Versione 1

21/06/2024

Relazione di Progetto:

ARCHITETTURA DATI

GoodWare VS Malware:

Conseguenze dello sporcare un DataSet sui modelli predittivi

Presented by: cavallini Francesco

Matricola: 920835

UNIVERSITÀ MILANO BICOCCA

# 0. Sommario:

[0. Sommario: 2](#_Toc1)

[1. Introduzione: goodware vs malaware 3](#_Toc2)

[1.1. Premessa: Scelta del tema: 3](#_Toc3)

[1.2. Il Dataset: 3](#_Toc4)

[1.3. Obbiettivi del progetto: 4](#_Toc5)

[1.4. Fasi del Progetto: 5](#_Toc6)

# 1. Introduzione: goodware vs malaware

## 1.1. Premessa: Scelta del tema:

Nel panorama attuale, la minaccia informatica è in costante aumento e diversificazione. I malware, software dannosi progettati per danneggiare o rubare informazioni dai sistemi informatici, rappresentano una seria preoccupazione per individui e organizzazioni. La capacità di distinguere tra goodware, software legittimo e sicuro, e malware è fondamentale per la sicurezza informatica. Si decide quidni di proporre un progetto che rispecchia sia tematiche attuali che, banalmente, di puro interesse personale.

## 1.2. Il Dataset:

Per sviluppare il progetto si vuole quindi leggere un dataset di applicazioni Android. Si parte dunque da un dataset da 4465 istanze con le seguenti 241 feature:

* Colonna 1 214: Permission-based features   
  sono tutte feature binarie, dove
  + 0 = permesso non richiesto,
  + 1 = permesso richiesto
* Colonna 215 241: API based features  
  sono tutte feature binarie:
  + 0 = api call non richiesta
  + 1 = api call richiesta
* Colonna 242: label (target)  
  dove le classi sono:
  + Malware
  + Goodware

## 1.3. Obbiettivi del progetto:

Avendo quindi chiari quali sono le premesse e su quali dati andremo a lavorare possiamo ora iniziare a parlare dell’implementazione del progetto. Gli obbiettivi ultimi che si voglio raggiungere con ‘Goodware VS Malware’ sono:

1. **La base:**

Dimostrare che è possibile predire correttamente se un applicazione è un malware o goodware in base ai permessi e le api call fornite dal dataset

1. **Feature selection:**

Partendo dal dataset completo di tutte le feature andare a capire quali sono le feature più importanti per la classificazione ed isolarle.

1. **Sporcare le feature:**

Verificare come cambiano le performance dei modelli di classificazione che andremo ad utilizzare sporcando i dati all'interno del dataset

1. **Riproducibilità dei dati:**

Si presta particolare attenzione sul rendere più alta possibile la riproducibilità dei dati, ma avendo che molte funzioni per l’inserimento di rumore all’interno del dataset utilizzano operazioni che dipendono dalle funzioni di random abbiamo un problema: anche impostando un seed se il kernel del notebook viene riavviato (o banalmente si effettuano lavorazioni su una macchina con un kernel diverso) i risultati cambieranno leggermente. In quanto i seed sono comunque dipendendi dal kernel del notebook.

## 1.4. Fasi del Progetto:

Per completare gli obbiettivi sono quindi stati realizzati i seguenti step:

1. **Caricare il dataset**
2. **Data exploration:**

Un'analisi esplorativa del dataset per comprendere la distribuzione delle feature, la presenza di valori mancanti, la correlazione tra le feature e altre informazioni rilevanti.

1. **Scelta dei modelli: Baseline e Modelli di validazione:**

Selezione di 2 modelli di machine learning adatti alla risoluzione del nostri obbiettivi. Una volta scelti questi modelli si potranno poi definire delle funzioni per trainare i modelli. Queste funzioni saranno dunque utili:

* 1. Sia per creare una baseline, ossia una sorta di modello “benchmark” che ci da informazioni di classificazione prima che il dataset venga sporcato.
  2. Sia per la feature selection, ossia il passaggio successivo, per verificare che la feature selection stia andando bene
  3. Trainare nuovi modelli, passando però come train set il dataset sporco.

1. **Preparazione dati: Feature selection:**

Identificare quali possono essere le feature più importanti del dataset, ossia ridurre il numero totale di feature da 256 ad un gruppo più ristretto fer facilitare le lavorazioni seguenti. Quindi in questa fase andremo poi anche a creare una nuova baseline in più rispetto alla precedente, ossia la baseline dei modelli addestrati su un pool ristretto di feature.

1. **Data exploration PT2: Feature che meglio discriminano:**

Su tutti i modelli di baseline che abbiamo creato al punto precedente andiamo poi a stabilire un elenco di tutte le feature che sono più discriminanti per i modelli selezionati.

1. **Sporcare il dataset:**

Il cuore del progetto: valutare le prestazioni di nuovi modelli trainati utilizzando lo stesso dataset ma che viene sporcato iterativamente. Misura metriche come precision, recall, F1-score, accuracy e qualsiasi altra metrica rilevante. In questa fase abbiamo inoltre una moltitudine di sotto-fasi:

* 1. **Confronto con il baseline**: Confrontare le prestazioni dei modelli trainati con il dataset sporchi rispetto al baseline. Analizzare le differenze nelle prestazioni e identificare le feature che hanno subito il maggior impatto.
  2. **Interpretazione dei risultati**: Spiegare il comportamento dei modelli in risposta al dataset sporco. Verranno identificate nuovamente le feature più discriminanti e si discuterà come il l’inserimento di rumore in esse ha influenzato le prestazioni del modello.
  3. **Riflessioni e conclusioni**: Conclusioni del progetto con riflessioni sull'efficacia delle tecniche di inserimento di rumore e nel rivelare l'importanza delle feature e sull'interpretazione dei risultati ottenuti.

# 2. Caricare il dataset + Data Exploration

## 2.1. Caricamento del dataset:

I passaggi di caricamento del dataset vengono effettuati regolarmente, tramite l’utilizzo della libreria pandas, in quanto il nostro dataset non è altro che un file CSV.

2.2. Data exploation:

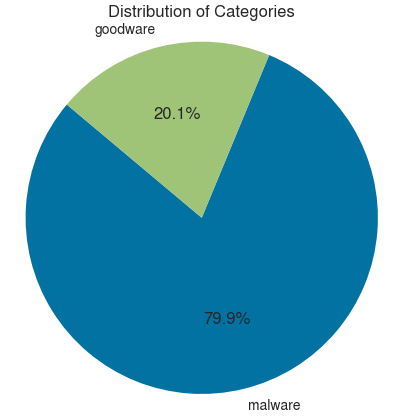
In questa prima fase di esplorazione del dataset andremo a scoprire quali sono le qualità intrinseche del dataset per formulare un idea di quali modelli utilizzare e cosa aspettarci dalla utilizzo di sopracitati modelli. Ovviamente dovremmo fare utilizzo di più passaggi per verificare più qualità del dataset, questi passaggi sono:

* Verifica dataset sbilanciato
* Verifica presenza valori nulli
* Verifica sparsità dei dati
* Verifica correlazione con il target

Questi passaggi vengono approfonditi nelle sezioni sottostanti:

### 2.2.1. Verifica dataset sbilanciato:

Ci chiediamo dunque se Il dataset è sbilanciato. Per verificare facciamo un analisi univariata sui valori di Label, questi sono i risultati:



Il dataset si presenta sbilanciato (anche se non fortemente sbilanciato). Per quanto possa non piacerci molto la notizia che il dataset sia sbilanciato (può comportare degli errori di bias nella classificazione dei modelli) abbiamo però la buona notizia della sicurezza che la label sia binaria.

Sarà quindi necessario utilizzare modelli che offrono buone prestazioni su dataset binari e non troppo suscettibili a dataset sbilanciati. Inoltre poiché questo set di dati è sbilanciato, consideriamo di non utilizzare l'accuratezza come parametro di valutazione, sarebbe meglio invece usare l'F1-score come soglia primaria per determinare le performance dei nostri modelli in quanto meno suscettibile.

### 2.2.2. Verifica presenza valori nulli:

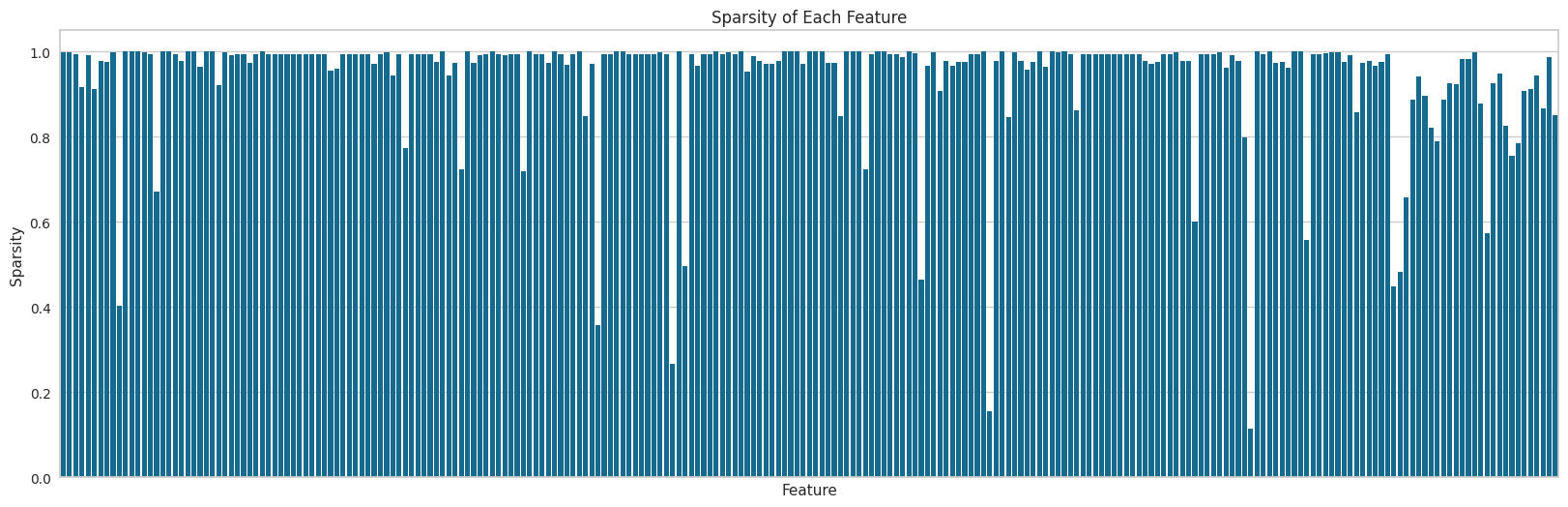
Controlliamo ora se il dataset contiene valori mancanti. Dopo un primo controllo sembra che tutte le features contengano almeno un valore nullo, Ma in seguito ad un analisi più approfondita (tramite il richiamo di una funzione che ci andrà ad elencare tutte le instanze all'interno di ogni features con valori mancanti) ci accorgiamo che è solo la riga 2533 ad avere valori nulli.

Decidiamo dunque eliminare la riga dal dataset, in quanto

1. Non fornisce nessun valore infromativo, tutte le sue feature sono nulle)
2. Stiamo eliminando una sola riga, anche se avesse avuto delle feature non nulle la perdita di informazione sarebbe stata pressapoco irrisoria

2.2.3. Verifica sparsità dei dati:

Guardiamo la distribuzione dei valori delle features. In particolare ci interessa vedere se molte di queste feature hanno valori di perlopiù zero o perlopiù uno, questo è ciò che otteniamo:



Considerando che:

* **Alta sparsità** (barra vicino a 1): La feature è dominata da zeri. Questo può indicare che l'evento rappresentato da un 1 è raro.
* **Bassa sparsità** (barra vicino a 0): La feature è dominata da 1. Questo può indicare che l'evento rappresentato da un 1 è comune.
* **Sparsità media** (barra intorno a 0.5): La feature ha un numero bilanciato di 0 e 1.

Allora è facile notare che la maggior parte delle feature non è bilanciata, la stra-grande maggioranza contiene quasi unicamente zeri. (Pensando al nostro contesto questo valore ha perfettamente senso, la maggior parte delle applicazioni android usa solo una piccola parte di tutti i permessi disponibili che si possono inserire al interno del manifest, quindi avremo che per ogni istanza la maggior parte di colonne sarà valorizzata a zero). Questo comunque non rappresenta un grosso problema al di fuori del fatto che avremo probabilmente moltissime colonne che detengono lo stesso valore informativo.

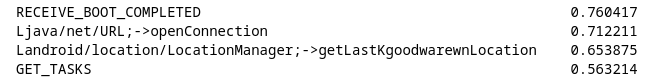
### 2.2.4. Verifica correlazione con il target:

Esploriamo ora alla ricerca di colonne con un altro coefficiente di Pearson.

Considerando che il coefficiente di correlazione di Pearson ha valori compresi tra -1 e 1, abbiamo che:

* Un valore più vicino allo 0 implica una correlazione più debole (0 esatto implica nessuna correlazione)
* Un valore più vicino a 1 implica una correlazione positiva più forte
* Un valore più vicino a -1 implica una correlazione negativa più forte

Definiamo quindi, una funzione per trovare le colonne che hanno correlazione con il target superiore a una certa percentuale parametro (se ce ne sono), di seguito se ne mostrano i risultati di ricerca di colonne con correlazione con il target superiore al 50%:



Dal test appena eseguito si dimostra che, su tutte le 241 feature, esistono quindi delle colonne, per la precisione 4, che hanno correlazione superiore al 50% con il target. Questo è un dato estremamente favorevole, infatti ciò ci mostra che esistono veramente poche colonne che hanno un alta correlazione con il target. Quindi, quando andremo ad inserire rumore all’interno del dataset, sarà necessario prestare particolare attenzione a queste colonne.

3. Scelta dei modelli: Baseline e Modelli di validazione:

3.1. Scelta modelli ed implementazione di funzioni di train:

Paradossalmente, per controllare se le operazioni di preparazione del dataset che faremo allo step successivo siano corrette, dobbiamo testare le operazioni di selezione delle feature con dei modelli giocattolo (per valutare le performance della feature selection che vengono fatte passo passo); in modo da avere conferma che non stiamo peggiorando le performance della classificazione eliminando troppe fatures, o peggio eliminando features importanti per la classificazione.

È dunque fondamentale che in questa fase si proceda con la scelta dei modelli, basandoci sulle informazioni che abbiamo ottenuto dall’esplorazione dati al passaggio precedente.

Si sviluppano dunque due funzioni (una per ogni modello) che, forniti come parametro i dati di train e i dati di test, saranno utili al training del rispettivo modello. In questo modo garantiamo anche per tutte le operazioni future tutti i modelli dello stesso tipo vengano trainati allo stesso modo e le uniche differenze nel train saranno i dati di train e test forniti.

Per quanto riguarda invece la scelta dei modelli usati per la classificazione dei dati ricade sui 2 seguenti modelli:

* **Logistic Regression**:

Un modello molto ben prestante alla presenza di un dataset sbilanciato, sparso e con molte feature (che quindi ci aspettiamo funzioni molto bene fin da subito). Si sceglie dunque questo modello per la sua robustezza anche in presenza di rumore, inoltre, come qualità aggiuntiva, abbiamo che Logistic Regression è un modello lineare che fornisce coefficienti (Theta) che possono essere interpretati facilmente, permettendo di capire l'importanza relativa di ciascuna feature.

* **Bernnulli Naive bayes**:

Generalmente i modelli di Naive Byes sono modelli molto soggetti a bias, ma, in particolare, il modello di Bernulli Naive Bayes è particolarmente robusto quando molte feature assumono valori zero (come descritto nel dataset). Il motivo principale per la scelta di questo modello, però, ricorre nel fatto che è un modello molto semplice e computazionalmente efficiente, adatto a dataset con molte feature. Data la sua semplicità ci aspettiamo performi leggermente peggio rispetto al modello precedente, anche se non di troppo avendo comunque un dataset con caratteristiche sorprendentemente ben allineate con le richieste di bernoulli naive bayes (Rispettiamo infatti anche l'assunzione di indipendenza tra le feature, in quanto all’interno del nostro dominio applicativo generalmente l’utilizzo di una api call / permesso non è mai, o quasi mai, indipendente dall’utilizzo di un altra api call / permesso). Ci aspettiamo dunque che performi peggio rispetto al modello precedente nel momento in cui introduciamo rumore nel dataset

In sostanza la scelta di questi due modelli è data dal fatto che il primo si presta molto bene anche in presenza di rumore nel dataset, il secondo invece ne è molto più suscettibile; di coneguenza sarà interessante andare ad esaminare nelle fasi successive la differenza nei comportamenti tra i 2 modelli.

Nota: siccome stiamo usando un data-set sbilanciano e sparso è bene se ci basiamo sulla metriche di:

* F1-score
* MCC-score

invece che sull'accuracy per avere una valutazione più realistica delle performance dei modelli.

Sia per creare una baseline, ossia una sorta di modello “benchmark” che ci da informazioni di classificazione prima che il dataset venga sporcato.