INTRODUZIONE

Ognuno di noi oggigiorno, possiede uno smartphone con cui può messaggiare, chiamare, scaricare applicazioni da internet liberamente.
Ciò che però ignoriamo, è la possibilità che all'interno del nostro smartphone, ci siano software indesiderati come per esempio malware e virus che agiscono indisturbatamente raccogliendo informazioni sensibili e private.
Il nostro scopo è quello di individuare queste applicazioni malevoli tramite il machine learning.

1. OBIETTIVI

L'obiettivo principale che ci siamo posti è stato quello di riuscire a riconoscere quali applicazioni android sono in realtà dei malware.
In particolare abbiamo deciso di compiere due analisi differenti, una "statica" e una "dinamica". Quella statica è definita così perché può essere svolta prima dell'esecuzione dell'applicazione e quindi prima di essere eventualmente "infettati". Mentre quella dinamica si basa sullo studio di un applicazione dopo che è stata eseguita.

2. STUDI SIMILI

Attualmente sono già state realizzate diverse applicazioni che sfruttando l'intelligenza artificiale classificano un'applicazione come elemento nocivo oppure no in base a specifici parametri.

Questi sono alcuni esempi:

- https://hal.inria.fr/hal-01812448/document

  In questo progetto, ciò che si è andato ad analizzare è stato il traffico che veniva generato da un'applicazione nella rete e l'utilizzo dei processori della CPU.
  Questo perché alcuni malware erano facilmente individuabili analizzando solamente il traffico che generavano via rete mentre altri, un po' più difficili da trovare, potevano essere identificati solamente analizzando l'utilizzo della CPU, memoria RAM etc..

- https://www.researchgate.net/profile/Yajin-Zhou-2/publication/267787299_Hey_You_Get_Off_of_My_Market_Detecting_Malicious_Apps_in_Official_and_Alternative_Android_Markets/links/5b348b050f7e9b0df5d2a119/Hey-You-Get-Off-of-My-Market-Detecting-Malicious-Apps-in-Official-and-Alternative-Android-Markets.pdf".

  Altra ricerca che è stata realizzata per prevenire attacchi malware sui dispositivi android che ci è sembrata utile andare ad analizzare è "DroidRanger

DroidRanger è uno schema che è stato costruito in base alle seguenti caratteristiche:

1. Permission-Based Filtering
    a. Analizzando i permessi che ogni applicazione richiede si può andare a fare una stima e vedere se c'è un mismatch tra permessi necessari e richiedenti.
2. Behavioral footprint matching
    a. Anziché analizzare i permessi di un'applicazione si analizza il comportamento nel momento in cui viene eseguita/lanciata.
3. Heuristic-Based Filtering
    a. Si basa su due analisi euristiche:
        i. La prima va ad analizzare il caricamento dinamico del codice binario di Java da un sito web esterno e ne valuta il comportamento
        ii. La seconda va ad analizzare il caricamento del codice nativo localmente e viene considerata malevole un'applicazione che cerca di caricare il codice in una directory diversa da quella default di Linux (lib/armeabi)
4. Dynamic execution monitoring
    a. Ispezione in modo dinamico del file Manifest.xml

- https://www.researchgate.net/publication/342614130_A_Review_of_Android_Malware_Detection_Approaches_Based_on_Machine_Learning

  Anche in questo progetto si è sfruttato il machine learning per individuare malware attraverso lo studio dei permessi che un'applicazione richiede.

3. STRUMENTI UTILIZZATI

I principali strumenti che abbiamo utilizzato sono stati:

1. Python come linguaggio di scripting
2. Jupyter Notebook come IDLE per esecuzione del codice
3. Pandas, NumPy, MatPlotLib, SkLearn : librerie di Python

4. DATASET UTILIZZATI

Per il nostro progetto ci siamo concentrati su 2 dataset che abbiamo trovato online da due siti differenti.

Il primo dataset lo abbiamo ottenuto da kaggle.com, sito contenente diversi dataset che affrontano qualsiasi tipo di tematica.

Il dataset che abbiamo scelto per il nostro progetto contiene informazioni relative ai permessi che ogni applicazione richiede ogni volta che viene utilizzata. Questi permessi sono salvati in formato booleano.
Infatti per poter funzionare correttamente ogni applicazione ha bisogna di ricevere l'autorizzazione da parte dell'utente per i permessi che richiede.
Questo primo dataset si basa su una lettura "statica" delle informazioni delle app, cioè ottiene queste informazioni senza avviare l'applicazione, controllando solamente le autorizzazioni che richiede, prima di installarla.
Il link di questo dataset è il seguente:
https://www.kaggle.com/shashwatwork/android-malware-dataset-for-machine-learning?select=drebin-215-dataset-5560malware-9476-benign.csv

	transact	onServiceConnected	bindService	attachInterface	ServiceConnection
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
...	...	...	...	...	...
15026	1	1	1	1	1
15027	0	0	0	0	0
15028	0	0	0	0	0
15029	1	1	1	1	1
15030	1	1	1	1	1

Il secondo dataset che abbiamo selezionato lo abbiamo ottenuto dal sito unb.ca, sito dell'università di New Brunswick, ed è stato ottenuto attraverso un software che metteva in esecuzione e controllava l'andamento di diverse app android. Questo dataset infatti, a differenza di quello di prima, e stato creato secondo un'analisi "dinamica" delle app android, ovvero su risultati ottenuti dopo aver eseguito l'applicazione.
All'interno del dataset, vengono salvate per ogni applicazione il numero di chiamate al sistema.
Il link di questo dataset è il seguente:
https://www.unb.ca/cic/datasets/maldroid-2020.html

	ACCESS_PERSONAL_INFO___	ALTER_PHONE_STATE___	ANTI_DEBUG_____	CREATE_FOLDER_____
0	1	0	0	3
1	3	0	0	6
2	2	0	0	4
3	1	0	0	4
4	3	0	0	11
...	...	...	...	...
11593	2	0	0	11
11594	6	0	0	10
11595	0	0	0	0
11596	1	0	0	15
11597	0	0	0	10

11598 rows × 471 columns

5. MANIPOLAZIONE DEI DATASET

- ## DATASET Analisi Statica:

Innanzitutto il dataset presentava 5 app nelle quali mancava l'informazione riguardo ad un determinato permesso, invece di 1 o 0 era presente un "?".
Trattandosi di soltanto 5 record su 15036, abbiamo pensato semplicemente di eliminarli dal dataset, visto che non avrebbero influito in modo particolare.

Non abbiamo usato un processo di standardizzazione siccome il formato dei valori era di tipo booleano.
Abbiamo quindi proceduto con una feature selection, tramite un algoritmo di backward elimination, il quale ci ha portato da 216 features iniziali a 138 features finali.
Processo che è risultato abbastanza lento da completare portando un miglioramento del classificatore di solo 1% e che quindi in un rapporto tempo/risultati non si dimostra essere molto conveniente.

- ## DATASET Analisi Dinamica:

Abbiamo standardizzato i valori attraverso l'algoritmo "StandardScaler", che ha trasformato i valori in modo da ottenere media uguale a 0 e varianza pari a 1.
Abbiamo deciso di eseguire questa standardizzazione poiché i risultati standardizzati sarebbero stati più comodi da usare nei classificatori.

Le app all'interno del dataset erano divise in 5 classi distinte:

1. La classe 5 identificava le applicazioni benigne.
2. Le classi da 1 a 4 rappresentavano invece 4 tipi di malware differenti.

Abbiamo quindi raggruppato le classi da 1 a 4 in un' unica classe malware (classe 1), e la classe 5 in una classe di non malware (classe 0).
Così facendo però le due classi risultavano essere molto sbilanciate: 9803 malware, contro 1795 non malware.
Per fare fronte a questo problema abbiamo provveduto a bilanciare le classi usando l'algoritmo chiamato "SMOTE", che si occupa di creare nuovi record della classe in minoranza basandosi però sui dati già presenti e noti.

6. CLASSIFICATORI E ALGORITMI

Avendo a disposizione molti dati abbiamo proceduto con una addestramento supervisionato, utilizzando i ⅔ dei dati come valori di apprendimento e il restante ⅓ di valori come test.

Per entrambi i dataset abbiamo utilizzato i seguenti classificatori:

1. Alberi decisionale : classification tree
2. Support Vector Machines: SVC Lineare
3. Support Vector Machines: SVC di tipo “ovo”
4. Random Forest

Questi sono i risultati dei classificatori:

- ANALISI Statica

DecisionTreeClassifier() 0.971779883088087 0.23 seconds

	precision	recall	f1-score	support
0	0.98	0.98	0.98	3126
1	0.96	0.96	0.96	1835
accuracy			0.97	4961
macro avg	0.97	0.97	0.97	4961
weighted avg	0.97	0.97	0.97	4961

LinearSVC() 0.9776254787341262 0.29 seconds

	precision	recall	f1-score	support
0	0.98	0.99	0.98	3126
1	0.97	0.96	0.97	1835
accuracy			0.98	4961
macro avg	0.98	0.97	0.98	4961
weighted avg	0.98	0.98	0.98	4961

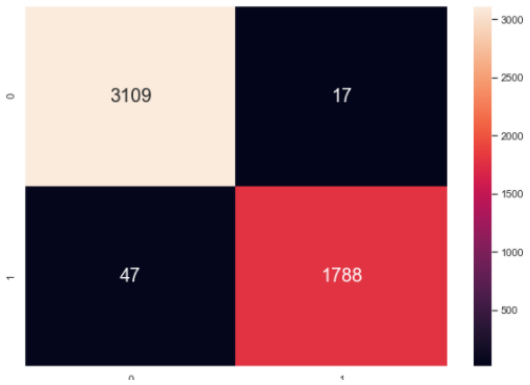
SVC(decision_function_shape='ovo') 0.9818584962709132 4.76 seconds

	precision	recall	f1-score	support
0	0.98	0.99	0.99	3126
1	0.99	0.96	0.98	1835
accuracy			0.98	4961
macro avg	0.98	0.98	0.98	4961
weighted avg	0.98	0.98	0.98	4961

Per l’algoritmo della RandomForest invece i risultati e la matrice di confusione sono i seguenti:

RandomForestClassifier()

	precision	recall	f1-score	support
0	0.99	0.99	0.99	3126
1	0.99	0.97	0.98	1835
accuracy			0.99	4961
macro avg	0.99	0.98	0.99	4961
weighted avg	0.99	0.99	0.99	4961



- ANALISI Dinamica

DecisionTreeClassifier() 0.9741885625965997 2.23 seconds

	precision	recall	f1-score	support
0.0	0.97	0.97	0.97	3240
1.0	0.97	0.97	0.97	3230
accuracy			0.97	6470
macro avg	0.97	0.97	0.97	6470
weighted avg	0.97	0.97	0.97	6470

LinearSVC() 0.9431221020092736 7.89 seconds

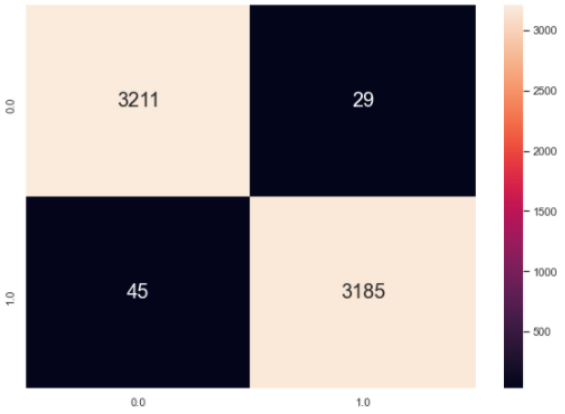
	precision	recall	f1-score	support
0.0	0.96	0.92	0.94	3240
1.0	0.92	0.96	0.94	3230
accuracy			0.94	6470
macro avg	0.94	0.94	0.94	6470
weighted avg	0.94	0.94	0.94	6470

SVC(decision_function_shape='ovo') 0.9088098918083463 55.86 seconds

	precision	recall	f1-score	support
0.0	0.96	0.86	0.90	3240
1.0	0.87	0.96	0.91	3230
accuracy			0.91	6470
macro avg	0.91	0.91	0.91	6470
weighted avg	0.91	0.91	0.91	6470

Per l'algoritmo della RandomForest invece i risultati e la matrice di confusione sono i seguenti:

	precision	recall	f1-score	support
0.0	0.99	0.99	0.99	3240
1.0	0.99	0.99	0.99	3230
accuracy			0.99	6470
macro avg	0.99	0.99	0.99	6470
weighted avg	0.99	0.99	0.99	6470



7. RISULTATI

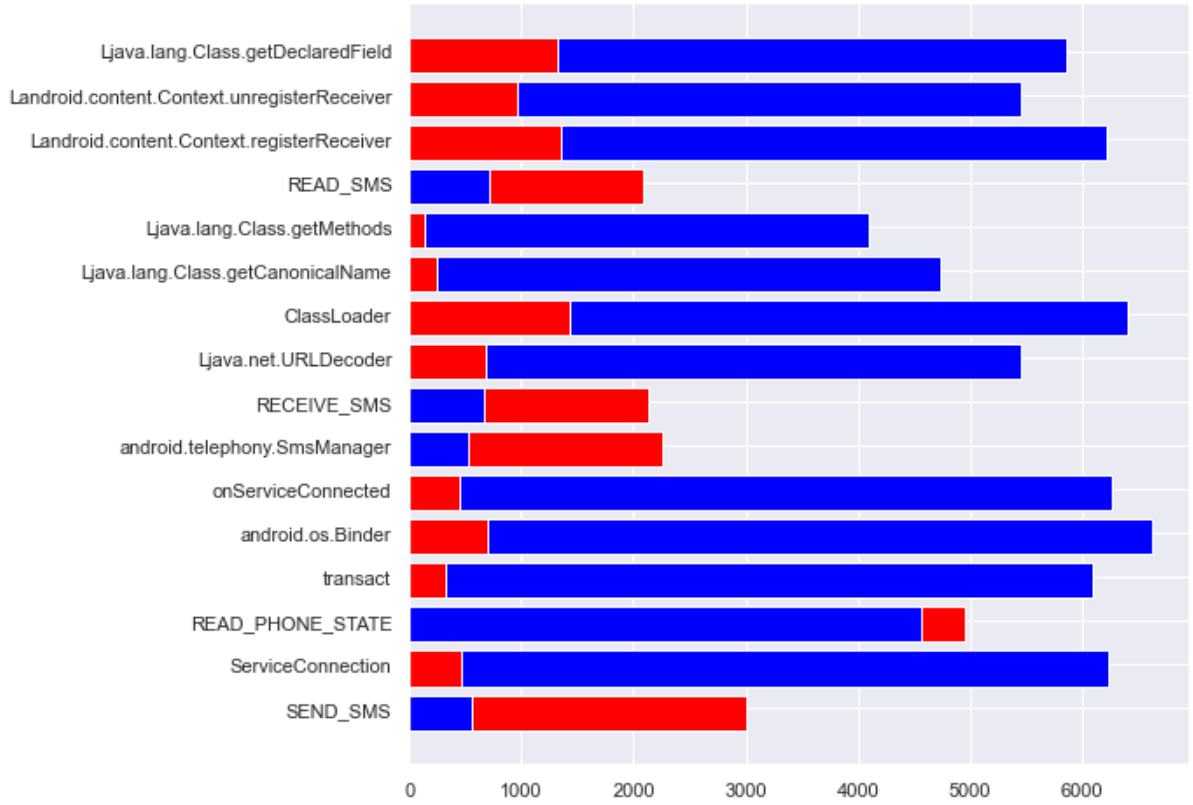
I punteggi finali dei classificatori sono tutti molto buoni, con accuratezze alte.

Per l'analisi statica prendendo in considerazione l'algoritmo della Random Forest abbiamo ottenuto che le 16 feature più significative, con le rispettive percentuali sono:

- ('READ_PHONE_STATE', 0.061347730725012185)
- ('SEND_SMS', 0.061060388820738914)
- ('transact', 0.05974109290115132)
- ('android.telephony.SmsManager', 0.05128039042098174)
- ('ServiceConnection', 0.04718310473939612)
- ('onServiceConnected', 0.04497747832712361)
- ('android.os.Binder', 0.039824765811694855)
- ('RECEIVE_SMS', 0.032835314246563004)
- ('Ljava.lang.Class.getCanonicalName', 0.029607366373408585)
- ('Landroid.content.Context.registerReceiver', 0.020071245746877475)
- ('Ljava.net.URLDecoder', 0.01943457627835149)

- ('READ_SMS', 0.019287613380217245)
- ('ClassLoader', 0.018636189070683554)
- ('Ljava.lang.Class.getMethods', 0.015391070825985897)
- ('Landroid.content.Context.unregisterReceiver', 0.012843899403123953)
- ('Ljava.lang.Class.getDeclaredField', 0.008748934053531488)

Nel grafico qua sotto vengono mostrate le feature a confronto tra malware (rosso) e non malware(blu). Per ogni feature sono state contate il numero di app che richiedono quel permesso, divise appunto tra malware e non.



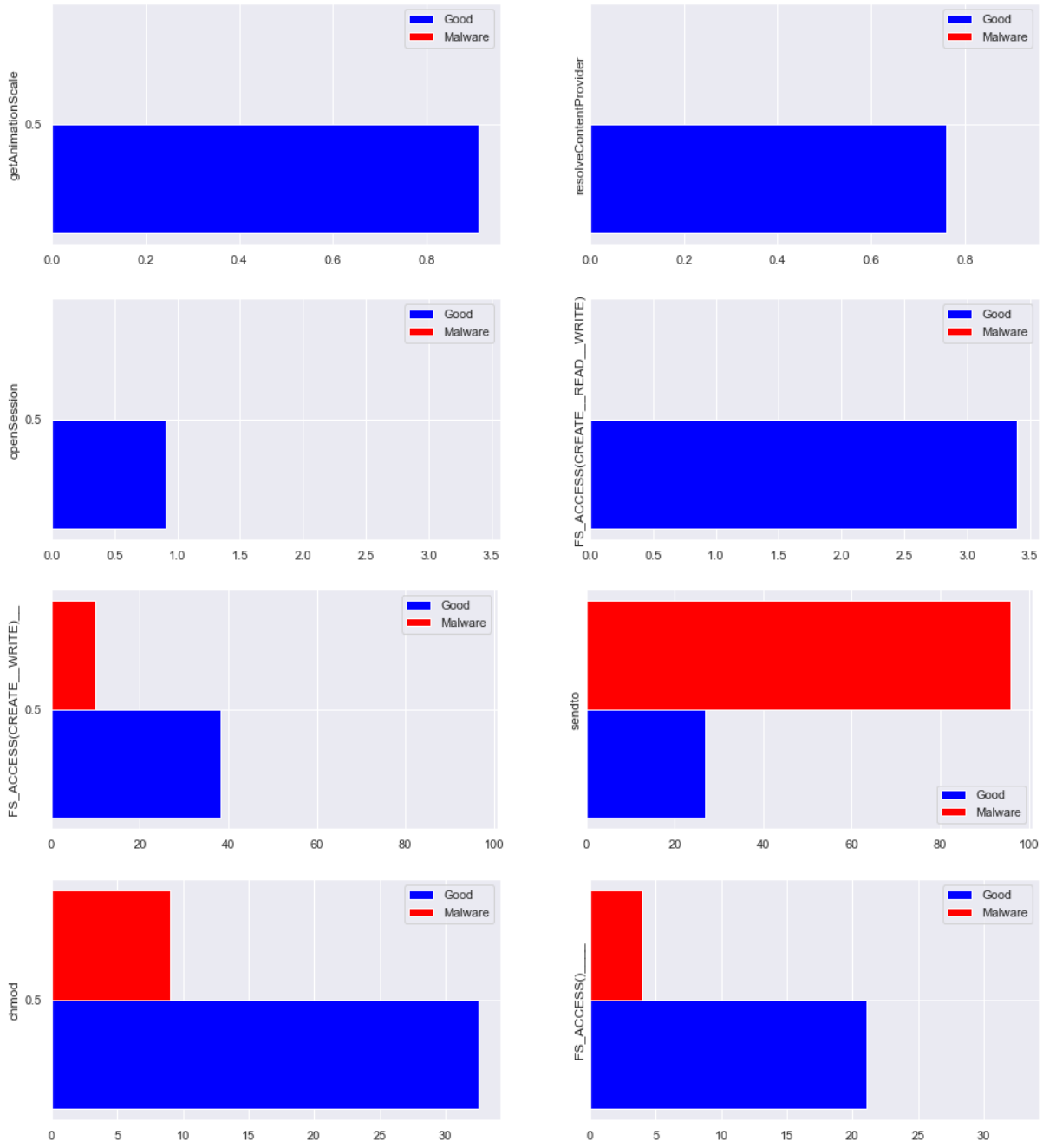
Come si può vedere dal grafico ci sono più feature caratterizzanti per le app benigne. Questo sta a significare che probabilmente l'algoritmo si basa di più su un uso consueto di app, rispetto ad uno anomalo. Cioè che se un malware non richiede un permesso abbastanza basilare significa che avrà un uso inconsueto e potrebbe essere un malware. Mentre le feature in cui prevalgono i malware sulle benigne sono : SEND_SMS, READ_PHONE_STATE, RECEIVE_SMS, READ_SMS e android.telephony.SmsManager. Che come si può intuire dal nome, sono tutti permessi che si occupano di gestire gli SMS del telefono, principio sul quale al giorno d'oggi si basano molti sistemi di recupero e password e autenticazione a due fattori.

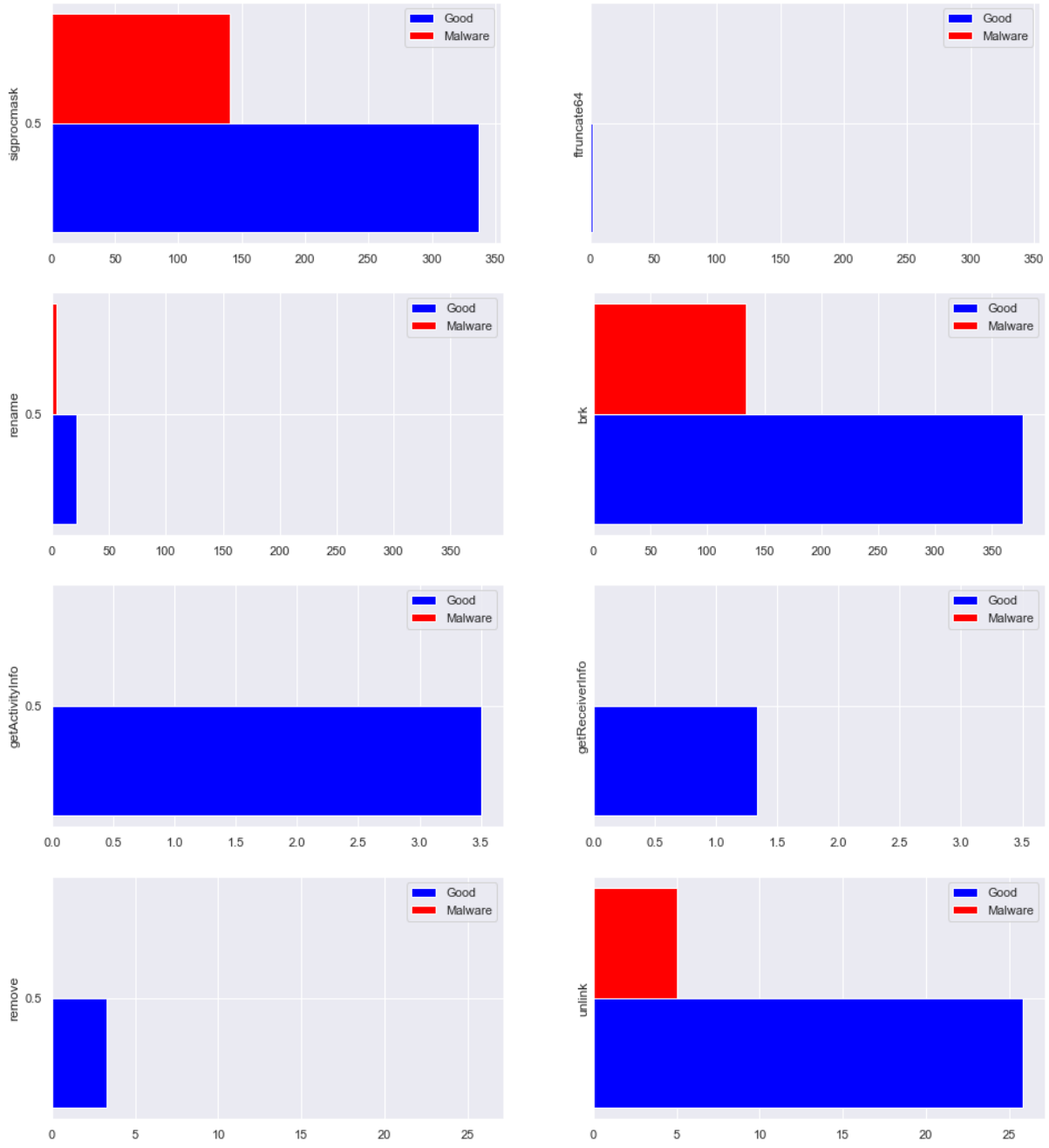
Per l'analisi dinamica invece prendendo in considerazione l'algoritmo della Random Forest abbiamo ottenuto che le 16 feature più significative, con le rispettive percentuali sono:

- ('remove', 0.013086390097639218)
- ('getActivityInfo', 0.01220684992966622)
- ('getReceiverInfo', 0.011163230672380902)
- ('FS_ACCESS(CREATE__WRITE)__', 0.007793865573969517)
- ('brk', 0.007218753550627883)
- ('unlink', 0.0066045254249748764)
- ('FS_ACCESS(CREATE__READ__WRITE)', 0.005698358417476972)
- ('FS_ACCESS()____', 0.00504405795211472)

- ('sendto', 0.0042206283362561485)
- ('sigprocmask', 0.004066025858129672)
- ('chmod', 0.0038334155829664405)
- ('ftruncate64', 0.002916479886587882)
- ('rename', 0.002344590672433962)
- ('openSession', 0.0008107942595521399)
- ('getAnimationScale', 0.0006355255766087705)
- ('resolveContentProvider', 0.0005096282369284176)

Nei grafici qua sotto vengono messi a confronto malware e non malware sulle 16 feature più significative. Per ogni feature viene confrontata la media del numero di “chiamate” a quella determinata funzione.





Anche qua si può notare che prevalgono i non malware sui malware. Quindi anche qua probabilmente il classificatore identifica un malware non tanto basandosi su feature caratteristiche dei malware ma più su un funzionamento non consueto di un'app. Anche perché la maggior parte delle feature più significative sono funzioni abbastanza semplici che vengono usate spesso all'interno di un'applicazione per le richieste più basiche. L'unica feature in cui prevalgono i malware è la feature "sendto", che una azione che serve per inviare mail. Le mail sono alla base di tutte le autenticazioni sui siti web ed è chiaro che non dovrebbero essere gestite da eventuali malware.

8. CONCLUSIONI E IDEE FUTURE

I risultati dei classificatori sono buoni e avendo effettuato 2 tipi diversi di analisi possiamo ottenere informazioni su app in 2 momenti diversi del suo ciclo di vita su uno smartphone: prima che venga eseguita e a seguito dell'esecuzione. In particolare dalla nostra analisi è risultato come principali metodi per l'individuazione dei malware:

1. Funzionamento anomalo e inconsueto da parte di un'applicazione, distante cioè dai normali permessi e utilizzi che richiedono le applicazioni
2. Richiesta di permessi che hanno a che fare con gli SMS
3. Utilizzo di funzioni che consentono l'invio e la gestione di Mail

Per il futuro potremmo mettere in relazione le due analisi tra loro, magari confrontando i permessi con le effettive chiamate/funzioni di sistema.