters[3] in order to build the best classifier (according to its testing score);
- *Classification metrics computing* outputs precision, recall and F-measure metrics belonging to the best classifier build before;
- *Confusion matrix computing* outputs the confusion matrix related the the same classifier.

[3] Chosen from the ones already explained in section 2.4.

Chapter 4

Results and analysis

The framework is implemented with python: `dataprocessing.py` contains the preprocessing part, while `results.py` contains the ML training, testing and the actual results.
The identification analysis are done separately for each application and they take into account a pool of 7 user actions[1]; this is done for 4 different apps: Facebook, Twitter, Gmail and Tumblr. The following table contains user actions considered for each app:

Facebook	**Twitter**	**Gmail**	**Tumblr**
open facebook	open Twitter	open gmail	open tumblr
user profile selection	tweets selection	sending mail	refresh home
post button selection	DM selection	reply selection	search page
send message selection	send selection	sending mail reply	user page
menu message selection	contact selection	chats selection	user likes
status post selection	back to home	delete selection	following page
status selection	writing tweet	inbox selection	new post

As far the flows intervals taken into consideration, the cluster cardinalities and the number of feature vectors[2], they are different for each application[3]. The following tables show them all:

Facebook	[1, 12]
Twitter	[7, 10]
Gmail	[1, 6]
Tumblr	[1, 10]

Facebook	200
Twitter	70
Gmail	190
Tumblr	100

Facebook	1000
Twitter	5000
Gmail	5000
Tumblr	5000

[1]The *other* action is a common label for all the other application actions not take into consideration (not labeled).

[2]This value is bounded just to the machine computational power where scripts were executed.

[3]Note that for Tumblr it is used just simple intuition in order to guess the cluster cardinality and interval values, since there are not guidelines present in [1]

The few sections ahead present results grouped by application and the respective analysis. They concern: a comparison between few ML settings for each classifier (in order to find the more accurate); metrics evaluation and confusion matrix (in order to evaluate the classifier accuracy for each single user action). Also, a comment on the classifiers performance and a comparison with results from [1]. Further discussion topics are present in the next chapter.

4.1 Facebook

4.1.1 Random forest classifier

As noted in figure 4.1, there is not a clear linear trend between `n_estimators` and the respective precision score[4]. Since the best score is achieved for `n_estimators` = 60, this value is chosen for further computations based on this classifier.

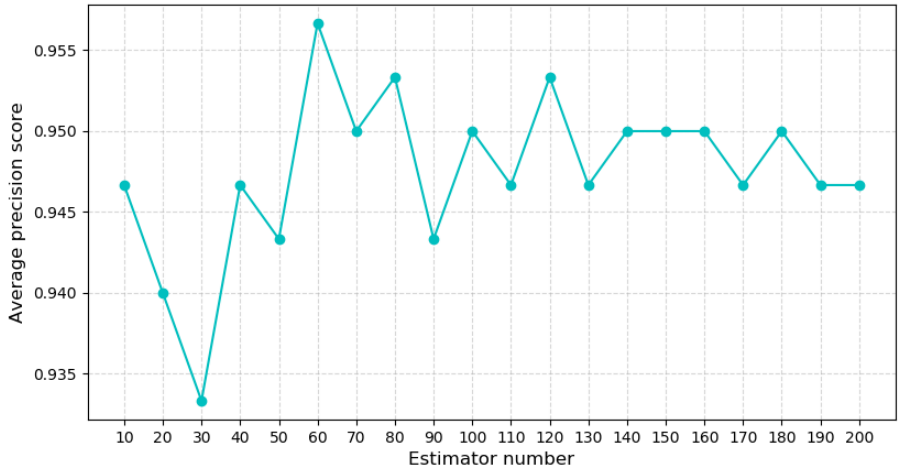


Figure 4.1: Precision score of Facebook random forest classifier in relation to the `n_estimators` parameter.

Table 4.1 and figure 4.2 display some classification metrics by means of precision, recall, F-measure and confusion matrix.
It is clear that not all user actions are correcly classified: even though precision and F-measure are good for almost all of them except *menu message selection*, the recall score is the lowest one (that is reflected in the confusion matrix entries) since lot of actions were wrongly classified as *other*, which is understandable because the latter label contains lot of different action types inside.
Finally there is not a strong correlation between [1] scores and the project's

[4]Note that this observation is true for this project only: no general hypotheys are done in this section.

(except for *open user profile*), even though the user action analysed are the same.

Actions	**Precision**	**Recall**	**F-measure**
open facebook	1	1	1
user profile selection	1	0.67	0.80
post button selection	1	1	1
send message selection	1	0.78	0.88
menu message selection	1	0.14	0.25
status post selection	1	0.71	0.83
status selection	1	1	1
other	0.95	1	0.98

Table 4.1: Facebook random forest classification metrics, related to each user action.

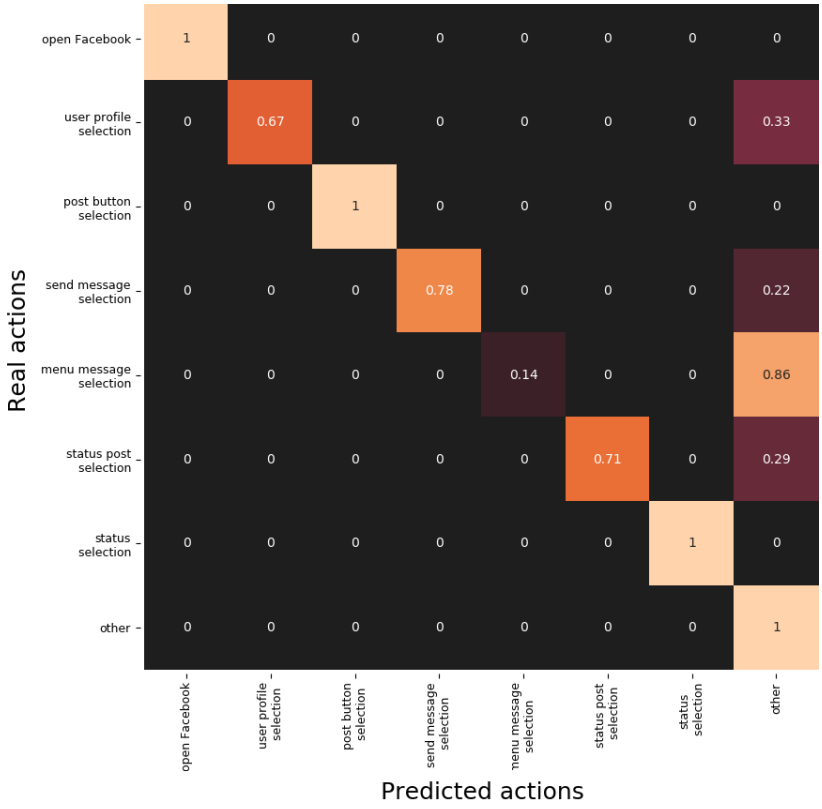


Figure 4.2: Facebook confusion matrix related to the random forest classifier.

4.1.2 Deep neural network classifier

The deep neural network classification give overall a similar results trend to the random forest one: few actions are classified better (like *status post selection*) and other worse (like *user profile selection* and *status selection*). Results are displayed below in figure 4.3 and table 4.2. 4.2.

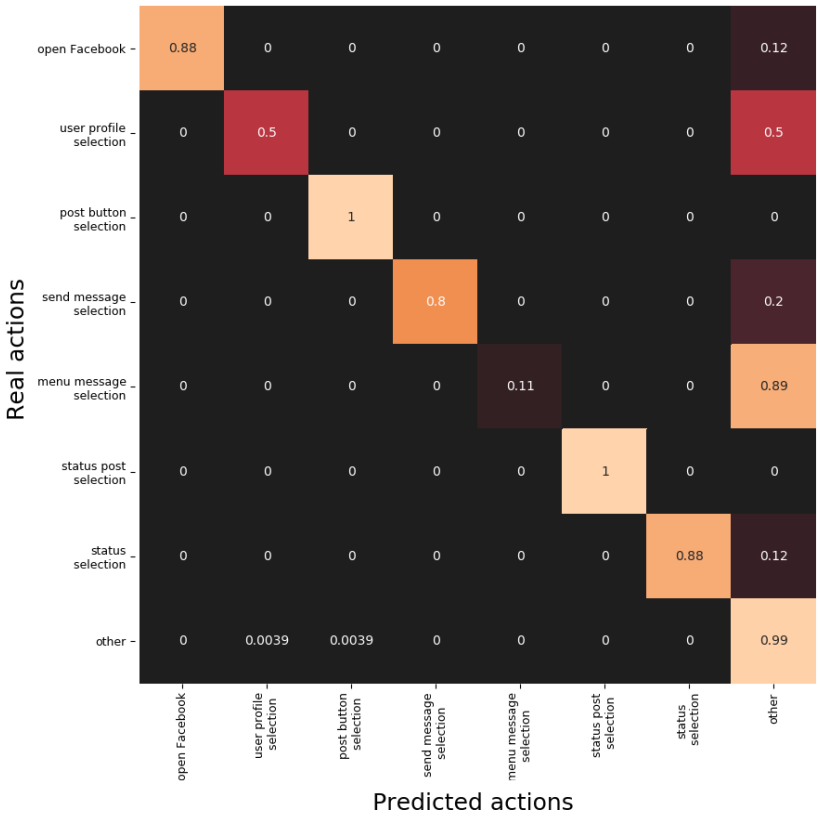


Figure 4.3: Facebook confusion matrix related to the deep neural network classifier.

Actions	Precision	Recall	F-measure
open facebook	1	0.88	0.93
user profile selection	0.77	1	0.57
post button selection	0.88	0.5	0.93
send message selection	1	1	0.89
menu message selection	1	0.80	0.2
status post selection	1	1	1
status selection	1	0.88	0.93
other	0.95	0.99	0.97

Table 4.2: Facebook deep neural network classification metrics, related to each user action.

4.2 Twitter

4.2.1 Random forest classifier

As noted in figure 4.4, seems that the higher `n_estimators`, the low is the precision score (even though there are some strange low peaks for `n_estimators` = 20 and 50). Since the best score is achieved for `n_estimators` = 30, this value is chosen for further computations based on the random forest classifier.

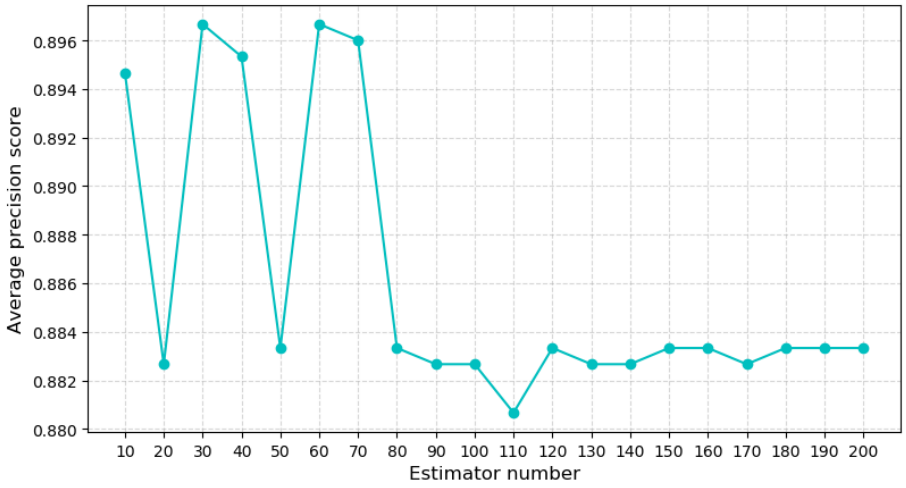


Figure 4.4: Precision score of Twitter random forest classifier in relation to the `n_estimators` parameter.

Table 4.3 and figure 4.5 display classification metrics and confusion matrix. All user action are overall correcly classified except for *back to home* and *send selection*. Even for Twitter, the recall score is the lowest since lot of actions were wrongly classified as *other* and few *writing tweet* actions were mistaken for *send selection*: the former has a low recall score (same problem seen in few Facebook

actions), while the latter has a low precision and an high recall. This means that the classification give lot of FPs and it makes sense inspecting the matrix, because it is clear that a lot of *writing tweet* actions were wrongly labeled as *send selection*[5].
Finally there is not a strong correlation between [1] scores and the project's.

Actions	**Precision**	**Recall**	**F-measure**
open Twitter	1	1	1
tweets selection	0.99	0.96	0.97
DM selection	0.95	1	0.97
send selection	0.46	0.91	0.61
contact selection	0.98	0.95	0.97
back to home	1	0.07	0.13
writing tweet	0.89	0.81	0.85
other	0.92	0.93	0.92

Table 4.3: Twitter random forest classification metrics, related to each user action.

[5]The same problem is obviously present also for the *other* class. The metrics do not highlight it because how much this actions label is frequent in the feature vector dataset.

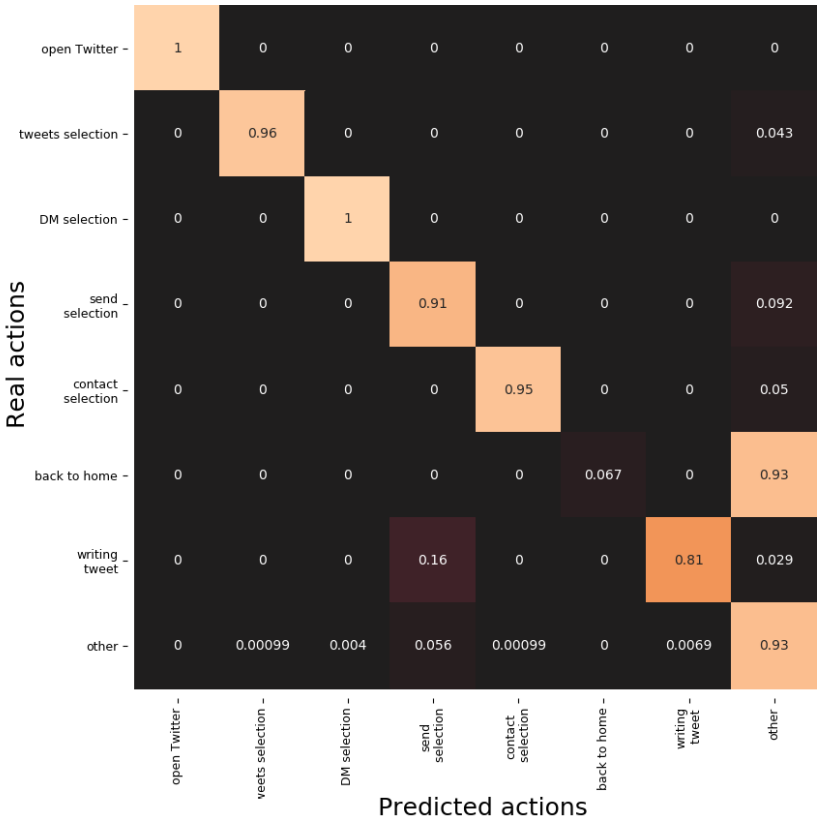


Figure 4.5: Twitter confusion matrix related to the random forest classifier.

4.2.2 Deep neural network classifier

The deep neural network classification give overall a similar results trend to the random forest one. Results are displayed below in figure 4.6 and table 4.4.

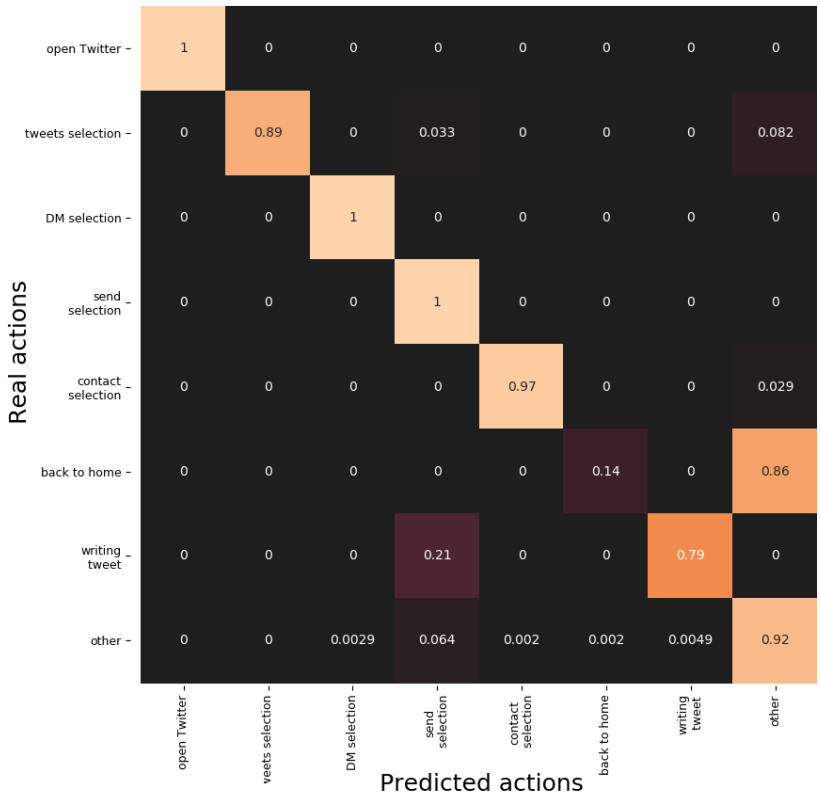


Figure 4.6: Twitter confusion matrix related to the deep neural network classifier.

Actions	**Precision**	**Recall**	**F-measure**
open Twitter	1	1	1
tweets selection	1	0.89	0.94
DM selection	0.97	1	0.98
send selection	0.48	1	0.65
contact selection	0.97	0.97	0.97
back to home	10.82	0.14	0.24
writing tweet	0.91	0.79	0.84
other	0.94	0.92	0.93

Table 4.4: Twitter deep neural network classification metrics, related to each user action.

4.3 Gmail

4.3.1 Random forest classifier

As noted in figure 4.7, there is not a clear monotonic trend that correlates the `n_estimators` and precision score. Since the best score is achieved for the `n_estimators` = 50 peak, this value is chosen for further computations based on this classifier.

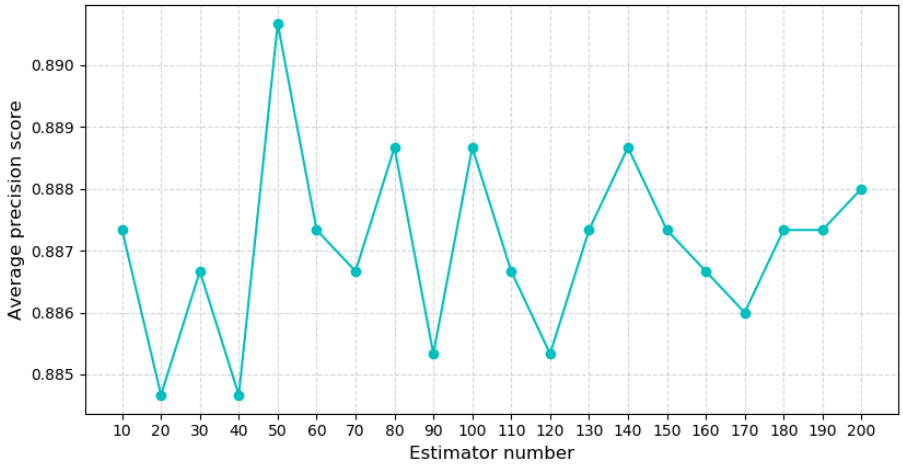


Figure 4.7: Precision score of Gmail random forest classifier in relation to the `n_estimators` parameter.

Table 4.5 and figure 4.8 display classification metrics and confusion matrix. All user action were overall correcly classified except for *chat selection*, *inbox selection* and *delete selection*, because of the same FNs problem related to *other* action.
Finally there is not a strong correlation between [1] scores and the project's.

Actions	**Precision**	**Recall**	**F-measure**
open gmail	0.80	0.95	0.87
sending mail	0.92	0.98	0.95
reply selection	0.81	1	0.90
sending mail reply	0.97	0.88	0.92
chats selection	0.75	0.79	0.77
delete selection	0.93	0.59	0.72
inbox selection	0.73	0.37	0.49
other	0.91	0.92	0.91

Table 4.5: Gmail random forest classification metrics, related to each user action.

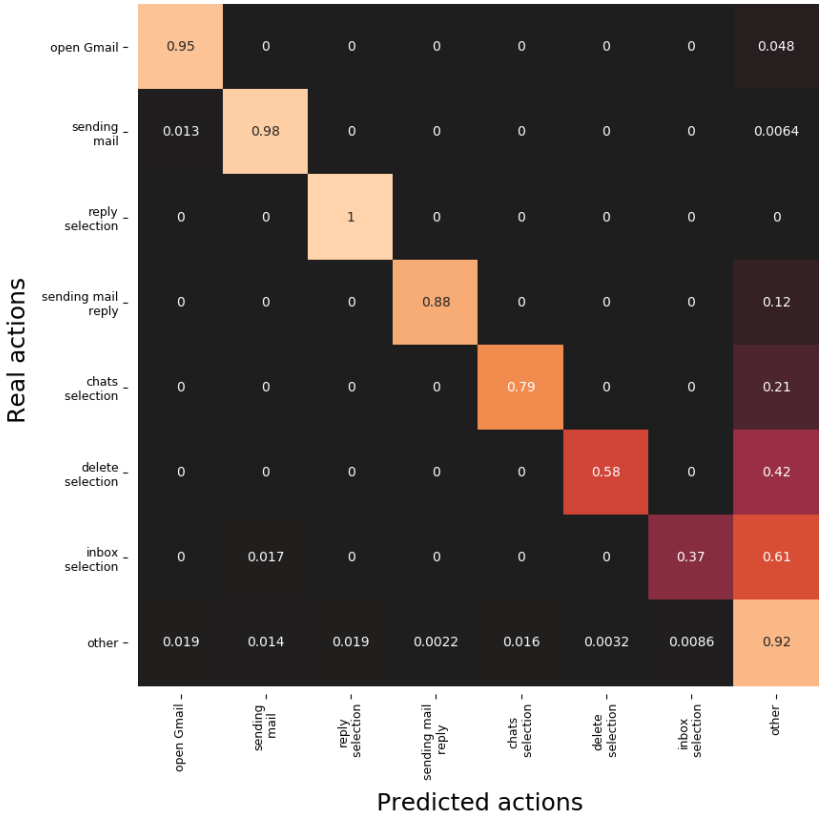


Figure 4.8: Gmail confusion matrix related to the random forest classifier.

4.3.2 Deep neural network classifier

The deep neural network classification give a similar results trend to the random forest one, but all metric scores are better, except for *delete selection* action and some other minor discrepancies. The results are displayed below in figure 4.9 and table 4.6.

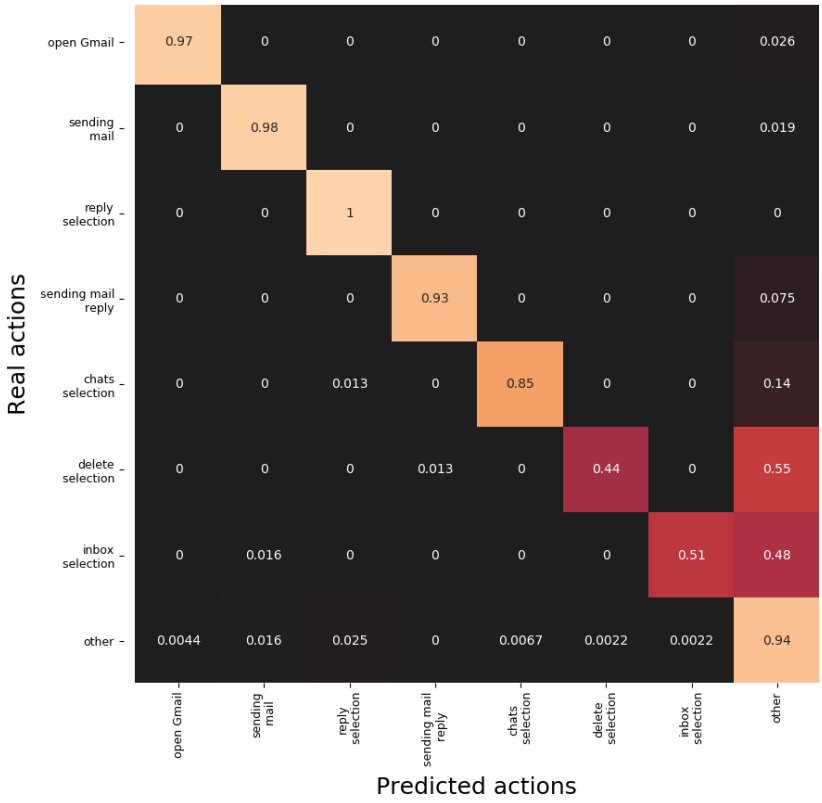


Figure 4.9: Gmail confusion matrix related to the deep neural network classifier.

Actions	**Precision**	**Recall**	**F-measure**
open gmail	0.95	0.97	0.96
sending mail	0.92	0.98	0.95
reply selection	0.76	1	0.86
sending mail reply	0.98	0.93	0.95
chats selection	0.92	0.85	0.88
delete selection	0.94	0.44	0.60
inbox selection	0.94	0.51	0.66
other	0.90	0.94	0.92

Table 4.6: Gmail deep neural network classification metrics, related to each user action.

4.4 Tumblr

4.4.1 Random forest classifier

As noted in figure 4.10, there is somewhat a linear monotonic trend that links `n_estimators` and precision score. However, best score is achieved for the peak on `n_estimators` = 70, so this value is chosen for further computations based on this classifier.

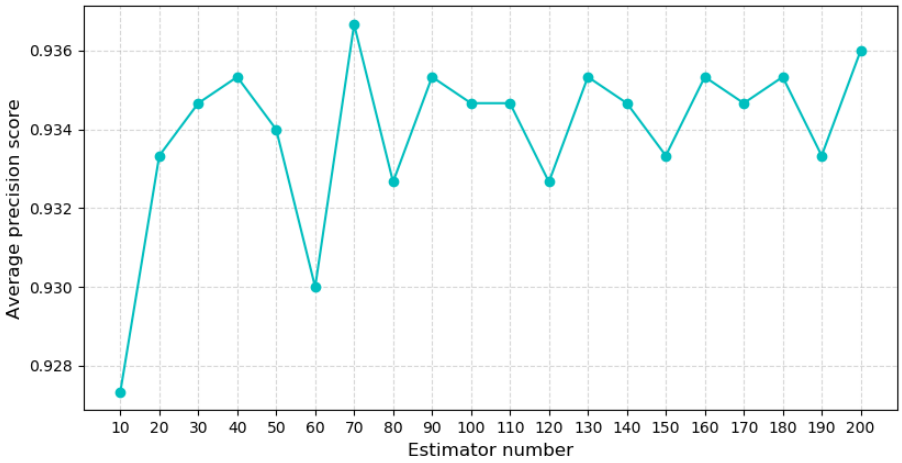


Figure 4.10: Precision score of Tumblr random forest classifier in relation to the `n_estimators` parameter.

Table 4.7 and figure 4.11 display classification metrics and confusion matrix.
Even for this application, the only significant problem is related to FNs regarding *other* class, which drastically lowers the score of *refresh home* and *new post* actions; *user page* action has also few FNs related to the *new post* which lowers the latter action's accuracy. Finally there is not a strong correlation between [1] scores and the project's.

Actions	**Precision**	**Recall**	**F-measure**
open tumblr	1	0.96	0.98
refresh home	0.5	0.37	0.69
search page	0.94	0.97	0.92
user page	0.94	0.67	0.80
user likes	1	0.97	0.98
following page	1	1	1
new post	0.55	0.28	0.37
other	0.94	0.99	0.96

Table 4.7: Tumblr random forest classification metrics, related to each user action.

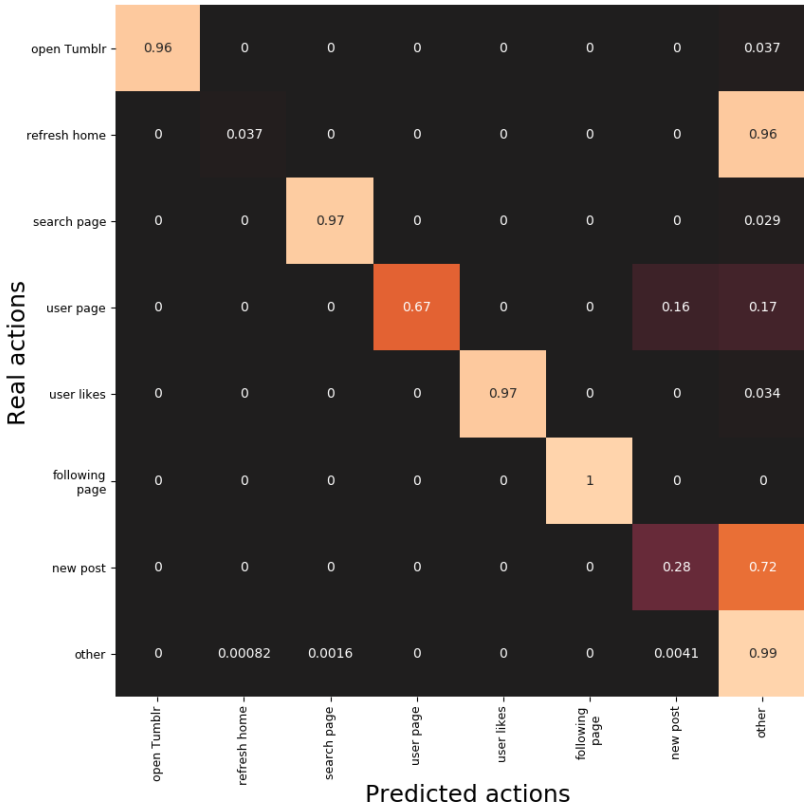


Figure 4.11: Tumblr confusion matrix related to the random forest classifier.

4.4.2 Deep neural network classifier

The deep neural network classification give overall a similar results trend to the random forest one, except for the *new post* action which metrics score better. Results are displayed below in figure 4.12 and table 4.8.

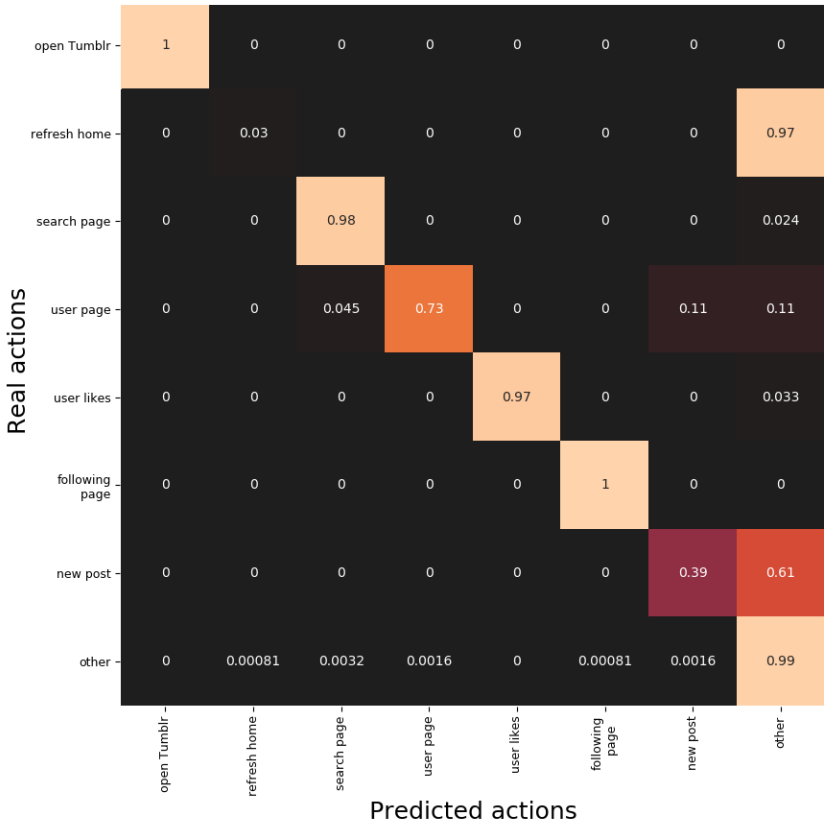


Figure 4.12: Tumblr confusion matrix related to the deep neural network classifier.

Actions	**Precision**	**Recall**	**F-measure**
open tumblr	1	1	1
refresh home	0.5	0.03	0.57
search page	0.87	0.98	0.92
user page	0.94	0.73	0.82
user likes	1	0.97	0.98
following page	0.97	1	0.99
new post	0.74	0.39	0.51
other	0.95	0.99	0.97

Table 4.8: Tumblr deep neural network classification metrics, related to each user action.

Chapter 5

Conclusions

This project allows to classify user actions belonging to different Android applications, by using clustering and different ML strategies, knowing only the length of the packets exchanged during communications.
Even if the core ideas implementation are taken from [1], results are slightly worse. In fact the action pool, datasets length (especially for Facebook features vectors) and clustering method are downgraded in efficiency, according to the small computational power avaiable to run project scripts. Moreover, model train and test has an aleatory bias since the dataset is spitted randomly in train and test sets.

As far a possible upgrade on the framework classification accuracy, if it is taken into consideration just a subset of the whole classified user actions, results are drastically better, as displayed in table 5.1[1][2].

[1] Only actions which scored a F-measure greater than 0.9 are chosen.

[2] Actually the *other* action scores are not correct: it is included as a lower bound under the reasonable assumption that an hypothetic classification with less labels upgrades its scores.

App	Action	Precision	Recall	F-measure
Facebook	open Facebook	1	0.88	0.93
	post button selection	1	0.5	0.93
	status post selection	1	1	1
	status selection	1	0.88	0.93
	other	0.95	0.99	0.97
	Average	**0.99**	**0.85**	**0.95**
Twitter	open Twitter	1	1	1
	tweets selection	0.99	0.96	0.97
	DM selection	0.95	1	0.97
	contact selection	0.98	0.95	0.97
	other	0.92	0.93	0.92
	Average	**0.97**	**0.97**	**0.97**
Gmail	open Gmail	0.95	0.97	0.96
	sending mail	0.92	0.98	0.95
	sending mail reply	0.98	0.93	0.95
	other	0.90	0.94	0.92
	Average	**0.94**	**0.96**	**0.95**
Tumblr	open Tumblr	1	1	1
	search page	0.87	0.98	0.92
	user likes	1	0.97	0.98
	following page	0.97	1	0.99
	other	0.95	0.99	0.97
	Average	**0.96**	**0.99**	**0.97**

Table 5.1: Classification metrics for a subset of the whole user action analysed in the framework.

In conclusion, in this project it is developed a framework which is actually really effective for the classification of a substantial number of user actions. The efficiency drops as more actions are taken into consideration.

Bibliography

[1] M. Conti, L. V. Mancini, R. Spolaor, and N. V. Verde, "Can't you hear me knocking: Identification of user actions on android apps via traffic analysis," in *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, pp. 297–304, 2015.

[2] M. Conti, L. V. Mancini, R. Spolaor, and N. V. Verde, "Analyzing android encrypted network traffic to identify user actions," 2014.

[3] Wikipedia contributors, "Time series — Wikipedia, the free encyclopedia," 2020. [Online; accessed 10-July-2020].

[4] Wikipedia contributors, "Taxicab geometry — Wikipedia, the free encyclopedia," 2020. [Online; accessed 10-July-2020].

[5] slaypni, "fastdtw 0.3.4," 2020.

[6] Wikipedia contributors, "Hierarchical clustering — Wikipedia, the free encyclopedia," 2020. [Online; accessed 10-July-2020].

[7] T. S. community, "Hierarchical clustering (scipy.cluster.hierarchy)," 2020.

[8] S. Milani, "Notes on machine learning strategies in computer vision." University Lecture, 2018.

[9] Wikipedia contributors, "Random forest — Wikipedia, the free encyclopedia," 2020. [Online; accessed 10-July-2020].

[10] scikit-learn developers, "3.2.4.3.1. sklearn.ensemble.randomforestclassifier," 2020.

[11] scikit-learn developers, "sklearn.neuralnetwork.mlpclassifier," 2020.

[12] scikit-learn developers, "sklearn.base: Base classes and utility functions," 2020.