

VERSIONE 1.3

25/06/2024

Relazione di Progetto:

# ARCHITETTURA DATI

GoodWare VS Malware:

Conseguenze dello sporcare un DataSet sui modelli di classificazione

PRESENTED BY: CAVALLINI FRANCESCO

MATRICOLA: 920835

UNIVERSITÀ MILANO BICOCCA



## 0. SOMMARIO:

0. Sommario:.....	3
1. Introduzione: goodware vs malware.....	6
1.1. Premessa: Scelta del tema:.....	6
1.2. Il Dataset:.....	6
1.3. Obiettivi del progetto:.....	7
1.4. Fasi del Progetto:.....	8
2. Caricare il dataset + Data Exploration.....	10
2.1. Caricamento del dataset:.....	10
2.2. Data exploration:.....	10
2.2.1. Verifica dataset sbilanciato: .....	10
2.2.2. Verifica presenza valori nulli:.....	11
2.2.3. Verifica sparsità dei dati: .....	11
2.2.4. Verifica correlazione con il target:.....	12
3. Scelta dei modelli: Baseline e Modelli di validazione: .....	14
3.1. Scelta modelli ed implementazione di funzioni di train: .....	14
3.2. Modelli di baseline.....	16
4. Preparazione dati: Feature selection.....	20
4.1 Analisi bivariata della correlazione - Eliminazione feature riNdondanti:.....	20
4.2. Analisi bivariata di correlazione - Highlight feature più correlate al target.....	24
4.3. RFE-CV (Recursive feature elimination):.....	26
5. Data exploration PT2: Feature che meglio discriminano:.....	30
5.1. Visualizzazione dei risultati: .....	30
5.2. Osservazioni generali.....	33
5.3. Note implementative e teoriche:.....	33
5.2.1. Per Logistic Regression: .....	33
5.2.2. Per Bernoulli Naive Bayes:.....	33
6. Sporcare i dataset:.....	35
6.0. Introduzione: .....	35
6.1. Sostituzione di valori con valori casuali: .....	37
6.1.1. Visualizzazione differenze introdotte con inserimento valori casuali: .....	37
6.1.2. Visualizzazione risultati prodotti dai modelli .....	38
6.1.3. Considerazioni e conclusioni sul decadimento delle performance tra baseline e modelli con dati sporchi:.....	39
6.2 Sostituzione di valori con valori nulli:.....	43
6.2.1. Visualizzazione differenze introdotte con inserimento valori casuali: .....	44
6.2.2. Visualizzazione risultati prodotti dai modelli .....	45
6.2.3. Considerazioni e conclusioni sul decadimento delle performance tra baseline e modelli con	

dati sporchi:.....	46
6.2.3.1. Significatività nel dominio applicativo:.....	47
6.2.3.2. Considerazioni performance temporali:.....	48
6.3. Sostituzione di valori con valori opposti.....	49
6.3.1. Visualizzazione risultati prodotti dai modelli:.....	50
6.3.2. Considerazioni e conclusioni sul decadimento delle performance tra baseline e modelli con dati sporchi:.....	53
6.3.2.2. Naive Bayes: Considerazioni sul cambiamento delle performance:.....	57
6.3.2.3. Considerazioni sulla significatività nel Dominio applicativo:.....	58
7. Recap conclusivo.....	60
7.1. Recap sugli obiettivi di classificazione: .....	60
7.2. Recap sulla significatività delle predizioni sul dominio:.....	61



# 1. INTRODUZIONE: GOODWARE VS MALWARE

## 1.1. PREMESSA: SCELTA DEL TEMA:

Nel panorama attuale, le minacce informatiche sono 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 quindi 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 (che rietichetteremo come 0)
  - Goodware (che rietichetteremo come 1)

### **1.3. OBIETTIVI 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 obiettivi ultimi per la classificazione che si vogliamo 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.

**2. Feature selection:**

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

**3. Sporcare le feature:**

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

**4. 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 dipendenti dal kernel del notebook.

## 1.4. FASI DEL PROGETTO:

Per completare gli obiettivi 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.

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

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

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

**4. 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 per 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.

**5. 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.

**6. Sporcare il dataset:**

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

- a. **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.
- b. **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.
- c. **Riflessioni e conclusioni:** Conclusioni relative a ciascun istanza di inserimento di dati sporchi, 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 EXPLOSION:

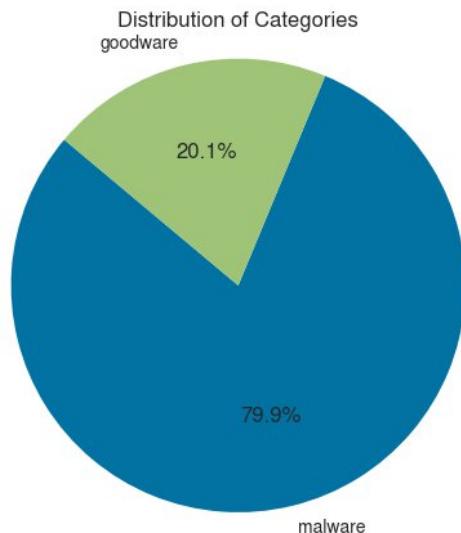
In questa prima fase di esplorazione del dataset andremo a scoprire quali sono le qualità intrinseche del dataset per formulare un'idea di quali modelli utilizzare e cosa aspettarci dall'utilizzo di sopracitati modelli. Ovviamente dovremmo fare utilizzo di più passaggi per verificare più qualità del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

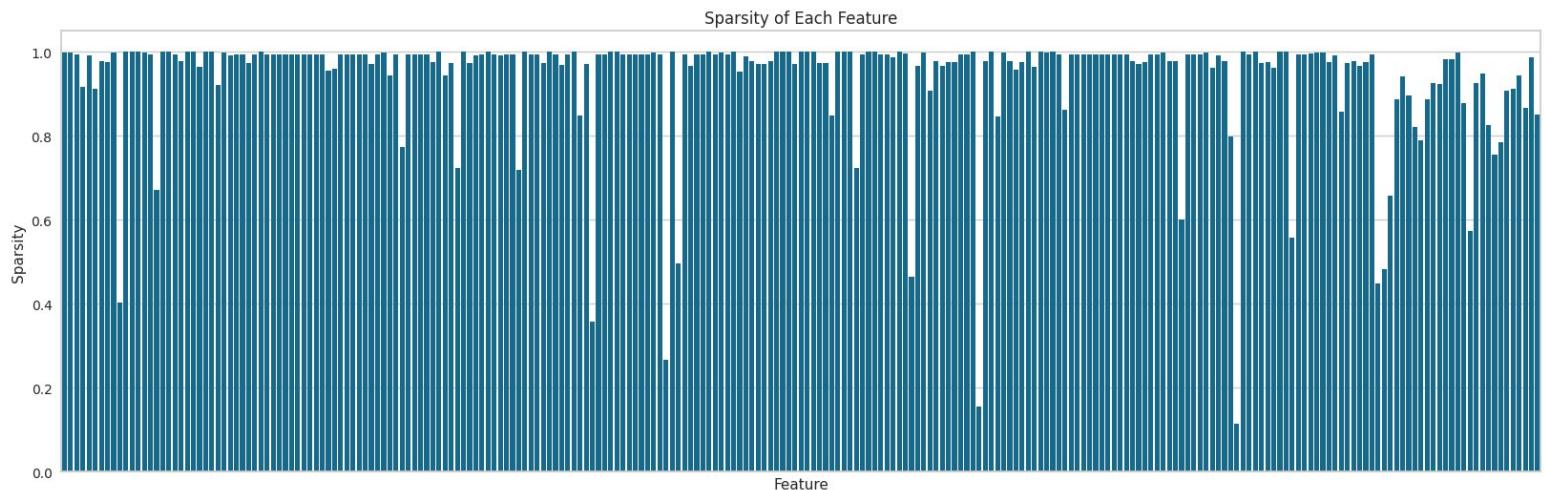
Controlliamo ora se il dataset contiene valori mancanti. Dopo un primo controllo sembra che tutte le features contengano almeno un valore nullo. Ma in seguito ad un analisi più approfondita (tramite il richiamo di una funzione che ci andrà ad elencare tutte le istanze all'interno di ogni features con 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 informativo, 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 irrilevante.

### 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 valore assoluto di 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%:

RECEIVE_BOOT_COMPLETED	0.760417
Ljava/net/URL;:->openConnection	0.712211
Landroid/location/LocationManager;:->getLastKnownLocation	0.653875
GET_TASKS	0.563214

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 features, 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 che 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.

- **Bernoulli Naive bayes:**

Generalmente i modelli di Naive Bayes sono modelli molto soggetti a bias, ma, in particolare, il modello di Bernoulli Naive Bayes è particolarmente robusto quando molte feature assumono valori zero (come descritto nel nostro 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, dipendente dall'utilizzo di un'altra api call / permesso). Ci aspettiamo però 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 conseguenza sarà interessante andare ad esaminare nelle fasi successive la differenza nei comportamenti tra i 2 modelli.

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

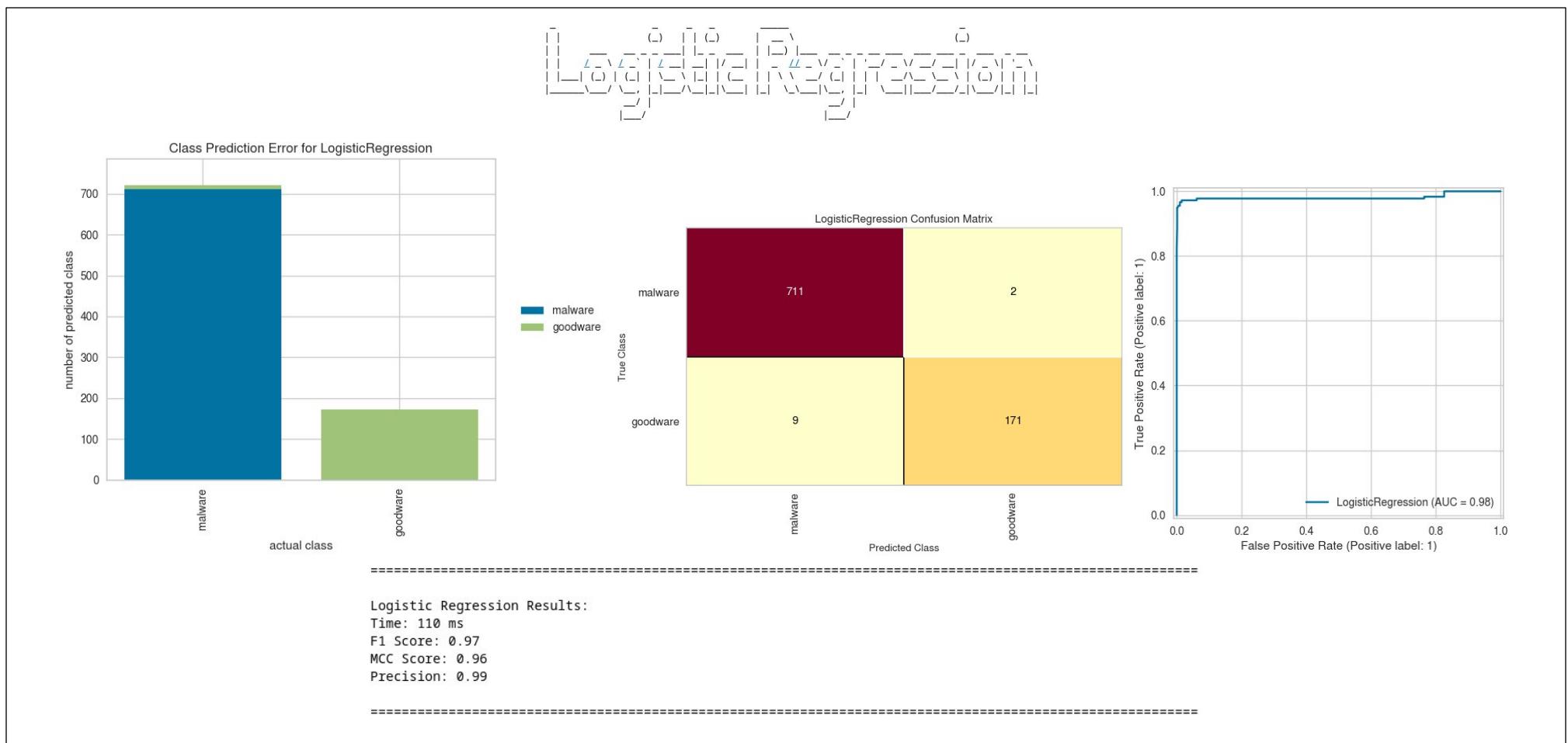
- F1-score
- MCC-score

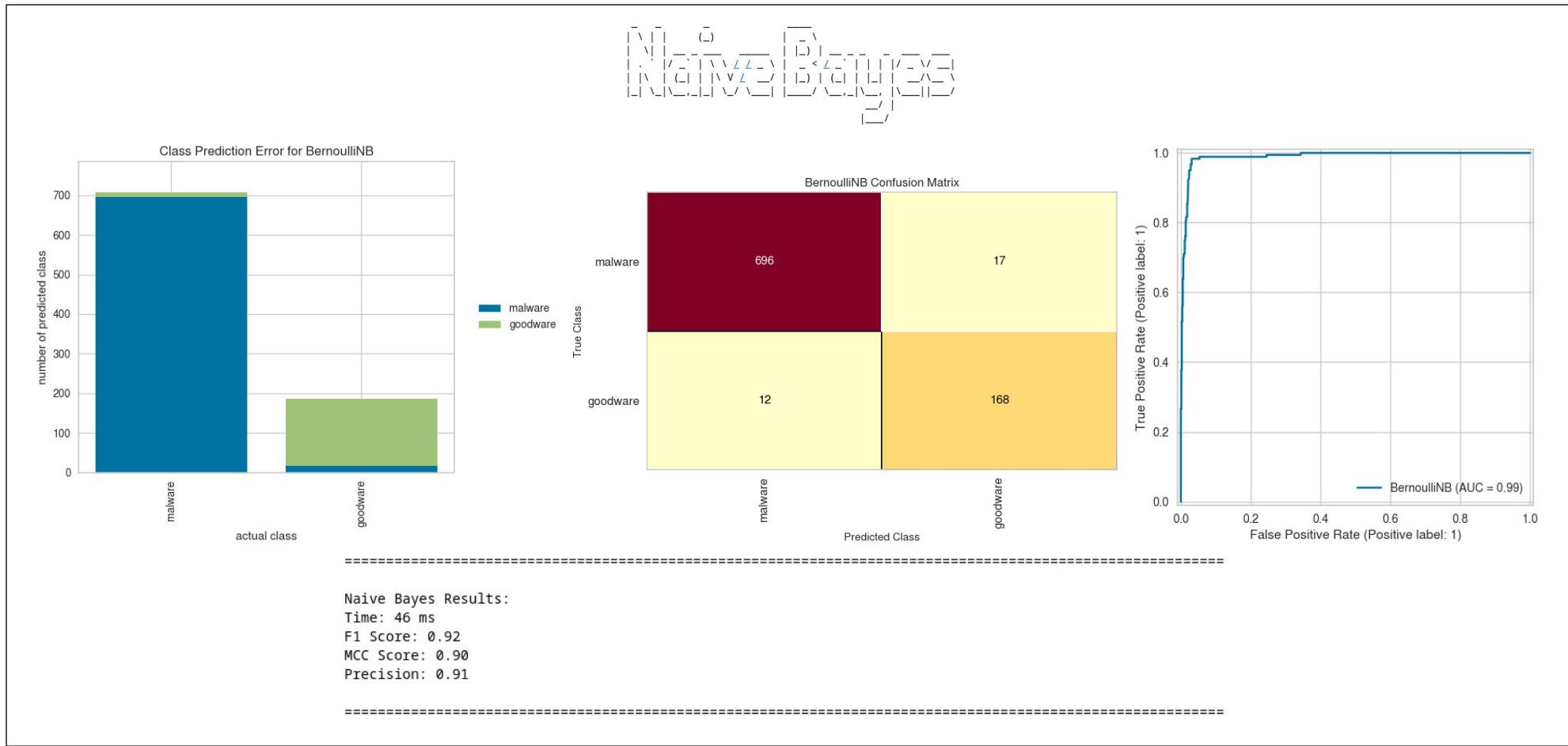
invece che sull'accuracy per avere una valutazione più realistica delle performance dei modelli.

### 3.2. MODELLI DI BASELINE

Una volta scelti i modelli e aver definito delle funzioni per trainarli, prima di passare al passo successivo, di fare feature extraction, possiamo ora creare un baseline di entrambi i modelli, facendo train e test su tutto il dataset. I risultati offerti da questi modelli verranno usati, appunto, come "benchmark" per andare a vedere come cambieranno gli score di classificazione quando, nella sezione dedicata, andremo a sporcare il dataset intero.

Di seguito vengono quindi mostrati i risultati di performance classificativa ottenuti dai modelli di baseline:





Dai plot appena mostrati riusciamo a concludere che nonostante ci siamo molte features, riusciamo comunque a valutare tutto il dataset in tempi accettabili. Questo ci piace (anche se siamo a conoscenza del fatto che dovessimo avere più istanze nel nostro dataset il tempo di predizione non scalerebbe per niente bene). Siccome noi però non dobbiamo fare altro che creare un modello giocattolo per vedere se eliminando alcune feature dal dataset riusciamo a mantenere lo f1-score questa cosa ci va bene.

Inoltre, come descritto nella sezione precedente, non spicca [da subito, con il dataset pulito] moltissimo la differenza di performance di classificazione dei due algoritmi: laddove la logistic regression fin da subito riesce ad ottenere degli ottimi risultati di classificazione (che rasentano il classificatore perfetto) il modello di naive bayes riesce ad ottenere risultati molto simili ed in tempi molto minori, misurando performance di circa 5-6% inferiori.

**Nota:** L'importanza di una scelta accurata di modelli – vecchie versioni di progetto

In versioni precedenti del progetto per sviluppare il modello di Naive Bayes era stato usato il modello Gaussian Naive Bayes, che sarebbe più adatto per dataset con feature continue. Differentemente dalle performance che abbiamo appena mostrato, con il modello di Gaussian Naive Bayes si misuravano performance che, essendo un modello inadatto, rasentavano un classificatore casuale. Questo va a dimostrare quanto sia importante aver effettuato in questa fase una scelta accurata dei modelli di classificazione.



## 4. PREPARAZIONE DATI: FEATURE SELECTION

In questa fase andremo a dimostrare come pulire i dati, in particolar modo fare feature extraction per far rimanere solo le feature più importanti alla classificazione, possa velocizzare di molto entrambi i tempi di predizione per entrambi i modelli (rendendo i modelli di classificazione più scalabili a dataset con molte più istanze).

Si noti però che questo non è il nostro fine ultimo. Noi vogliamo fare feature extraction delle feature più importanti per poi andare a sperimentare come i due modelli si dovessero comportare nel caso in cui queste feature venissero "sporcate" (passaggio che verrà eseguito in una sezione successiva).

Si noti, inoltre, che, come avevamo già anticipato al capitolo precedente, per verificare che stiamo facendo un buon lavoro di estrazione delle feature è necessario trainare da capo i modelli precedentemente visti ad ogni passaggio di feature extraction eseguito; in modo da avere conferma di non peggiorare le performance della classificazione eliminando troppe features, o peggio eliminare features importanti per la classificazione. Ci salveremo poi solo l'ultima versione dei modelli che sono in grado di valutare i dati leggendo solo le feature più importanti di tutto il dataset; ossia i **BASELINE MODEL** per il dataset contenente solo le feature più importanti.

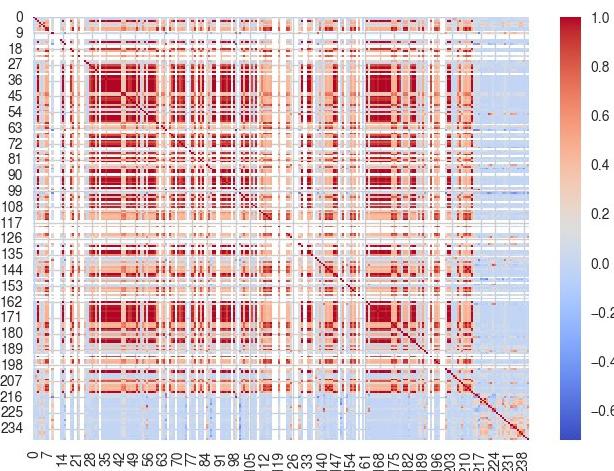
Dunque i passaggi che andremo ad eseguire per effettuare feature extraction sono:

1. Analisi bivariata della correlazione - Eliminazione feature ridondanti
2. Analisi bivariata della correlazione - Highlight delle feature più correlate con il target
3. RFE-CV (Recursive feature elimination)

Nota che tutte queste operazioni vengono effettuate su una copia del dataset, dal quale passo passo andiamo ad eliminare delle feature in più rispetto l'elaborazione precedente.

### 4.1 ANALISI BIVARIATA DELLA CORRELAZIONE - ELIMINAZIONE FEATURE RINDONDANTI:

Come abbiamo già notato in precedenza dall'operazione di data exploration, ci siamo accorti che molte feature potevano avere lo stesso valore informativo (in quanto avevamo visto che molte colonne erano formate unicamente, o quasi, da zeri). Per provare una volta per tutte questa nozione generiamo una heatmap del dataset:



Dato che dal grafo risulta evidente che molte feature combaciano quasi perfettamente (correlazione = 1); possiamo procedere con l'eliminazione delle feature rindondanti. Questo perché se due features non sono indipendenti l'una dall'altra hanno una correlazione assoluta elevata e le informazioni che offrono per i nostri modelli sono sostanzialmente le stesse. Le features correlate, quindi, in generale non migliorano i modelli, quindi se ne può eliminarne una in quanto ridondante.

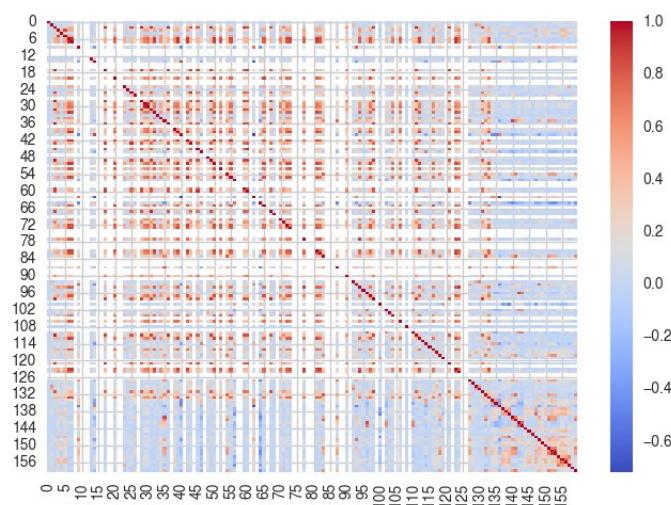
Si sviluppa quindi una funzione che permette di selezionare feature altamente correlate, più precisamente la funzione tornerà in output l'elenco di tutti i feature names, tutte le feature all'interno di questa lista sono correlate con un'altra feature per almeno X%, dove X è un parametro preso in input (nel nostro caso avremo del X=95%). Per sicurezza, per evitare di eliminare troppe feature, si decide di tenere una soglia di correlazione molto alta (appunto del 95%), in modo da eliminare tutte le colonne doppioni che corrispondono con un'altra feature per una soglia di correlazione molto alta, maggiore o uguale a 95%.

#### Nota: L'importanza della validazione dopo la rimozione

Prima di procedere con l'individuazione e successiva rimozione delle feature ridondanti calcoliamo prima la divisione in train e test set perché volgiamo applicare il metodo appena definito **solo sul train set** per poi verificare, tramite validazione con i modelli, se produca buoni risultati anche sul test set. Se si andasse immediatamente ad applicare la funzione sopra descritta direttamente su tutto il dataset si causerebbe overfitting per il train dei modelli di validazione. Qualora i nuovi modelli trainati sul dataset troncato producano score di classificazione simili alla baseline della sezione 3 allora possiamo applicare la rimozione delle stesse feature alla copia contenente tutto il dataset e procedere allo step successivo.

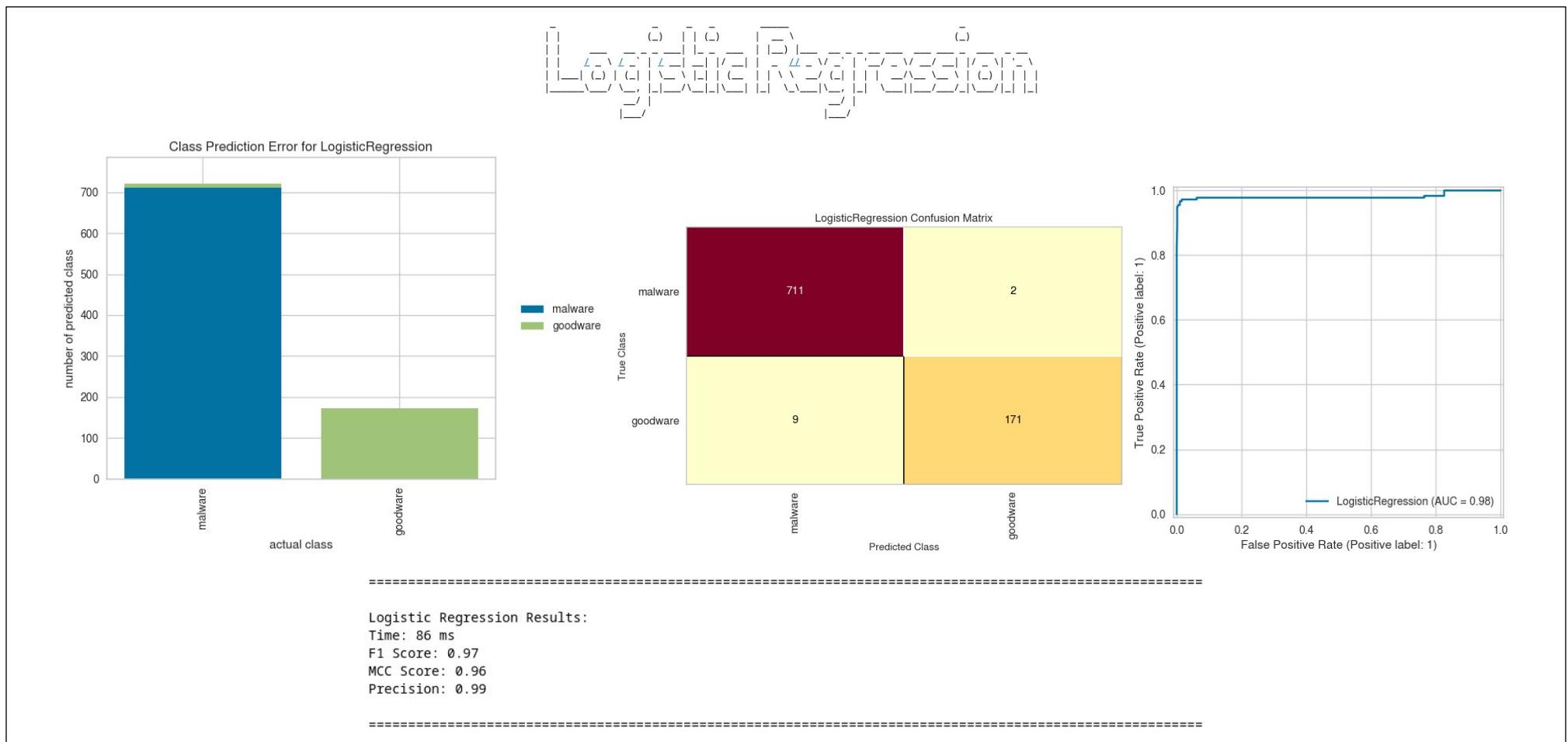
Se non effettuiamo questa operazione di validazione corriamo dunque il rischio di andare ad eliminare feature molto importanti e peggiorare di molto la classificazione. Per questo motivo seguiremo questa procedura di validazione divisione del dataset in train e test per validare le performance su anche tutte le elaborazioni seguenti.

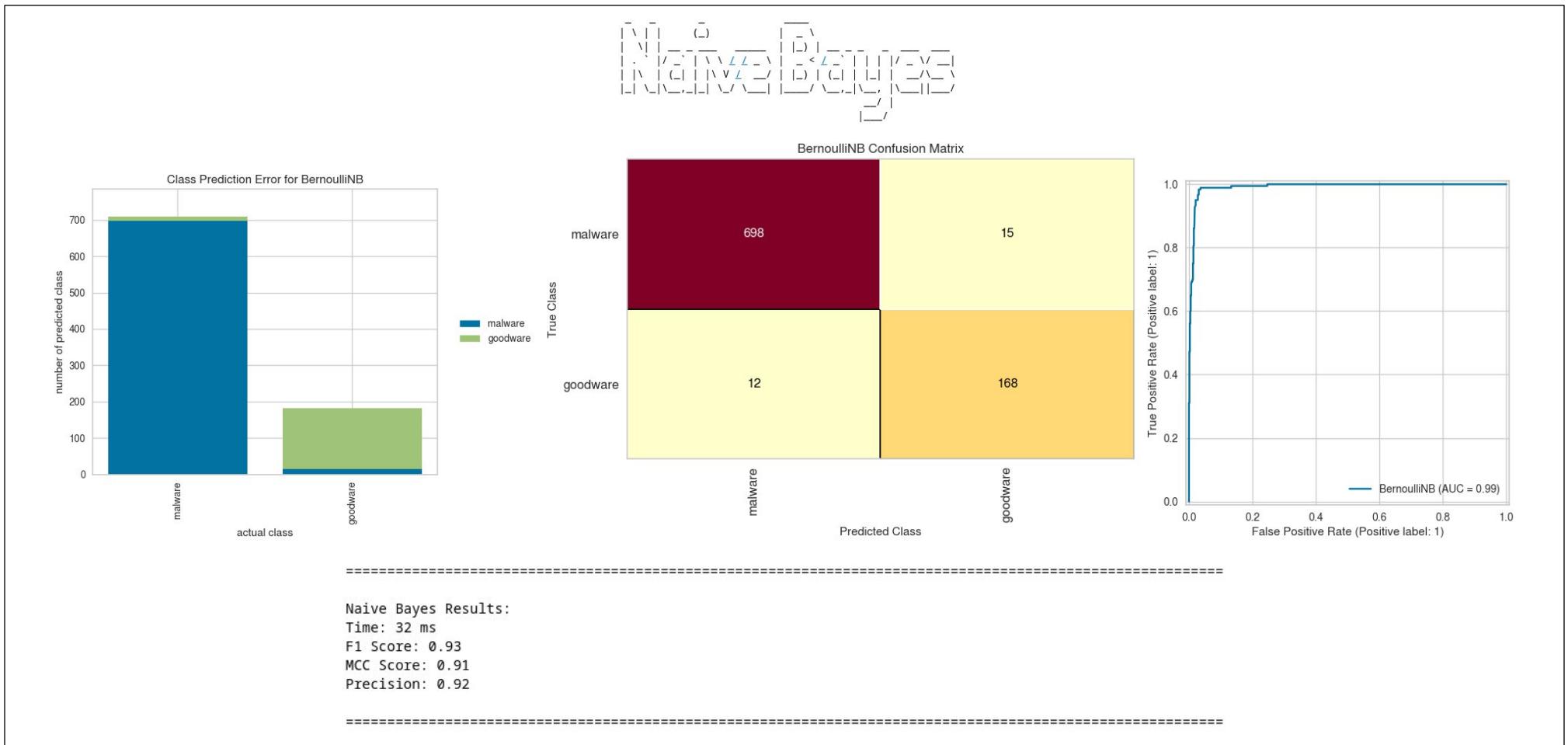
Possiamo dunque procedere a richiamare la funzione appena decritta, questo è il risultato che otteniamo:



Otteniamo quindi degli ottimi risultati. Possiamo confermare che in seguito alla rimozione la situazione sia molto migliorata, è chiaramente visibile che le zone rosse al di fuori della diagonale (seppur ancora sporadicamente presenti) sono diminuite moltissimo. riusciamo infatti a droppare ben 81 features, facendo rimanere solo 160 features.

Possiamo quindi procedere con l'addestramento dei 2 nostri modelli di validazione, per verificare se le performance non diminuiscono in seguito all'eliminazione di queste features, eccone i risultati:





Entrambi i modelli di validazione riportato risultati di classificazione molto simili alla baseline. Addirittura il modello di Logistic Regression riporta le stesse identiche soglie di F1-score e MCC-score, riportando persino lo stesso identico numero di falsi positivi e falsi negativi. Per quanto riguarda, invece, le performance di classificazione di Naive Bayes queste addirittura migliorano rispetto la baseline. Si vuole far notare inoltre che oltre agli ottimi score di classificazione ottenuti sono anche diminuiti significativamente le soglie di tempo. Possiamo dunque confermare il successo di questa operazione e procedere con l'eliminazione delle feature per passare al passaggio successivo.

## 4.2. ANALISI BIVARIATA DI CORRELAZIONE - HIGHLIGHT FEATURE PIÙ CORRELATE AL TARGET

A differenza della correlazione a coppie di feature vista prima, in cui è desiderabile una correlazione bassa tra due feature diverse, in questo caso vorremmo che le nostre features avessero un'elevata correlazione con il target. Se una feature ha una bassa correlazione con il target, significa che non è una feature utile per prevedere il target e, pertanto, potrebbe essere rimossa.

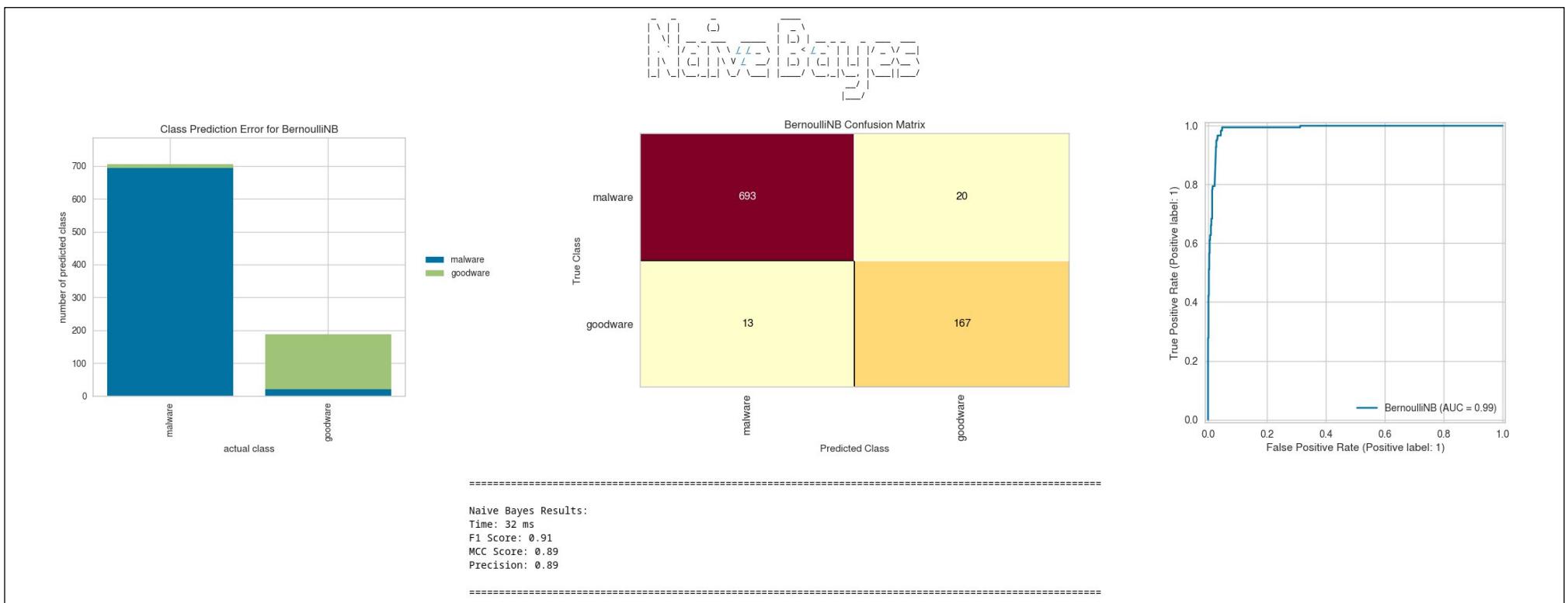
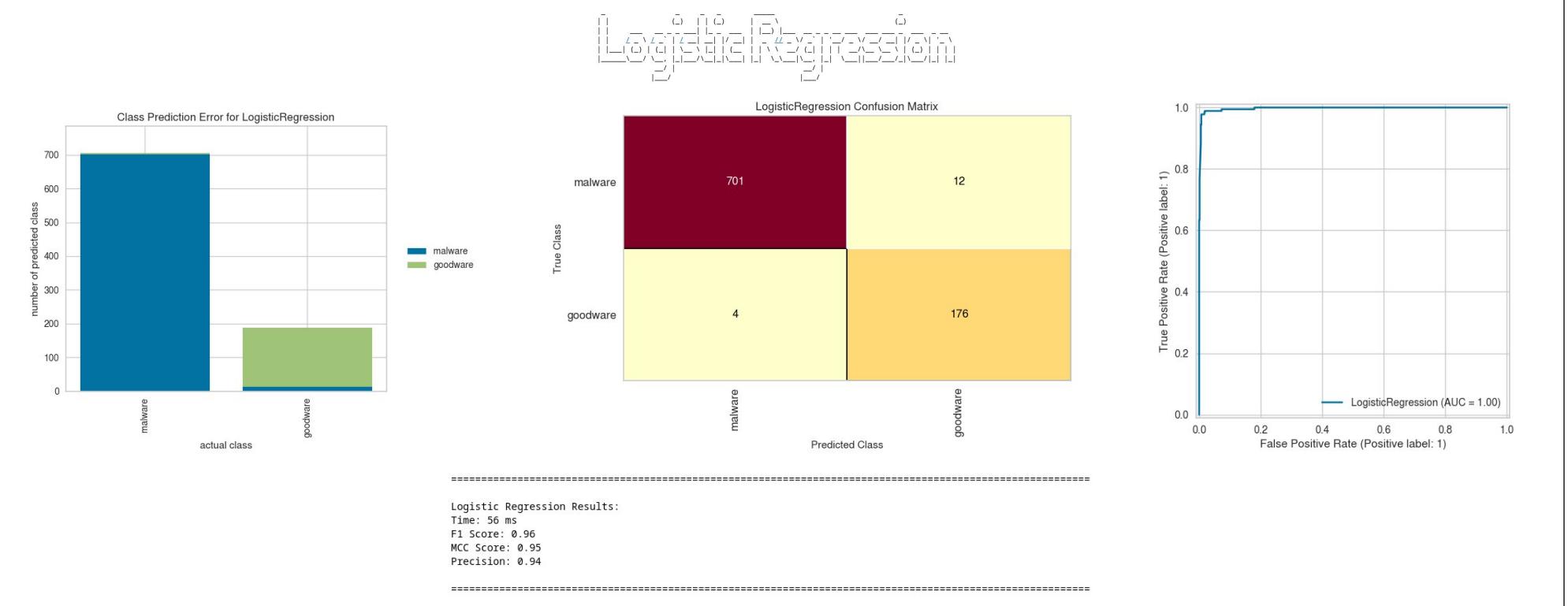
Anche qui dunque si sviluppa una nuova funzione. Questa calcolerà la correlazione di ciascuna feature con il target e poi restituirà il nome delle colonne che hanno una correlazione strettamente inferiore alla soglia percentuale ( $X$ ) scelta. Siccome dall'analisi del dataset fatta precedentemente abbiamo notato che ci sono molte poche feature con un alta correlazione con il target, per evitare di eliminare feature che potrebbero comunque essere utili alla classificazione a discapito della loro correlazione con il target si decide di mantenere una soglia  $X$  molto bassa del 5%.

Come anticipato, dunque, si procede con la suddivisione in train e test set, e si richiama questa funzione sul train-set per individuare quali e quante feature eliminare. I risultati ottenuti dalla funzione sono i seguenti:

Si individua un totale di 84 feature con una correlazione inferiore al 5%, dopo l'eliminazione  
rimarrebbero solo 76 features.

I risultati sembrano abbastanza allarmanti, considerando le 81 feature precedentemente eliminate e aggiungendone altre 84 si arriverebbe ad un totale di 165. Individuare così tante feature da eliminare potrebbe far pensare che le performance potrebbero peggiorare, in quanto stiamo andando a perdere moltissimo valore informativo. Se però andiamo, nuovamente, a creare 2 nuovi modelli di validazione sul dataset troncato otteniamo, ancora una volta, ottimi risultati (che vengono mostrati di seguito, alla pagina successiva). Possiamo infatti notare che le performance di classificazione diminuiscono dalla baseline solo di un misero 1% o 2%. Infatti anche il numero di falsi positivi e negativi, per quanto leggermente aumentato, non contribuisce abbastanza a modificare le soglie ne di F1-Score, MCC-score o AUC. Ciò che invece cambia significativamente sono i tempi di elaborazione che, come ci potevamo aspettare droppando 165 features, diminuiscono significativamente, per il modello di Logistic Regression siamo infatti addirittura a dimezzare il tempo di train+test dal risultato della baseline.

Dati i risultati inaspettatamente positivi possiamo dunque procedere con un ulteriore drop delle feature dal database di copia e andare al prossimo step di feature extraction.



#### **4.3. RFE-CV (RECURSIVE FEATURE ELIMINATION):**

RFE è una tecnica che prende in input un modello ed il dataset del quale vogliamo diminuire la numerosità di feature. Questa tecnica, infatti, come indica il nome, serve a diminuire la numerosità delle colonne delle colonne del dataset, effettua automaticamente le seguenti operazioni:

**1. Selezione iterativa delle feature:**

RFE opera selezionando iterativamente un sottoinsieme di feature dal dataset. Inizia con tutte le feature nel dataset e addestra il modello su di esse.

**2. Valutazione delle feature:**

Dopo aver addestrato il modello, viene calcolata una metrica di importanza delle feature, che può essere basata su coefficienti (nel caso della regressione lineare, per esempio), importanza delle variabili (nel caso degli alberi decisionali) o altre metriche rilevanti per il tipo di modello utilizzato.

**3. Eliminazione delle feature meno importanti:**

Dalle feature attualmente selezionate, RFE rimuove quelle che sono considerate meno importanti secondo la metrica di importanza definita.

**4. Valutazione delle prestazioni:**

il modello viene valutato utilizzando il sottoinsieme ottimale di feature selezionate tramite RFE.

**5. Ripetizione:**

Il processo viene ripetuto ricorsivamente con un numero inferiore di feature, continuando fino a raggiungere un numero prefissato di feature o fino a quando il miglioramento delle prestazioni del modello non è significativo.

Il vantaggio principale di RFE è che permette di ottenere un modello più semplice e interpretabile, mantenendo (o migliorando) allo stesso tempo le prestazioni predittive. Rimuovendo le feature meno informative, si riduce anche il rischio di overfitting, specialmente in presenza di un gran numero di feature rispetto al numero di istanze nel dataset.

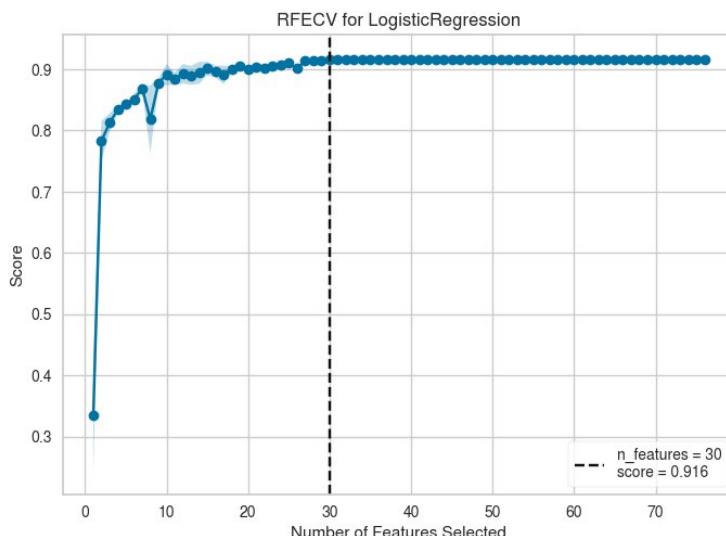
Tuttavia, è importante notare che RFE può richiedere un tempo di calcolo significativo, specialmente con dataset molto grandi e complessi (essendo un metodo ricorsivo non scala bene). Si vuole far notare, dunque, che avremmo potuto applicare RFE fin da subito (senza applicare le elaborazioni precedenti) ma in quel caso i tempi di computazione sarebbero diventati non accettabili (intorno ai 40 minuti). Invece applicando RFE dopo le 2 fasi precedenti di feature extraction che abbiamo precedentemente effettuato riusciamo ad ottenere un elaborazione di RFE in circa 2 minuti; che è sempre un tempo alto, ma a noi va bene comunque perché a noi importa ottenere performance temporali alte quando vogliamo addestrare nuovi modelli. Questa operazione va eseguita una sola volta e diminuirà i tempi di addestramento dei prossimi modelli di molto.

### Nota: Applicazione di RFE solo sul modello Logistic Regression

In questo caso RFE verrà applicato dando in input solo il modello di Logistic Regression per i seguenti 3 motivi:

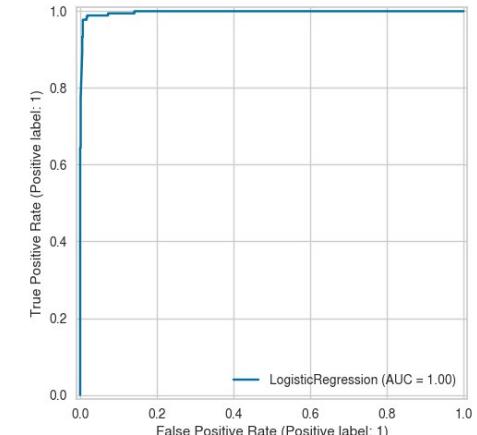
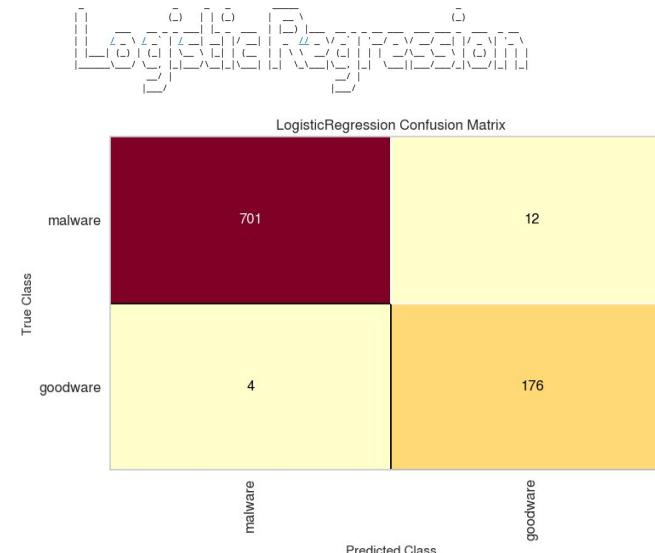
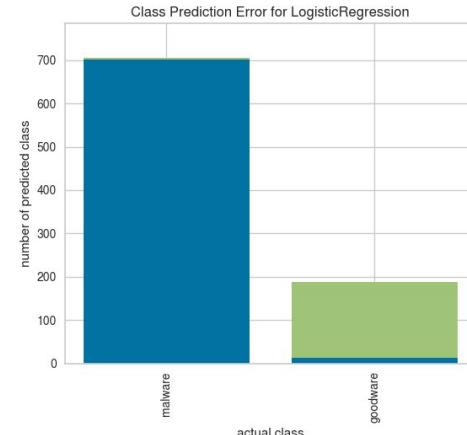
1. Computare un miglioramento delle prestazioni su entrambi i modelli costerebbe troppo tempo
2. Se si ha un miglioramento sul modello di Logistic Regression lo si ha sicuramente anche sul modello Naïve Bayes
3. I modelli di BernoulliNB non offrono un metodo intrinseco per valutare l'importanza delle feature. I metodi Naïve Bayes funzionano determinando le probabilità condizionali e incondizionate associate alle feature e prevedono la classe con la probabilità più alta. Pertanto, non sono presenti coefficienti calcolati o associati alle funzionalità utilizzate per addestrare il modello. Pertanto non è possibile applicargli RFE

Per confermare ciò che abbiamo appena detto procediamo, come sempre, applicando prima la trasformazione di RFE solo sul train set. In questo modo riusciamo ad ottenere il risultato di ridurre ancora di più il numero di feature a 30 sole colonne, che da ora in poi chiameremo “feature più importanti”:

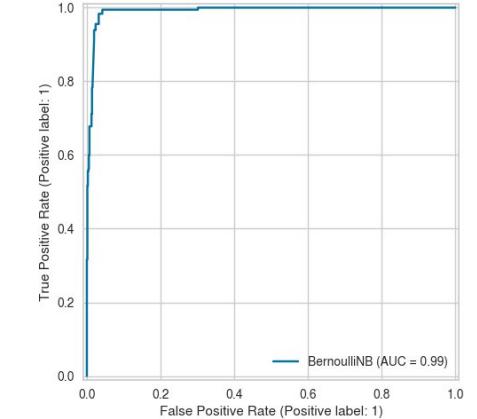
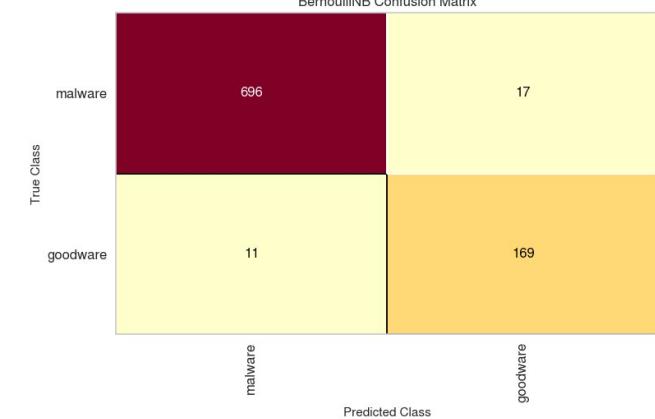
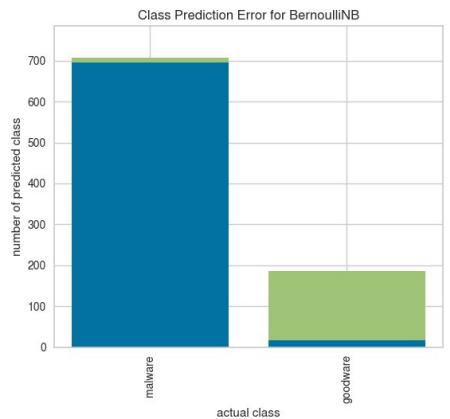


Procedendo poi con la validazione dei risultati (i quali risultati sono disponibili a pagina seguente) riusciamo, ancora una volta, a confermare che le prestazioni classificazione di entrambi i modelli è rimasta ottima, e sorprendentemente supera addirittura gli score precedenti (avendo un numero minore di falsi positivi e falsi negativi). Inoltre le prestazioni temporali migliorano nuovamente, ora per entrambi i modelli abbiamo tempi di train+test che sono inferiori alla metà della prima baseline.

Dato che questa è l'ultima operazione di feature extraction che facciamo ci salveremo gli score ottenuti in questa fase come nuova baseline. Questa non andrà a sostituire la precedente, sarà infatti utilizzata con la precedente: verranno utilizzate entrambi poi in fase di inserimento di rumore. La prima baseline verrà utilizzata per paragoni quando andremo a sporcare tutto il dataset, la seconda baseline (quella che abbiamo appena ottenuto) verrà utilizzata quando andremo a sporcare solo le “feature più importanti”.



```
Logistic Regression Results  
Time: 41 ms  
F1 Score: 0.96  
MCC Score: 0.95  
Precision: 0.94
```



Naive Bayes Results  
Time: 18 ms  
F1 Score: 0.92  
MCC Score: 0.90  
Precision: 0.91

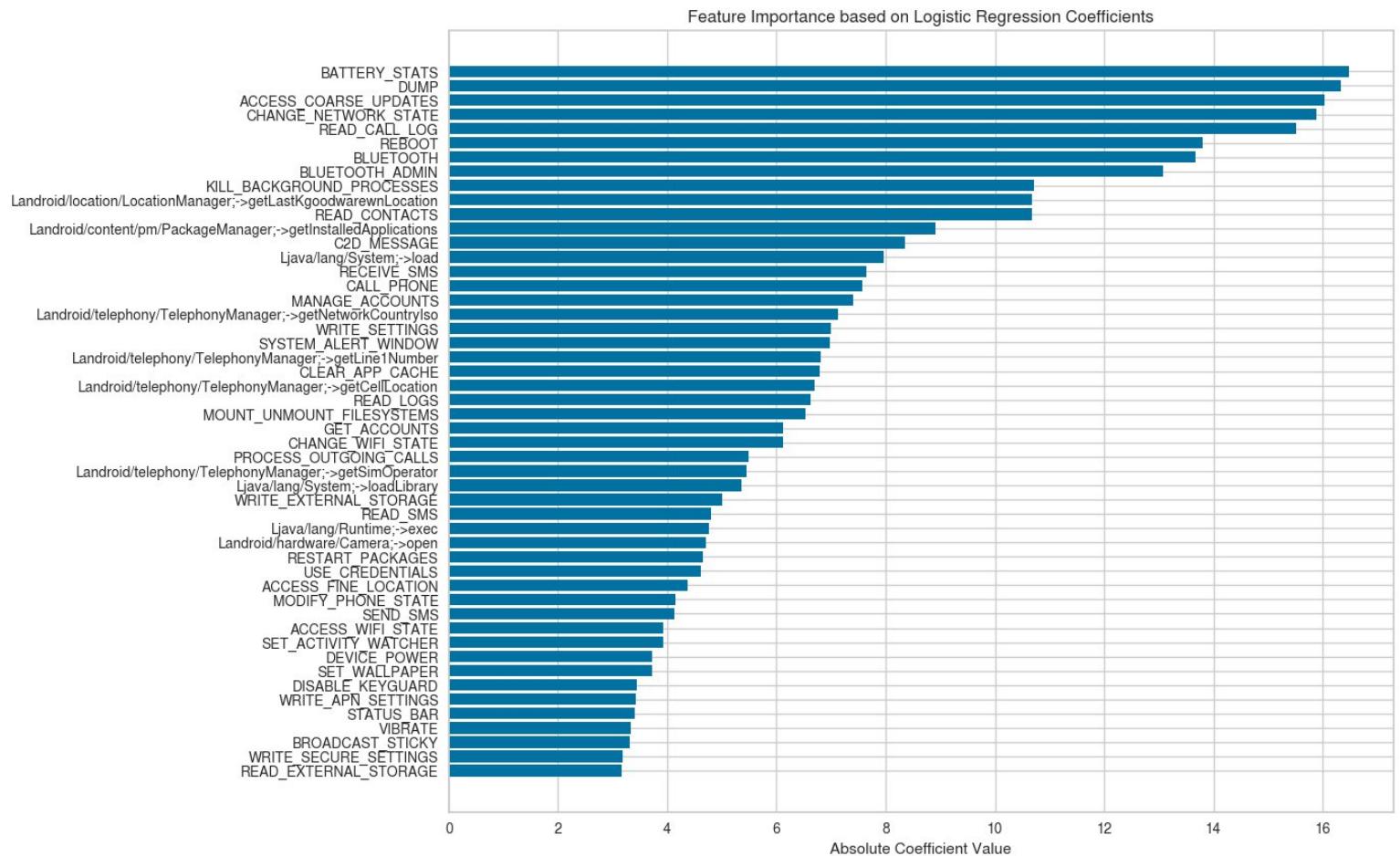


## 5. DATA EXPLORATION PT2: FEATURE CHE MEGLIO DISCRIMINANO:

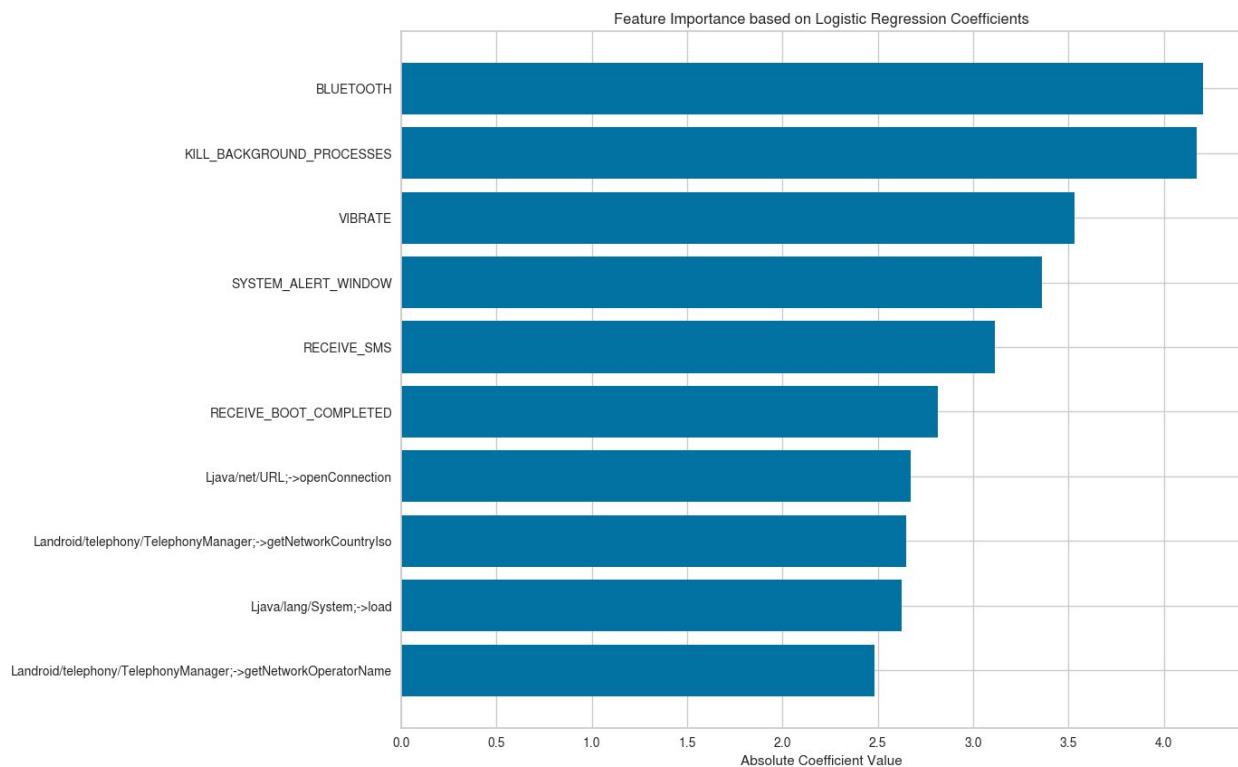
Dato che abbiamo creato una baseline per ogni modello, sia sull'intero dataset che sul range più piccolo di feature più importanti per entrambi i modelli, possiamo ora andare a vedere quali sono le feature che più contribuiscono alla classificazione fatta da ogni modello. Ossia ciò che vogliamo a fare in questa sezione è andare a stilare 4 liste una per ogni baseline.

### 5.1. VISUALIZZAZIONE DEI RISULTATI:

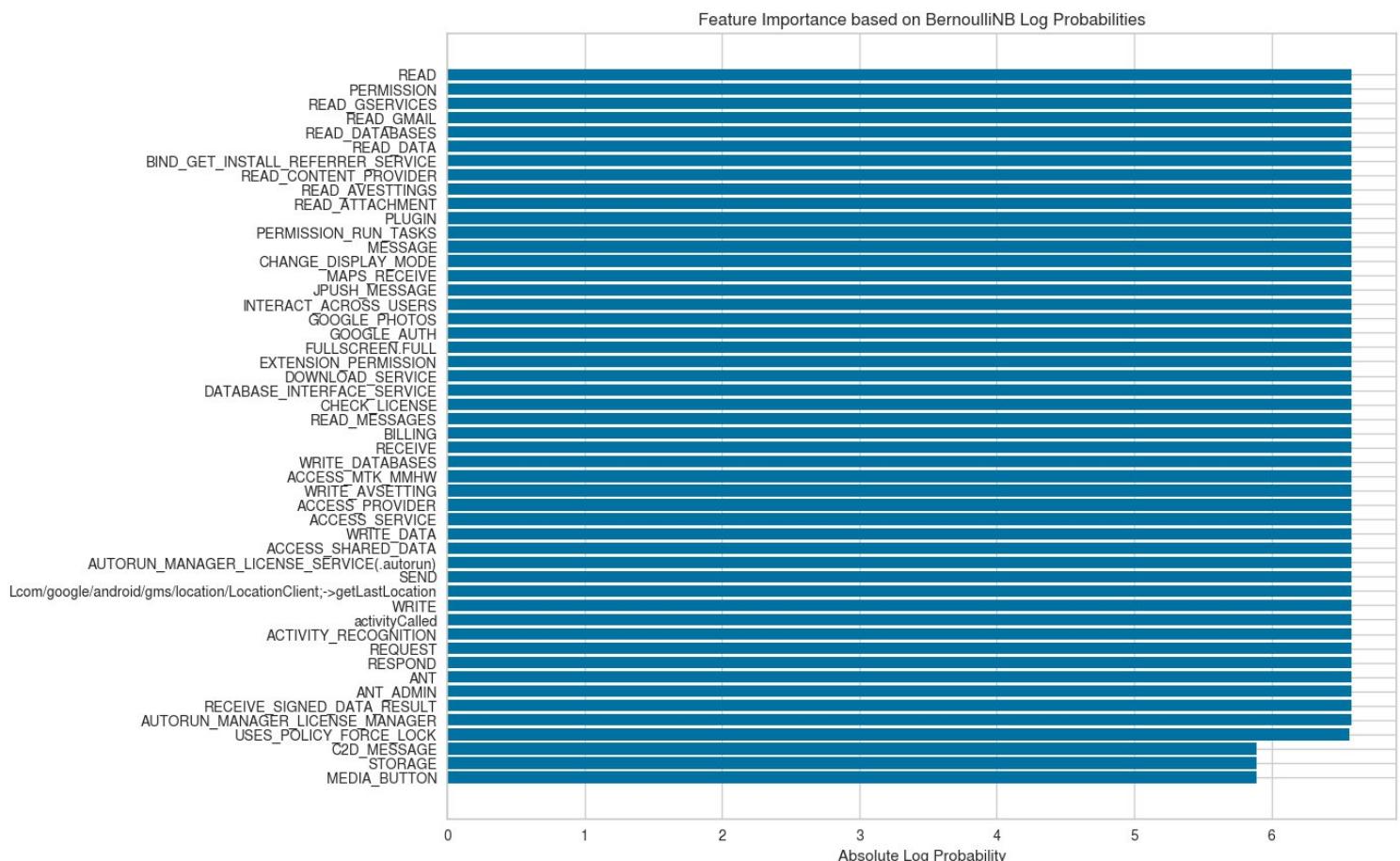
1. Top 50 feature discriminanti che più contribuiscono alla classificazione per il modello di Logistic regression trainato su tutto il dataset:



2. Top 10 feature più discriminanti che più contribuiscono alla classificazione per il modello di Logistic regression trainato sul dataset con solo le feature più importanti:



3. Top 50 feature discriminanti che più contribuiscono alla classificazione per il modello di Bernoulli Naive Bayes trainato su tutto il dataset:

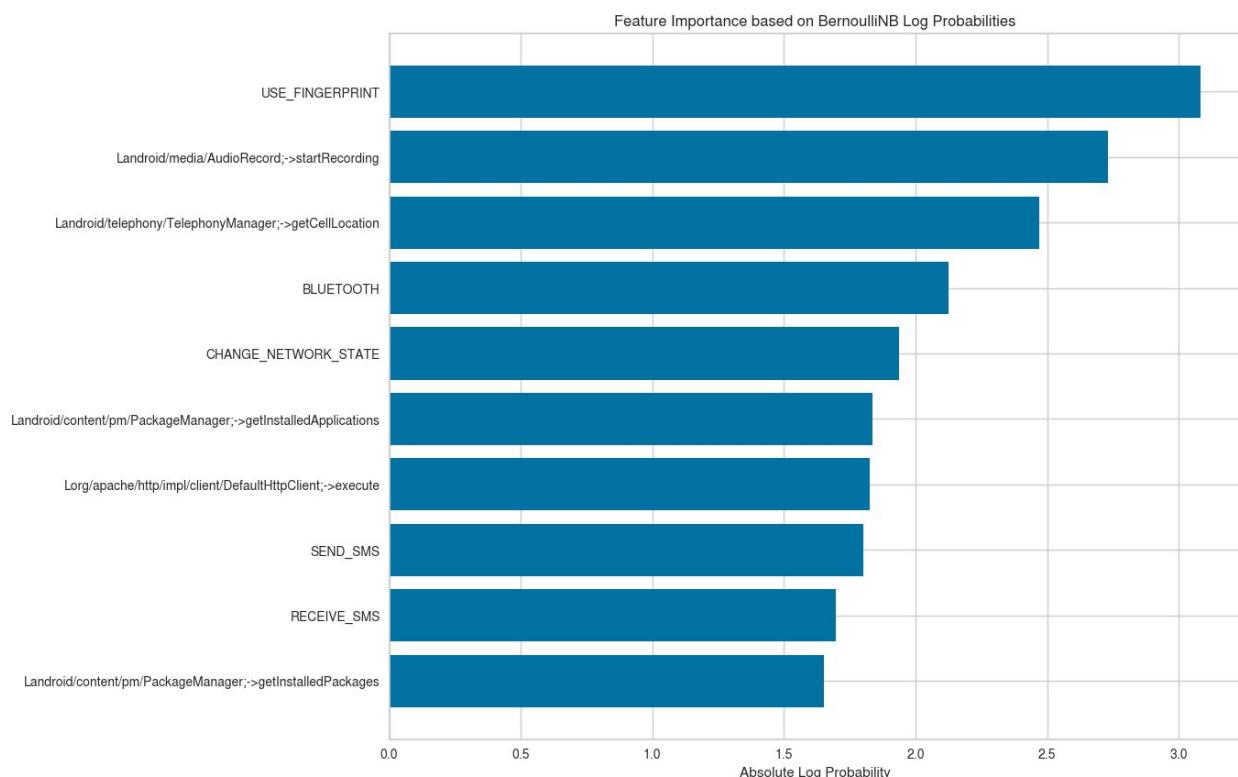


#### Nota: Considerazioni Top 50 Naive Bayes - dataset full

Dal risultato precedente notiamo qualcosa di insolito, moltissime feature sono in pareggio alla prima posizione, che è un risultato un po' particolare probabilmente dovuto a 2 fattori:

1. Dati sparsi: come già dimostrato in diverse occasioni abbiamo molte feature con valori binari dove la maggior parte di questi valori è pari a totalità di 0 / totalità di 1, quindi le probabilità di log potrebbero non variare in modo significativo tra queste feature.
2. Ridondanza delle features: Abbiamo già avuto modo di verificare anche questo aspetto, molte features hanno informazioni simili o ridondanti, questo porta a probabilità di log simili.

4. Top 10 feature più discriminanti che più contribuiscono alla classificazione per il modello di Bernoulli Naive Bayes trainato sul dataset con solo le feature più importanti:



#### Nota: Considerazioni Top 10 Naive Bayes - dataset feature più importanti

I problemi notati precedentemente si risolvono andando ad utilizzare la stessa funzione ma con il dataset di feature più importanti (che ha quindi rimosso sparsità dati e feature ridondanti), confermando che le ipotesi formulate precedentemente sono corrette.

## 5.2. OSSERVAZIONI GENERALI

Si vuole far notare che presi gli stessi modelli passando dal dataset completo al dataset con solo le feature più importanti le colonne più discriminanti cambiamo. Questo è un comportamento strano, se una colonna è particolarmente informativa in un dataset lo dovrebbe essere anche nell'altro a parità di modello utilizzato. Questo comportamento è probabilmente dovuto al fatto che nel passare dal dataset completo al dataset ristretto abbiamo eliminato le colonne con lo stesso valore informativo e, tra queste che abbiamo eliminato, è possibile ci fossero proprio quelle colonne che "mancano" nel dataset con solo le feature più importanti.

## 5.3. NOTE IMPLEMENTATIVE E TEORICHE:

### 5.2.1. PER LOGISTIC REGRESSION:

Per prendere le feature più discriminanti dai modelli di Logistic Regression si usa la funzione ``model.coef_.flatten()`` di modo da predire i coefficienti theta per ogni feature di ogni modello, dove abbiamo che:

- I coefficienti positivi significano che un aumento del numero di istanze positive di quella feature porta ad un aumento di target positivi.
- I coefficienti negativi, al contrario, significano che un aumento del numero di istanze positive di quella feature porta ad una diminuzione di target positivi.

Con questo coefficiente se ne farà il valore assoluto, infatti il valore assoluto di un coefficiente indica la forza della relazione tra la feature ed il target: valori assoluti più grandi indicano che la feature ha un impatto maggiore sulle previsioni del modello. Per determinare le feature più importanti del modello, quindi, basta guardare i valori assoluti dei coefficienti. Le feature con valore assoluto maggiore sono più importanti. Poi ognuno tutti questi valori assoluti verranno ordinati dal più grande al più piccolo e si terranno solo le top X feature, dove X è un numero passato come parametro.

### 5.2.2. PER BERNOUlli NAIVE BAYES:

Come abbiamo già anticipato prima parlando di RFE, il Bernoulli Naive Bayes non ha nessun coefficiente sul quale basarsi per andare a fare una stima di rank di feature. Per questo motivo la costruzione di una funzione per individuare e rankare le feature più importanti può essere un po' più problematica della precedente. A questo scopo si sceglie di utilizzare l'attributo ``feature_log_prob_`` del modello di NB. Questo attributo rappresenta il logaritmo della probabilità condizionata che ciascuna feature sia data ciascuna classe. Questo può essere interpretato come un'indicazione di quanto sia importante ciascuna caratteristica per la classificazione ma può portare poi ad avere problemi di ranking in casi di dati sparsi e ridondanti, esattamente come abbiamo visto prima.



## 6. SPORCARE I DATASET:

### 6.0. INTRODUZIONE:

Arriviamo dunque al cuore del progetto, la fase in cui andiamo a sporcare il dataset inserendo rumore di vario genere. Per l'inserimento di rumore proponiamo quindi le seguenti operazioni:

- Sostituzione di valori con valori casuali
- Sostituzione di valori con valori nulli
- Sostituzione di valori con valori opposti

In particolare, con ciascuna delle operazioni appena citata (al di fuori dell'ultima) seguiremo questi step:

1. **Step 1:** Logistic Regression & Naive Bayes - sporcare le feature meno significative, ossia pool di tutte le feature tranne quelle più discriminanti (sul dataset intero)
2. **Step 2:** Logistic Regression & Naive Bayes - sporcare le rispettive feature più discriminanti di ogni modello (sul dataset intero)
3. **Step 3:** Logistic Regression & Naive Bayes - sporcare tutte le features (sul dataset intero)
4. **Step 4:** Logistic Regression & Naive Bayes - sporcare le rispettive feature più discriminanti di ogni modello (dataset ristretto)

#### Nota: Modalità di train, test e visualizzazione dei risultati

Come già specificato nel caso in cui si andava a fare rimozione di feature, anche in questi casi in cui andremo a sporcare il dataset applicheremo i cambiamenti **solo sul train-set** e non sul test-set, se si facesse altrimenti una volta addestrato il modello sui dati sporchi, quando adremmo a testarlo sui dati di test sporcati nel medesimo modo registreremmo inevitabilmente una situazione di overfitting.

Dunque, per ognuno degli step sopracitati, quando andremo a creare un nuovo modello seguiremo la seguente modalità:

1. Divisione tra train e test set
2. Inserimento del rumore in una copia del train-set
3. Visualizzazione differenze tra train-set sporcato e train-set originale.

Nota che questa funzione per visualizzare le differenze serve anche per ritornare una lista di feature che sono cambiate per meno del 20% o più dell'80%, perchè:

- se una colonna è cambiata per meno del 20% dei dati allora non è cambiata così tanto
- avendo feature binarie, se una feature è cambiata per l'80% o più contiene ancora lo stesso valore informativo, in quanto, semplicemente, tutti gli 0 sono diventati 1 e viceversa, quindi la correlazione tra la feature in questione e la feature "sporcata" sarà sempre alta. Il che vuol dire che di fatto è come non aver sporcato per nulla la feature in questione.

4. Addestramento del nuovo modello sul train-set rumoroso + test con test-set originale
5. Salvataggio dei risultati di performance di classificazione e temporali in un array.

Una volta terminati tutti gli step utilizzeremo l'array sul quale sono salvati tutti i risultati per fare confronti con la rispettiva baseline, riflessione sui risultati ed infine trarre delle conclusioni.

Inoltre ciascuna operazione (con tutti i suoi step) verrà eseguita più volte modificando il valore percentuale di dati che andremo a sporcare, di volta in volta aumentando progressivamente il numero di dati sporcati. Abbiamo infatti che ciascuna operazione verrà ripetuta 4, una per ciascuno dei seguenti valori percentuali:

- 40%
- 55%
- 70%
- 85%

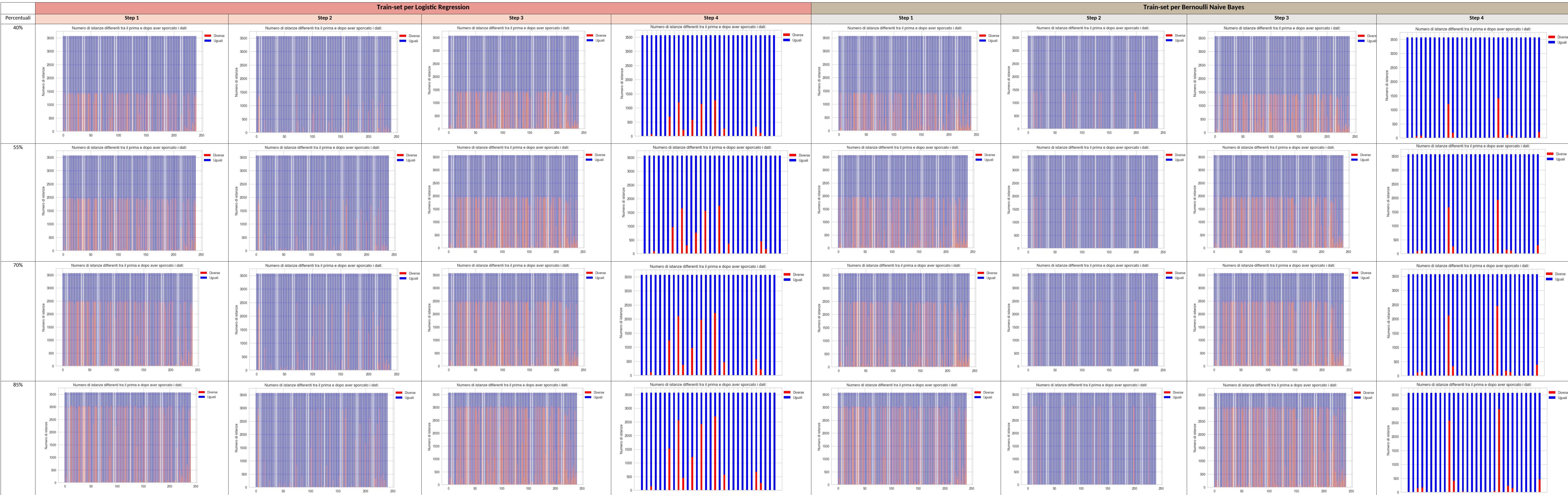
Per quanto riguarda invece l'ultima operazione di Sostituzione di valori con valori opposti seguiremo una struttura diversa per motivi che spiegheremo in seguito.

## 6.1. SOSTITUZIONE DI VALORI CON VALORI CASUALI:

Prima di procedere è estremamente importante far notare che per l'inserimento di valori casuali è possibile che violeremo il vincolo di riproducibilità dei risultati che ci siamo posti nell'introduzione. Questo perché la funzione di inserimento valori casuali all'interno del dataset utilizza, chiaramente, delle funzioni di random sia per scegliere casualmente il valore da inserire ma anche per scegliere quali celle all'interno della feature andranno sostituite. Anche impostando un seed per le funzioni di random 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 dipendenti dal kernel del notebook.

### 6.1.1. VISUALIZZAZIONE DIFFERENZE INTRODOTTE CON INSERIMENTO VALORI CASUALI:

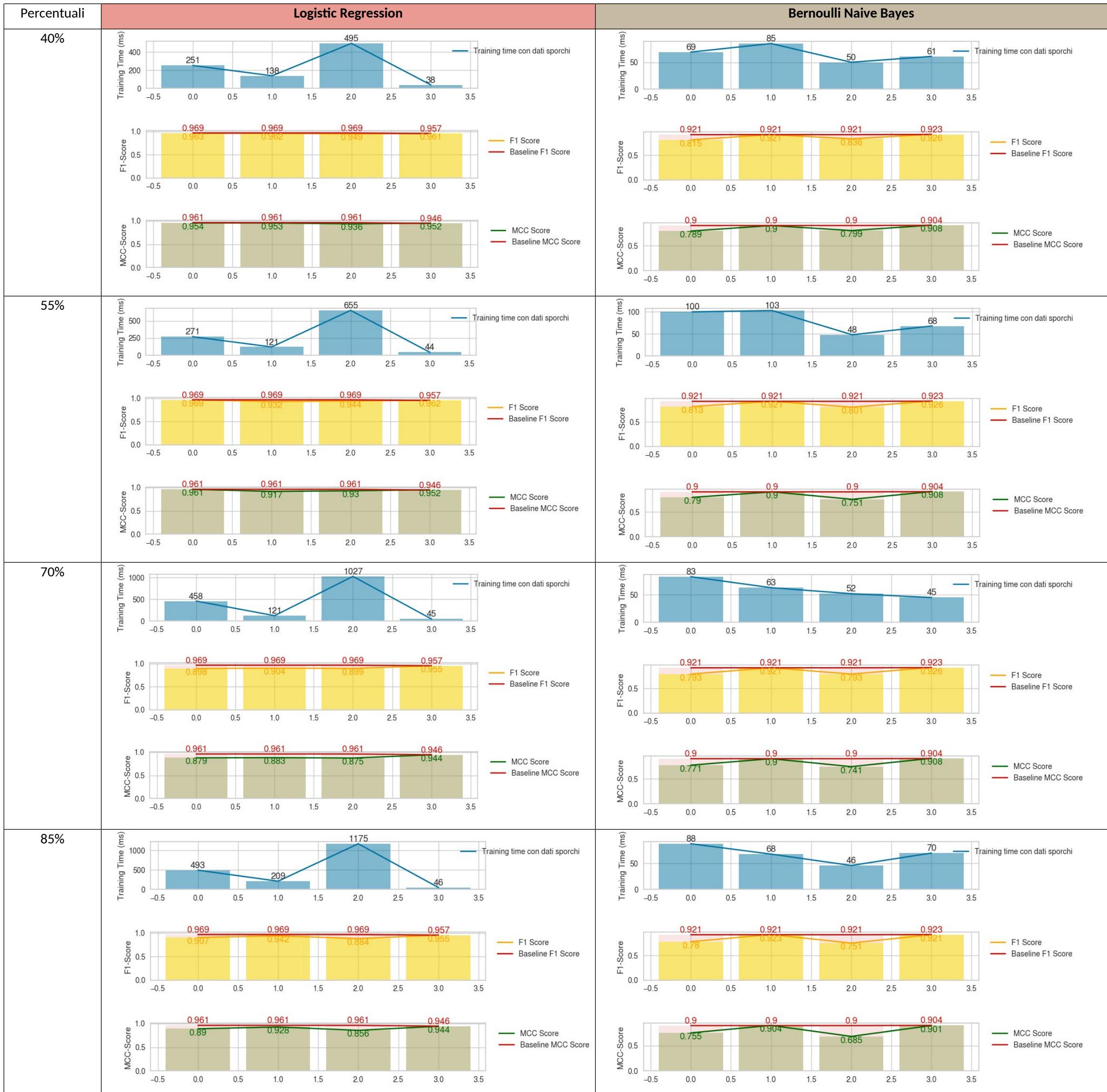
Di seguito vengono visualizzati tutti i grafici che evidenziano le differenze tra train-set pulito e train-set in seguito all'inserimento di valori casuali:



## 6.1.2. VISUALIZZAZIONE RISULTATI PRODOTTI DAI MODELLI

Guida alla lettura dei risultati:

- Colonna indicata da 0.0 = Risultati Step 1: Logistic Regression & Naive Bayes - sporcare le feature meno significative (sul dataset intero)
- Colonna indicata da 1.0 = Risultati Step 2: Logistic Regression & Naive Bayes - sporcare le rispettive feature più discriminanti di ogni modello (sul dataset intero)
- Colonna indicata da 2.0 = Risultati Step 3: Logistic Regression & Naive Bayes - sporcare tutte le features (sul dataset intero)
- Colonna indicata da 0.0 = Risultati Step 4: Logistic Regression & Naive Bayes - sporcare le rispettive feature più discriminanti di ogni modello (dataset ristretto)



### 6.1.3. CONSIDERAZIONI E CONCLUSIONI SUL DECADIMENTO DELLE PERFORMANCE TRA BASELINE E MODELLI CON DATI SPORCHI:

Come ci aspettavamo vediamo che con l'aumentare della percentuale cadono gradualmente (anche se di poco) anche le performance di classificazione, tranne nel passaggio da 70% a 85% dello step-2 di LogReg, in cui, sorprendentemente le performance migliorano.

Però con qualsiasi percentuale di inserimento dati casuali, le performance di classificazione del modello di Logistic Regression rimangono sempre molto simili alla baseline, cadendo del massimo di 10%.

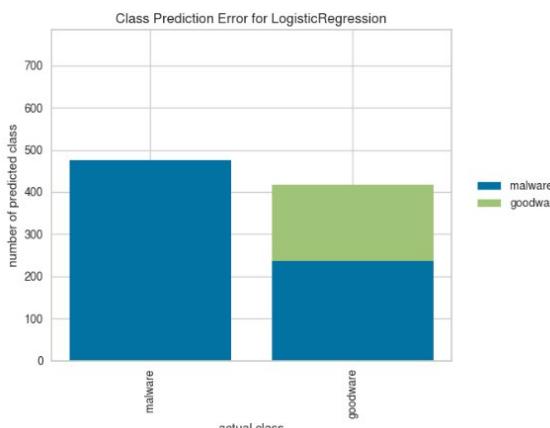
Mentre è facile notare che, come ci aspettavamo, il modello di Naive Bayes è molto più suscettibile alla presenza di dati errati. Vediamo infatti che, indipendentemente dalla percentuale, negli step 1 e 3 (ossia rispettivamente gli step nel quale si inseriscono dati casuali nelle feature meno rilevanti (in totale 191) e in tutte le feature (in totale 241)) si registra un grosso calo delle performance dalla baseline che partono dal 10% e arrivano fino al 20%. Si vuole far notare che, andando a verificare nella tabella di differenze tra dataset (mostrata al punto 6.1.1.) gli step 1 e 3, indipendentemente dalla percentuale, sono quelli in cui il numero di feature sporche diventa molto più elevato rispetto agli step 2 e 4; di conseguenza contengono molti più valori errati.

Un'altra cosa particolare che si vuole però far notare è che è strano non notare un grosso cambiamento nel calo di performance negli step 2 e 4, di tutte le percentuali, di entrambi i modelli. In questi casi si stanno andando a sporcare le feature più discriminanti (nel caso dello step 2 su tutto il dataset e nel caso dello step 4 sul dataset contenente solo le feature più importanti).

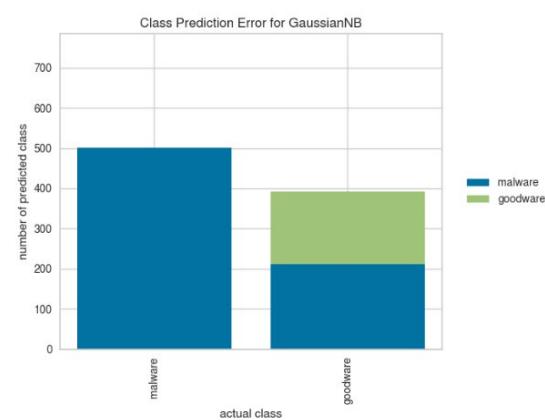
Questa sezione è quindi principalmente dedicata ad analizzare il perchè non osserviamo grossi cambiamenti dalla baseline al modello di LR in tutti i punti e dalla baseline nei punti 2 e 4 di NB.

#### Nota: Riproducibilità dei dati – possibili alterazioni

Nota che, come già specificato, all'avvio di un nuovo kernel si cambieranno i valori di random, pur avendo settato il seme. In questo caso è possibile che si registri un decadimento significativo nelle performance anche su questi punti 2 e 4 appena citati:



```
=====
Logistic Regression Results:
Time: 58 ms
F1 Score: 0.73
MCC Score: 0.54
Precision: 0.73
=====
```



```
=====
Naive Bayes Results:
Time: 41 ms
F1 Score: 0.76
MCC Score: 0.57
Precision: 0.76
=====
```

Però questo risultato non è persistente, anzi capita molto raramente. In questa sezione basiamo le nostre spiegazioni quindi sui risultati più comuni che coincidono, perlopiù, nelle sezioni 6.1.1. e 6.1.2.

Ciò che succede quando non registriamo grossi cambiamenti di performance è, semplicemente, che nel momento in cui i dati vengono sporcati è possibile che, ad alcuni valori all'interno delle colonne, per puro caso, venga riassegnato lo stesso valore esatto (che pensandoci non è neanche così improbabile avendo tutte le feature binarie), invece che un valore errato. Infatti andando a vedere i grafici delle differenza mostrati al punto 6.1.1., se andiamo a vedere i grafici di tutti gli step 4 è facile notare come molte feature che dovevano venire sporcate rimangono comunque molto pulite.

Di conseguenza ciò che succede è che entrambi i classificatori vanno a dare peso maggiore a feature meno sporche per fare la classificazione. In questo modo si spiegherebbe come entrambi i modelli riescano ad ottenere comunque ottimi risultati di classificazione: perché stanno basando la loro classificazione principalmente su feature che sono ancora molto pulite.

Un buon modo per testare questa ipotesi è di:

1. Prendere come esempio di riferimento il punto 4 per entrambi i modelli, in cui è evidente in cui le performance non peggiorano.
2. Riaddestrare 2 nuovi modelli giocattolo prendendo però nel train-set solo le colonne che sono state sporcate di meno, quindi, possiamo dire, che hanno un numero di differenze minore del 20% (o maggiore dell'80%, in quanto se la modifica si avvicina troppo al 100% su feature binarie poi il valore informativo rimane inalterato), e vedere quali sono le nuove performance.
3. Potremo infine comparare le performance dei 2 nuovi modelli a quelle ottenute nel punto 4. Se queste combaciano o sono molto simili allora la teoria è confermata.

Si noti che avrebbe senso aspettarsi che le performance di questo esperimento siano entrambe:

- O leggermente più alte dei modelli precedenti in quanto i nuovi modelli non sono influenzati dalle colonne contenenti più dati sporchi. Quindi ci aspettiamo performance simili ma leggermente più alte.
- O leggermente più basse rispetto ai modelli precedenti in quanto le colonne selezionate per il nostro esperimento (ossia le feature che contengono un numero di differenze minore del 20% dopo essere state sporcate) tra tutte le colonne disponibili in tra le feature più importanti possono comunque venire selezionate quelle con il valore di significatività più basso (o che magari avrebbero bisogno di altre colonne per raggiungere una significatività più alta)

Entrambi i ragionamenti sono corretti (specilmente se si vuole ri-runnare il progetto con un kernel diverso).

Dunque per testare questa ipotesi mi sono salvato in una variabile gli indici delle feature che vengono sporcate per il 20% o meno oppure per il 80% o più delle istanze totali nello step 4 e proviamo ora a comparare i risultati:

Modello Logistic regression	Modello Naive Bayes
ACCESS_COARSE_LOCATION	ACCESS_COARSE_LOCATION
ACCESS_FINE_LOCATION	ACCESS_FINE_LOCATION
BLUETOOTH	BLUETOOTH
CHANGE_NETWORK_STATE	CHANGE_NETWORK_STATE
DISABLE_KEYGUARD	DISABLE_KEYGUARD
GET_TASKS	GET_TASKS
READ_PHONE_STATE	KILL_BACKGROUND_PROCESSES
RECEIVE_SMS	READ_PHONE_STATE
SEND_SMS	RECEIVE_BOOT_COMPLETED
USE_FINGERPRINT	SEND_SMS
WAKE_LOCK	SYSTEM_ALERT_WINDOW
...	...

Dato che abbiamo appurato di avere molte colonne ancora pulite possiamo a questo punto formulare 2 ipotesi:

- **CASO 1:**

Tra queste ci sono ancora features con alta correlazione con il target, quindi per i modelli diventa estremamente facile predirre una buona parte del target.

Se proviamo a verificare questa ipotesi chiamando la funzione per ottenere la correlazione con il target (definita nella fase di Data Exploration) otteniamo i seguenti risultati:

Per modello Logistic regression:

Ljava/net/URL; ->openConnection	0.708878
Landroid/location/LocationManager; ->getLastKnownLocation	0.649943
GET_TASKS	0.560569

Per modello Naive Bayes:

RECEIVE_BOOT_COMPLETED	0.752251
Ljava/net/URL; ->openConnection	0.708878
Landroid/location/LocationManager; ->getLastKnownLocation	0.649943
GET_TASKS	0.560569

Esatto, è confermato quindi che stiamo cadendo in questa casistica per entrambi i modelli, il che vuol dire che finchè una qualsiasi di queste feature con correlazione al 65-70% rimarrà intatta allora avremo garantito che il risultato di precisione sarà sempre superiore al 65-70%. Nota che però è possibile cadere contemporaneamente anche nell'ipotesi 2.

- **CASO 2:**

Tra queste feature meno sporche del 20% e più sporche del 80% precedentemente salvate in una variabile ci sono ancora features più discriminanti che, per casualità, non si sono sporcate abbastanza. Pertanto i modelli possono imparare solo su quelle, trascurando le altre che sono state sporcate di più. Se controlliamo la presenza di feature discriminanti (dei rispettivi modelli) all'interno della variabile contenente le feature non sporcate otteniamo i seguenti risultati:

Per modello Logistic regression:

['Landroid/telephony/TelephonyManager; ->getNetworkCountryIso',
'Ljava/net/URL; ->openConnection',
'Landroid/telephony/TelephonyManager; ->getNetworkOperatorName',
'BLUETOOTH',
'RECEIVE_SMS']

Per modello Naive Bayes:

['Lorg/apache/http/impl/client/DefaultHttpClient; ->execute',
'USE_FINGERPRINT',
'Landroid/telephony/TelephonyManager; ->getCellLocation',
'CHANGE_NETWORK_STATE',
'Landroid/content/pm/PackageManager; ->getInstalledApplications',
'Landroid/media/AudioRecord; ->startRecording',
'SEND_SMS',
'BLUETOOTH',
'Landroid/content/pm/PackageManager; ->getInstalledPackages']

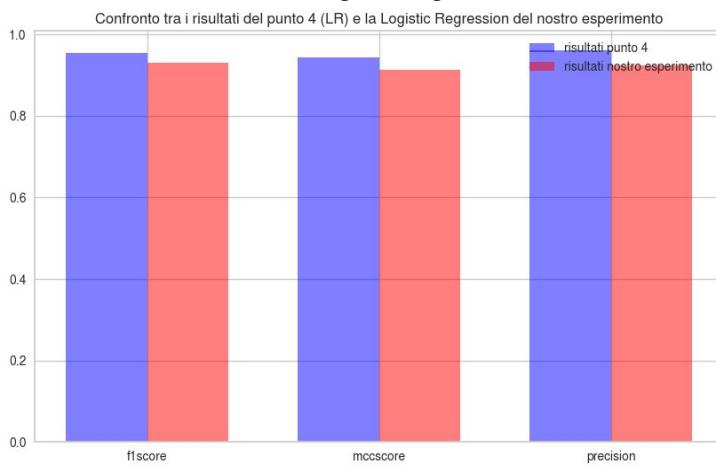
Il che vuol dire che stiamo cadendo anche in questa casistica per entrambi i modelli.

Siccome abbiamo appena confermato:

1. Sia che ci sono feature con alto valore di correlazione (>50%) con il target che non sono state sporcate
2. Sia che ci sono feature che nonostante siano state sporcate la casualità ha voluto riassegnare abbastanza valori corretti (anzichè valori errati) (valori sporchi <20%)

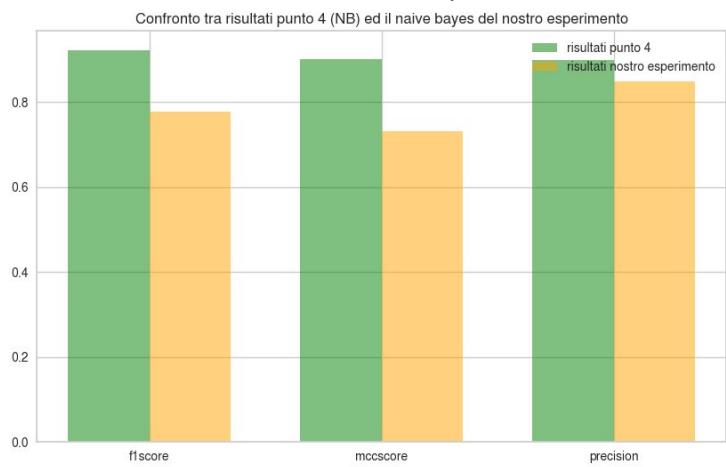
allora possiamo mettere insieme l'elenco di feature ottenuto dal Caso-1 e dal Caso-2 e trainare+testare dei modelli giocattolo (uno per NB ed uno per LR) i quali score di classificazione verranno paragonati con quelli dello step 4 (noi mostriamo quelli relativi alla percentuale di sporcaggio dati uguale a 85%) per andare a verificare che le performance di score ottenute al punto 4 (con percentuale 85%) saranno pressapoco identiche a quelle calcolate dai modelli giocattolo (esperimento). Di seguito vengono riportati i risultati:

**Modello Logistic regression:**



differenze per LR: \$[0.02417778598581044, 0.030514650391437526, 0.037176889543807934]

**Modello Naive Bayes:**



differenze per NB: \$[0.14430078035720117, 0.17080838128398945, 0.0507866889445836]

L'esperimento è dunque riuscito?

- **Per Logistic Regression:**

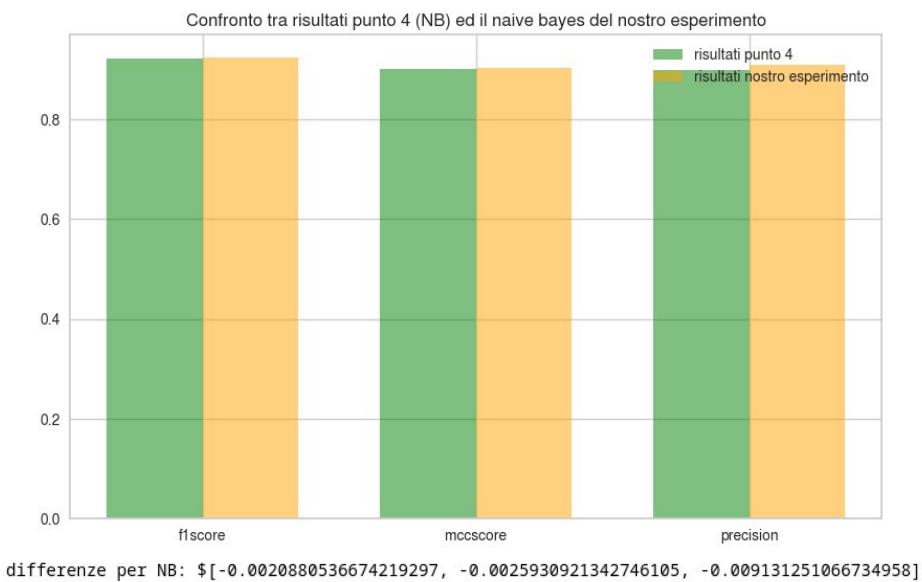
Siamo riusciti a confermare con prove concrete la nostra ipotesi: i modelli trainati solo con:

- feature sporcate solo al di sotto del 20% con alta correlazione con target (>50%)
- feature discriminatorie che non sono state sporcate più del 20% (o meno di 80%)

riportano performance pressapoco identiche a quelli trainati al punto 4 trainati con il dataset di feature sporcate (in seguito ad averlo sporcati con valori casuali). Possiamo affermare con certezza che il nostro modello di Logistic regression riesce comunque ad estrarre buoni risultati di predizione perchè apprende solo sulle feature più pulite.

- **Per Bernoulli Naive Bayes:**

Non siamo troppo certi dei risultati ottenuti, abbiamo una differenza di F1-score e MCC-score consistente intorno al 15% (che è abbastanza grande) tra il nostro esperimento ed i dati sporcati, il che significa che probabilmente il modello di Naive Bayes dipende molto anche dalle altre feature che non abbiamo inserito nel train del nostro esperimento. Di seguito viene infatti dimostrato come semplicemente prendendo tutte le feature con meno del 20% / più del 80% di dati sporchi (ossia invece che fare la selezione sulle feature discriminanti e con alta correlazione si prendono semplicemente tutte) si ottengono feature pressapoco identiche all'esempio del punto 4:



In questo modo si va a confermare, quindi, comunque la nostra teoria sul fatto che anche questo modello sta imparando solo ed esclusivamente su quelle feature che non si sono sporcate troppo.

Queste informazioni ci aiutano inoltre a spiegare in particolare cosa succede nel passaggio da 70% a 85% dello step-2 di LogReg, in cui le performance migliorano. Il modello con dati sporchi al 70% stava classificando basandosi su dati errati. Quando, passando ad 80%, questi sono diventati troppo errati, il modello ha smesso di imparare da quella colonna ed ha iniziato ad ignorarla, migliorando quindi le performance.

## 6.2 SOSTITUZIONE DI VALORI CON VALORI NULLI:

Vogliamo ora inserire valori nulli all'interno dei nostri set. Si vuole far notare che anche qui purtroppo siamo costretti a rompere il vincolo di riproducibilità dei risultati, in quanto la scelta della cella al quale inserire valore nullo è randomica.  
Una volta eseguita questa operazione, non possiamo rieseguire la classificazione in presenza di valori nulli nel nostro dataset, decidiamo quindi di ri-fillarle utilizzando l'imputing con la moda della relativa colonna.

### Nota: L'importanza della scelta dell'imputer

Generalmente un buon metodo per il riempimento di valori vuoti è il metodo 'KNNImputer' (utilizzato per imputare, ovvero stimare, i valori mancanti nei dati utilizzando un algoritmo basato sul vicinato più prossimo (K-nearest neighbor)). Tuttavia, quando tutte le tue feature sono binarie (0/1) come nel nostro caso, la nozione di "distanza" può essere problematica, poiché non è immediatamente chiaro come definire la similarità tra osservazioni basata solo su valori binari. Inoltre, l'imputazione tramite KNN potrebbe non produrre risultati molto significativi quando si trattano feature binarie. Pertanto, per quanto semplice, la scelta del refill con la moda sarà anche la strategia più efficace.

### 6.2.1. VISUALIZZAZIONE DIFFERENZE INTRODOTTE CON INSERIMENTO VALORI CASUALI:

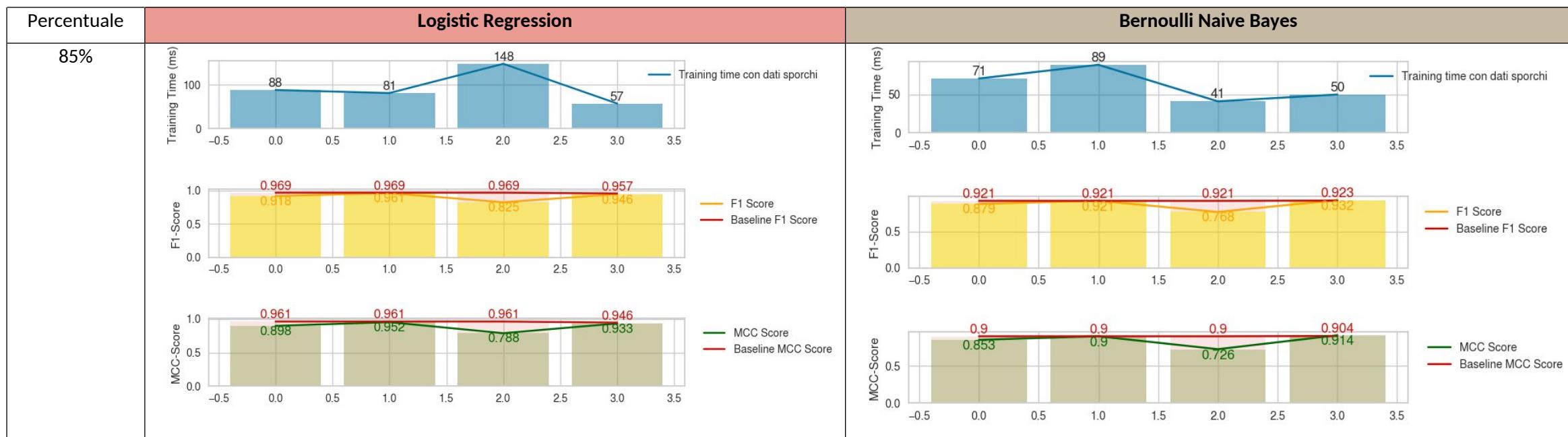
Di seguito vengono visualizzati i soli grafici dell'iterazione con percentuale 85% dell'inserimento valori nulli. Questo perché, come vedremo, già ben con 85% di valori mancanti, l'imputer fa un ottimo lavoro nel refilling dei dati mancanti, pertanto avremo pochissime differenze con il train-set originale. Risulta quindi inutile mostrare tutte le iterazioni precedenti.

	Train-set per Logistic Regression			
	Step 1	Step 2	Step 3	Step 4
Percentuale valori nulli: 85%				
Differenze con trainset originale dopo imputing	Numero di istanze differenti tra il prima e dopo aver sporcati i dati:  Diverse Uguali	Numero di istanze differenti tra il prima e dopo aver sporcati i dati:  Diverse Uguali	Numero di istanze differenti tra il prima e dopo aver sporcati i dati:  Diverse Uguali	Numero di istanze differenti tra il prima e dopo aver sporcati i dati:  Diverse Uguali
PER LOGISTIC REGRESION: in totale abbiamo che inserendo 85.0% di dati nulli si ha che su 241 featuresabbiamo che solo 4 hanno più del 35.0% di dati sporchi	PER LOGISTIC REGRESION: in totale abbiamo che inserendo 85.0% di dati nulli si ha che su 241 featuresabbiamo che solo 2 hanno più del 35.0% di dati sporchi	PER LOGISTIC REGRESION: in totale abbiamo che inserendo 85.0% di dati nulli si ha che su 241 featuresabbiamo che solo 6 hanno più del 35.0% di dati sporchi	PER LOGISTIC REGRESION: in totale abbiamo che inserendo 85.0% di dati nulli si ha che su 30 featuresabbiamo che solo 1 hanno più del 35.0% di dati sporchi	PER LOGISTIC REGRESION: in totale abbiamo che inserendo 85.0% di dati nulli si ha che su 30 featuresabbiamo che solo 1 hanno più del 35.0% di dati sporchi
	Train-set per Bernoulli Naive Bayes:			
	Step 1	Step 2	Step 3	Step 4
Percentuale valori nulli: 85%				
Differenze con trainset originale dopo imputing	Numero di istanze differenti tra il prima e dopo aver sporcati i dati:  Diverse Uguali	Numero di istanze differenti tra il prima e dopo aver sporcati i dati:  Diverse Uguali	Numero di istanze differenti tra il prima e dopo aver sporcati i dati:  Diverse Uguali	Numero di istanze differenti tra il prima e dopo aver sporcati i dati:  Diverse Uguali
PER NAIVE BAYES: in totale abbiamo che inserendo 85.0% di dati nulli si ha che su 241 featuresabbiamo che solo 6 hanno più del 35.0% di dati sporchi	PER NAIVE BAYES: in totale abbiamo che inserendo 85.0% di dati nulli si ha che su 241 featuresabbiamo che solo 0 hanno più del 35.0% di dati sporchi	PER NAIVE BAYES: in totale abbiamo che inserendo 85.0% di dati nulli si ha che su 241 featuresabbiamo che solo 6 hanno più del 35.0% di dati sporchi	PER NAIVE BAYES: in totale abbiamo che inserendo 85.0% di dati nulli si ha che su 30 featuresabbiamo che solo 0 hanno più del 35.0% di dati sporchi	PER NAIVE BAYES: in totale abbiamo che inserendo 85.0% di dati nulli si ha che su 30 featuresabbiamo che solo 0 hanno più del 35.0% di dati sporchi

## 6.2.2. VISUALIZZAZIONE RISULTATI PRODOTTI DAI MODELLI

Guida alla lettura dei risultati:

1. Colonna indicata da 0.0 = Risultati Step 1: Logistic Regression & Naive Bayes - sporcare le feature meno significative (sul dataset intero)
2. Colonna indicata da 1.0 = Risultati Step 2: Logistic Regression & Naive Bayes - sporcare le rispettive feature più discriminanti di ogni modello (sul dataset intero)
3. Colonna indicata da 2.0 = Risultati Step 3: Logistic Regression & Naive Bayes - sporcare tutte le features (sul dataset intero)
4. Colonna indicata da 0.0 = Risultati Step 4: Logistic Regression & Naive Bayes - sporcare le rispettive feature più discriminanti di ogni modello (dataset ristretto)



### 6.2.3. CONSIDERAZIONI E CONCLUSIONI SUL DECADIMENTO DELLE PERFORMANCE TRA BASELINE E MODELLI CON DATI SPORCHI:

Notiamo che generalmente le performance di classificazione non diminuiscono quasi per niente in tutti i punti, eccetto lo step-3. Questo è facilmente spiegabile dal fatto che abbiamo solo pochissime colonne subiscono una variazione molto significativa (più del 35% di dati differenti); e questo succede anche aumentando passo passo la percentuale di dati sporchi arrivando a 85%.

Se andiamo a guardare il grafico di differenze (mostrato al capitolo 6.2.1.) possiamo addirittura notare che allo step-2, per naive bayes, tutte le colonne selezionate come più discriminatorie vengono refillate perfettamente dall'imputer. Da questo possiamo dedurre che:

1. La predizione combacerà sempre con la baseline.
2. Siccome sappiamo che l'imputer rimpiazza i nulli con la moda allora lo strano comportamento che avevamo notato al capitolo 5 per estrarre le feature più discriminanti per naive bayes è giustificato dall'ipotesi che avevamo dato: le colonne selezionate non sono altro che piene di zeri (o quasi).

Ha perfettamente senso invece che (per entrambi i modelli) allo step-3 (e quando succede allo step-1) invece abbiamo performance di classificazione molto diminuite. Ciò avviene perché diversamente dagli altri step questo è l'unico che va a sporcare tutte le colonne. Ergo anche se l'errore inserito è molto piccolo entrambi i modelli sono comunque costretti su dati errati e quindi commettere errori in test.

Ha anche senso che questo comportamento che abbiamo appena spiegato sia meno frequente nello step-1. Infatti in questo step, per quanto venga sempre sporco un gran numero di faeture (191), non tutte le feature vengono sporcate. I modelli quindi in base a quanto i dati vengono sporcati dalla casualità possono comunque "cavarsela" meglio o peggio.

In sostanza quindi mettendo a paragone le performance di classificazione (F1-score e MCC-score) tra baseline e modello addestrato non abbiamo una grossa differenza perché, anche qui, come è visibile da tutti i grafici che mostrano le differenze tra il dataset prima e dopo averlo sporco, pur inserendo un grosso numero di dati nulli, l'imputer basato sulla moda riesce a ricostruire molto bene i dati, abbiamo quindi in verità una percentuale molto bassa di dati sporchi che non ci fa diminuire di molto le performance da baseline a modello con dati sporchi. Anzi andando a paragonare le figure che mostrano il numero di dati sporchi (mostrate al capitolo 6.2.1.) dell'inserimento di dati nulli e l'inserimento di valori casuali (mostrate al capitolo 6.1.1.) diventa palese osservare che il numero di feature sporche diminuisce di molto rispetto all'inserimento di valori casuali; questo è dovuto al fatto che nell'inserimento di valori nulli andiamo poi a ripopolare il dataset con una logica sensata per ricostruire i dati in maniera corretta, invece nell'altro caso, con l'inserimento dei dati casuali, è completamente dato al caso.

Invece il fenomeno dell'avere una variazione così piccola su tutte le colonne che vengono sporcate è spiegabile dal fatto che come abbiamo dimostrato con la prima Data Exploration in molte di queste colonne è presente per la maggior parte solo 0 oppure solo 1. Quindi sostituendo con la moda diventa facile capire come queste feature dopo aver fatto l'imputing non abbiano una variazione del neanche 35%. E, come abbiamo già visto precedentemente con la sostituzione dei dati casuali, se non abbiamo abbastanza colonne sporche i modelli si possono basare semplicemente sulle colonne più "pulite" per fare la predizione, in questo modo si riesce comunque a raggiungere un alto livello di precisione.

Le previsioni ottenute in questo scenario quindi ci piacciono, in quanto perfettamente giustificate dalla stessa spiegazione che abbiamo dato al punto precedente.

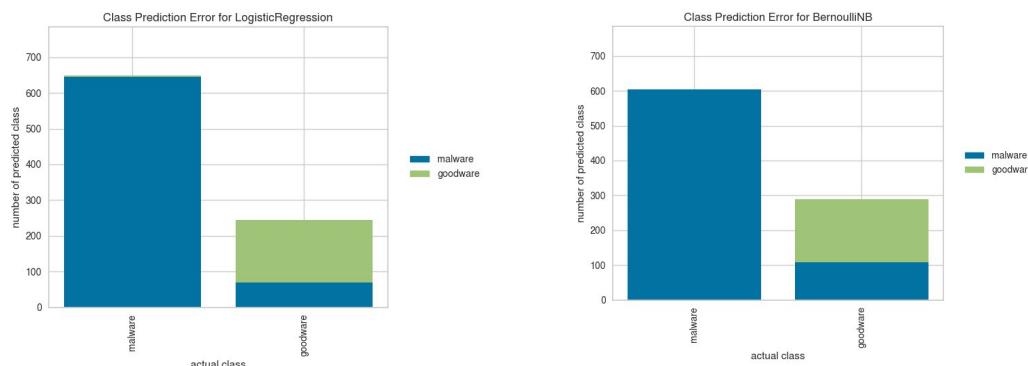
Dato che per quanto riguarda le conclusioni a livello di performance predittive per questo caso possiamo raggiungere le stesse conclusioni al quale siamo arrivati al punto precedente, possiamo invece avere un altro focus per questo punto:

- Considerare le performance temporali di classificazione.
- Considerare la significatività nel contesto.

#### 6.2.3.1. SIGNIFICATIVITÀ NEL DOMINIO APPLICATIVO:

Nello step-3 (sostituzione di valori nulli su tutto il dataset) il rumore diventa leggermente più influente e in certi casi (sporcano all'85%) si arriva anche ad osservare peggioramento delle performance discretamente significativo, più nello specifico:

- Il cambiamento nei risultati di score può essere considerato come sia come significativo che non molto significativo; In quanto (in base alla run dettata dal kernel del notebook) si possono perdere circa tra il 10% ed il 40% di MCC score ed il 5-20% circa di precisione ed f1-score rispetto al baseline model. Come già detto però questi valori cambiano a seconda della run, siccome è impossibile prevedere come il metodo di inserimento di valori nulli si comporterà.
- Allo stesso tempo il cambiamento può essere considerato come molto significativo perché, generalmente, aumenta il numero di istanze di malware erroneamente classificate come goodware.



Questo è un caso sicuramente è un molto più importante ed incisivo sullo studio che stiamo facendo piuttosto che quando invece dei goodware vengono classificati come malware. Se un buon software viene etichettato come malvagio è solo un falso allarme, non succederebbe nulla ad un ipotetico enduser, nel caso opposto, se un malware viene classificato come goodware allora il nostro ipotetico enduser mette la sua macchina a rischio perché potrebbe installare dei programmi malware che non gli sono stati segnalati come tali.

Dunque non sarebbe giusto accettare di avere performance del genere, seppur statisticamente discretamente buone; perchè la maggior parte degli errori di classificazione è su malware che vengono etichettati come goodware.

Nota che non sarebbe possibile fare un analisi di significatività del genere sull'inserimento di valori casuali, in quanto, per definizione, i valori sono inseriti casualmente. Non avremmo quindi modo di stabilire con consistenza se aumentasse la classe di goodware o malware.

### 6.2.3.2. CONSIDERAZIONI PERFORMANCE TEMPORALI:

Prima di procedere dobbiamo però fare una considerazione molto importante: i tempi di computazione possono cambiare molto da macchina a macchina e, alle volte, pur usando la stessa macchina non è inusuale ottenere tempi di elaborazione leggermente diversi in base ai processi attualmente in corso.

Abbiamo dunque che:

- **Per Logistic Regression:**

Ha senso registrare un grosso aumento del tempo di elaborazione ai punti 1 e 3, questo banalmente perché abbiamo un numero molto più alto di dati sporchi, ergo ci sarà bisogno di molte più iterazioni di regressione per andare a selezionare solo i dati che meglio effettuano la classificazione. Di conseguenza ha anche perfettamente senso avere che il tempo di computazione dell'ultimo punto è molto minore rispetto a tutti gli altri; perchè nel punto 4 abbiamo che stiamo usando il dataset ristretto alle feature selezionate in fase di feature extraction, in più stiamo anche andando a sporcare solo una piccola quantità di dati.

- **Per Naive Bayes:**

Differentemente dal precedente questo metodo non richiede ricorsione, ergo i tempi di computazione dipendono principalmente dal numero di istanze e dal numero di feature. Abbiamo quindi che i risultati di tempistiche saranno sicuramente inferiori ai modelli di Logistic Regression (come evidente dai grafici mostrati), in più abbiamo che i tempi di classificazione dell'ultimo punto saranno sempre inferiori rispetto a quelli dei punti precedenti (in quanto contiene un numero di feature inferiore). Se esploriamo la documentazione dell'implementazione di sklear di bernulliNB possiamo anche aggiungere inoltre che i tempi di comunicazione tendono ad aumentare quando abbiamo un numero più elevato di feature contenenti più zeri che uni, andando noi a sostituire con la moda, generalmente, non stiamo altro che aumentare il numero di 0 della colonna.

### 6.3. SOSTITUZIONE DI VALORI CON VALORI OPPosti

Dato che abbiamo individuato le cause dell'alto mantenimento delle performance di classificazione come:

- **Problema 1:** le funzioni scritte fino ad ora nonostante inseriscano sempre la stessa percentuale di valori sporchi molti dati vengono casualmente inseriti correttamente (pertanto ci possono essere colonne con molti pochi dati errati)
- **Problema 2:** Violiamo i vincoli da noi stessi imposti di riproducibilità quando riavviando il kerlen (nonostante si imposti sempre un random state) si aggiunge un ulteriore layer di casualità, che implica che a volte non si registrerà quasi alcun cambiamento con cifre alte di dati sporchi e altre volte si possono registrare risultati di peggioramento catastrofico (anche se raramente).

Quindi, in primo luogo, per evitare di ricadere in questi stessi problemi in questo passaggio cambieremo metodo: utilizzeremo una funzione che inserisce sempre lo stesso valore costante di valori errati (opposti rispetto al valore precedente). In questo modo riusciamo a risolvere il problema 1, ma non risolviamo il problema 2. Puntiamo però sul fatto che avendo sempre lo stesso numero costante di dati sporchi non avremo grosse differenze di classificazione anche se cambiasse il seed del random. Spiegato più con parole povere: se sappiamo che verranno sporcate sempre il 30% costante di una feature non dovremmo avere risultati molto differenti se abbiamo che vengono sporcate le istanze x-y piuttosto che le istanze a-b (in quanto avremo sempre comunque il 30% assicurato di dati sporchi).

In secondo luogo andremo a valutare poi le performance prendendo:

- Come dataset solo le feature più significative (dataset ristretto composto solo da 30 feature).
- Come colonne da sporcare inizialmente si volevano sporcare solo le feature più discriminanti per ciascun modello, ma poi ci si rende conto della necessità di sporcarle tutte (approfondiremo questa parentesi alla visualizzazione dei risultati).

Inizialmente queste feature sarebbero dovute venire sporcate una ad una ricorsivamente per tutti i valori da 5 a 100% (con del codice che aumenta ogni volta il grado di "sporcaggio" del 5%), ma ci si accorge che già passando dallo 0 al 5% c'è un grande cambiamento lista di feature più discriminanti:

		Feature	Coefficient
		Feature con rango 1 vine sporcata del 5%	
6	KILL_BACKGROUND_PROCESSES	-4.001096	
13	VIBRATE	3.876031	
12	USE_FINGERPRINT	3.244828	
11	SYSTEM_ALERT_WINDOW	-3.235401	
9	RECEIVE_SMS	-3.161649	
8	RECEIVE_BOOT_COMPLETED	-2.806361	
18	Ljava/net/URL;.>openConnection	2.705800	
25	Landroid/telephony/TelephonyManager;.>getNetwo...	-2.637748	
16	Ljava/lang/System;.>load	2.597927	
26	Landroid/telephony/TelephonyManager;.>getNetwo...	2.588948	
3	CHANGE_NETWORK_STATE	2.197176	
28	Landroid/telephony/TelephonyManager;.>getSimCo...	-2.018569	
21	Landroid/location/LocationManager;.>getLastKgo...	1.672308	
22	Landroid/content/pm/PackageManager;.>getInstal...	1.631474	
7	READ_PHONE_STATE	-1.558184	
27	Landroid/telephony/TelephonyManager;.>getSimOp...	1.450294	
19	Landroid/media/AudioRecord;.>startRecording	-1.439753	
23	Landroid/content/pm/PackageManager;.>getInstal...	1.409984	
17	Ldalvik/system/DexClassLeader;.>loadClass	1.295202	
14	WAKE_LOCK	-1.204094	
2	BLUETOOTH	1.191296	
29	Lorg/apache/http/impl/client/DefaultHttpClient...	-1.075982	
20	Landroid/telephony/TelephonyManager;.>getCell...	-1.046811	
0	ACCESS_COARSE_LOCATION	0.958635	
5	GET_TASKS	-0.905319	
4	DISABLE_KEYGUARD	0.714970	
15	WRITE_EXTERNAL_STORAGE	0.647902	
24	Landroid/telephony/TelephonyManager;.>getNetwo...	-0.554592	
10	SEND_SMS	-0.475986	
1	ACCESS_FINE_LOCATION	0.078024	

Passa dalla prima ad essere la 21esima.

Quindi si decide di fare tutte le iterazioni da 0% a 20% aumentando di solo 1% alla volta. Andremo poi a misurare quanto le qualità discriminatorie della feature sono cambiate per ogni iterazione dall'1 al 20%.

#### 6.3.1. VISUALIZZAZIONE RISULTATI PRODOTTI DAI MODELLI:

Abbiamo che, come al solito, prima di applicare qualsiasi trasformazione si divide il dataset in train e test e poi si esegue l'inserimento di valori opposti sulla copia del train-set con una funzione rappresentata da questo pseudo-codice:

```
Foreach(Feature_i in Feature_Dataset_ristretto)
{
    for(int x = 0; x<21; x++)
    {
        Si prende una nuova copia del Dataset_ristretto e si sporca la Feature_i per x%;

        Si ricalcola l'indice di significatività di tutte le feature all'interno del
        dataset (in modo da salvarci ottenere l'informazione del nuovo rank della feature);

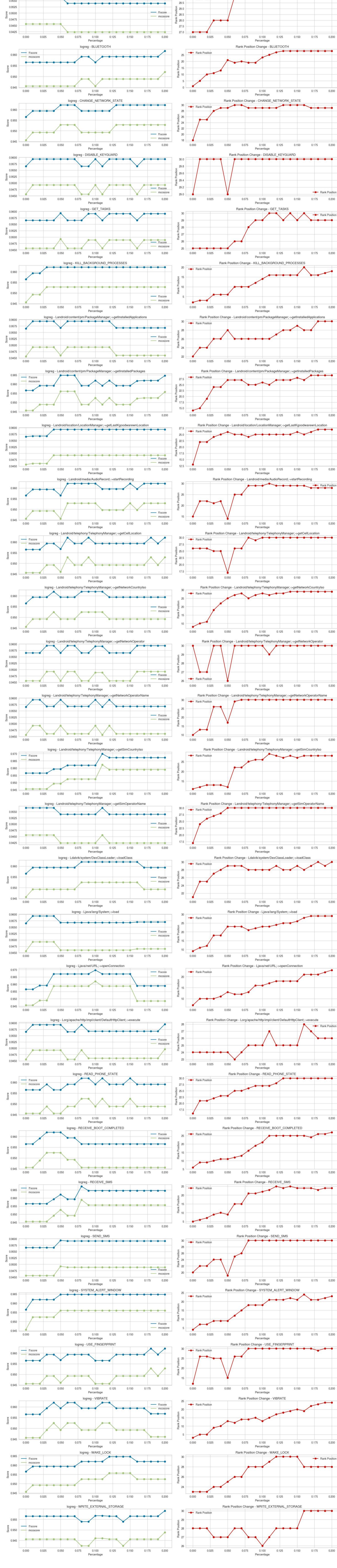
        Si fa train+test di entrambi i modelli;

        Si salvano tutti i risultati in una lista di oggetti;
    }
}
```

Alle pagine seguenti vengono mostrati i risultati ottenuti dall'inserimento di valori opposti su ogni feature (singolarmente), ossia i risultati prodotti dall'esecuzione dello pseudocodice appena mostrato. Abbiamo che:

- Il grafico a sinistra indica la variazione di F1-score ed MCC-score dei modelli trainati sul dataset ad ogni aggiunta di 1% valore sporco
- Il grafico a destra indica come varia la posizione (rank) sull'insieme delle 30 feature della Feature\_i con ogni aggiunta di 1%. Ovviamente più il rank sale più vuol dire che la feature sta perdendo di significatività (in quanto su 30 feature non ricopre più la posizione, per esempio 1, ma ricopre una posizione più bassa).

### logreg Performance Metrics



### naive\_bayes Performance Metrics



### 6.3.2. CONSIDERAZIONI E CONCLUSIONI SUL DECADIMENTO DELLE PERFORMANCE TRA BASELINE E MODELLI CON DATI SPORCHI:

Dai 2 plot di Logistic regression e Bernoulli Naive Bayes appena mostrati riusciamo a ricavare una serie di informazioni estremamente importanti:

1. I dati appena mostrati sono sempre consistenti indipendentemente dalla funzione di random e dal kerlen. Il che significa che le assunzioni che abbiamo fatto all'introduzione di questo sotto-capitolo per arrivare, finalmente, a rispettare il vincolo (da noi imposto) di riproducibilità dei risultati sono corrette.
2. In entrambi i modelli, modificando una feature alla volta, DATA UNA QUALSIASI FEATURE, si può notare come, nella maggior parte dei casi, la qualità discriminante della feature in questione diminuisca significativamente dal grafico della rank position. Infatti generalmente tutte le feature che registrano un aumento di rank tendono a scendere di 10-15-20 posizioni. Ossia da essere feature molto significative passano al fondo della classifica di significatività. Oppure è anche possibile, in casi meno frequenti, che:
  - a. Le feature mantengano lo stesso indice di importanza per tutte le iterazioni
  - b. Le feature aumentino proprio indice di importanza, ossia diventano più importanti, più significative. Questo, però, oltre ad avvenire in pochi casi, quando succede si può notare che generalmente la diminuzione di rank dalla baseline (dove la baseline è rappresentata dal dato sporco allo 0%) non è sostanziale. Generalmente la feature riesce a guadagnare una o due posizioni. Mettendo però a confronto con la situazione di quando le feature diminuiscono la loro significatività ed aumentano il loro rank di 10-15-20 posizioni (maggioranza dei casi), una diminuzione di 1 o 2 posizioni che abbiamo notato in questi casi diventa non molto significativa.

In seguito, si vuole far ora notare che in tutti i casi in cui sporchiamo le feature (per entrambi i modelli), anche nei casi in cui abbiamo che la significatività della feature è diminuita tanto da scalare di 10-15-20 posizioni, le performance del modello non cambiano per nulla in presenza del dato sporco (il cambiamento massimo è stato del 6%). Per questo motivo nella fase introduttiva alla visualizzazione dei risultati si specifica che questa operazione di "sporcaggio" su tutte le feature viene fatta, appunto, su tutte le feature e non solo sulla top 10 delle più discriminanti. Se si avessero usato solo le prime 10 più discriminanti avremmo avuto gli stessi risultati, ma ci sarebbe potuto venire il dubbio allora che la funzione per ottenere l'elenco ordinato delle feature, dalla più discriminante alla meno discriminante, avesse qualche problema o ritornasse risultati errati. Ma avendo che le performance non diminuiscono per tutte le feature indipendentemente da quanto queste cadano di significatività significa che entrambi i modelli riescono fare corretta classificazione perché probabilmente basano la loro classificazione su "gruppi" di feature, invece che su una sola feature. Quando una sola di queste feature viene sporcata anche solo dell'1% è abbastanza per farla cadere al di fuori del "gruppo di feature importanti per la classificazione" e il modello imparerà dalle altre senza quindi andare mai a perdere alcuno score di classificazione, in maniera simile a come abbiamo osservato nei casi 6.1 e 6.2.

Per procedere, se vogliamo arrivare al risultato finale di ottenere degli score di classificazione peggiori dovremmo quindi individuare quali sono questi gruppi e andarli a sporcare tutte le feature ad essi appartenti contemporaneamente, gruppo per gruppo. Solo che questa lavorazione potrebbe richiedere molto tempo (specialmente se si vuole usare un approccio naive a forza bruta in cui si va a provare ogni feature e ricorsivamente andandola a mettere in gruppo con tutte le possibili combinazioni delle altre feature, di fatto facendo l'insieme delle parti di tutte le combinazioni di feature possibili).

Quindi per risparmiare tempo, utilizziamo un approccio ancora più naïve, lo stesso che ai punti 6.1. e 6.2. ci ha dato i peggiori score di classificazione: consideriamo un solo gruppo contente tutte le feature, ossia le sporcheremo tutte contemporaneamente.

In questo caso, dato che abbiamo un livello di complessità inferiore, possiamo permetterci di sporcare tutti i dati dallo 0% al 100%, addestrando un nuovo modello per ogni iterazione, dove abbiamo che in ogni iterazione sporcheremo quindi tutte le feature di un 1% in più rispetto all'iterazione precedente. Ossia si svolge ciò che viene rappresentato da questo pseudo-codice:

```
for(int x = 0; x<101; x++)
{
    Si prende una nuova copia del Dataset_ristretto e si sporcano tutte le feature per x%;

    Si ricalcola l'indice di significatività di tutte le feature all'interno del
    dataset (in modo da salvarci ottenere l'informazione del nuovo rank della feature);

    Si fa train+test di entrambi i modelli;

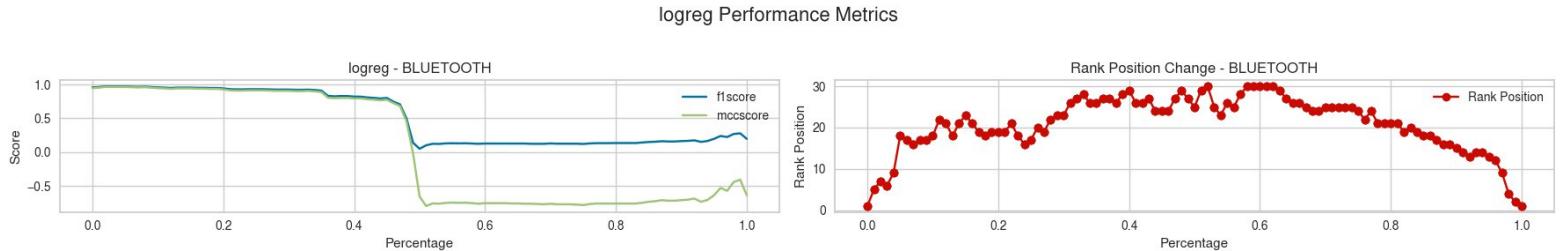
    Si salvano tutti i risultati in una lista di oggetti;
}
```

Avremo dunque che:

- Il grafico a sinistra indica la variazione di F1-score ed MCC-score dei modelli trainati sul dataset ad ogni aggiunta di 1% valore sporco
- Il grafico a destra indica come varia la posizione (rank) sull'insieme delle 30 feature della feature più importante (quella che nella baseline ha rank 1) con ogni aggiunta di 1% .

Di seguito vengono mostrati i risultati di questa elaborazione:

### 6.3.2.1. LOGISTIC REGRESSION: CONSIDERAZIONI SUL CAMBIAMENTO DELLE PERFORMANCE:



In conclusione, l'analisi dei grafici mostra chiaramente l'impatto del rumore sui modelli di regressione logistica, evidenziando una soglia critica (intorno al 50%) oltre la quale le prestazioni crollano drasticamente. La variabilità nella posizione di rank della feature più importante indica una dinamica complessa tra le feature man mano che aumenta il rumore, suggerendo un tentativo del modello di adattarsi alle condizioni mutevoli dei dati. Vediamo questi aspetti più nel dettaglio:

#### GRAFICO A SINISTRA: ANDAMENTO DI F1-SCORE E MCC-SCORE:

- **Andamento dell'F1-score:**

L'F1-score rimane relativamente stabile e vicino a 1, anche se registra una leggera tendenza negativa, fino a circa il 50% di dati sporchi. Questo conferma ulteriormente ciò che abbiamo avuto modo di appurare con gli esperimenti precedenti: che il modello è robusto fino a un certo livello di rumore nei dati, mantenendo alte soglie di F1-score e MCC-score.

Dopo il 50%, l'F1-score inizia a diminuire drasticamente, indicando che il modello inizia a perdere la capacità di identificare correttamente i campioni positivi. Ciò deve essere dovuto all'aumento significativo dei falsi positivi e falsi negativi, che portano ad avere il rapporto di F1-score pari a 0 nei modelli che contengono un quantitativo maggiore di 50% di dati opposti, fino al 100%. Avere F1-score pari a 0 ci indica che il modello non è riuscito a identificare correttamente nessun campione positivo (affermazione che viene confermata anche dallo score seguente).

- **Andamento del MCC-score:**

L'MCC-score, come lo score precedente, rimane positivo fino a circa il 50% di dati sporchi, indicando una buona capacità di classificazione anche in presenza di rumore.

Dopo il 50%, però, l'MCC-score crolla drasticamente, diventando negativo. Questo suggerisce che il modello non solo perde la capacità di classificare correttamente, ma inizia a farlo peggio di una classificazione casuale. Un MCC negativo indica che le previsioni del modello sono in gran parte sbagliate, nel nostro caso però, essendo anche il target una feature binaria significa che stiamo predicendo il risultato opposto. Di fatto, in questo modo, abbiamo confermato che introducendo dati opposti per una soglia strettamente maggiore al 50% dei valori originali abbiamo ottenuto un classificatore inverso. E, di conseguenza, probabilmente, in un punto di intermezzo tra il 45% e il 50% avremo un classificatore casuale.

#### **GRAFICO A DESTRA: CAMBIAMENTO DI POSIZIONE DI RANGO DELLA FEATURE PIÙ IMPORTANTE:**

Il grafico che indica il cambiamento di posizione di rango della feature più importante non fa' altro che confermare ulteriormente ciò che abbiamo dedotto dagli score, infatti abbiamo 3 situazioni:

##### **1. Lento aumento:**

La posizione di rango della feature più importante parte da 1 ed ha una lenta salita fino alla posizione 30 (ultima posizione) nel momento in cui sporchiamo il 40-50% di dati.

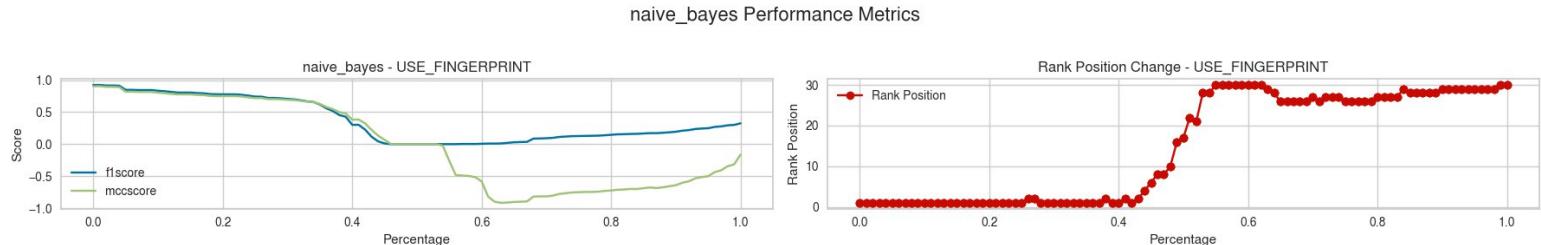
##### **2. Stabilità:**

Arrivati in vicinanza del 50%, il rango della feature più importante è molto vicina all'ultima. Ciò ci indica che abbiamo introdotto una quantità tale di rumore che tutte le feature hanno perso la loro significatività. Quindi, a questo punto, il modello è completamente "confuso" su quale feature utilizzare per la classificazione. Come indicato dagli score precedenti, infatti, sporcando al 50% abbiamo un classificatore casuale.

##### **3. Lenta Diminuzione:**

In seguito al 50%, si nota una lenta ascesa della feature più significativa che lentamente ri-scala la classifica e torna ad avere la prima posizione a livello di significatività. Ciò avviene in quanto avendo feature binarie invertendo il 100% dei valori della feature si finisce per avere lo stesso valore informativo di partenza. Ergo abbiamo che la feature si riavvicina ad avere significatività man mano che ci allontaniamo dal 50% e avviciniamo al 100%. Si finisce però ad avere un classificatore opposto in quanto trainato su dati appunto opposti (sporchi per il 100%) e testati su un test\_set pulito.

### 6.3.2.2. NAIVE BAYES: CONSIDERAZIONI SUL CAMBIAMENTO DELLE PERFORMANCE:



In conclusione, possiamo trarre molte similitudini tra i due algoritmo di Naive Bayes e Logistic Regression, in quanto l'analisi dei grafici mostra, anche qui, l'impatto alto del rumore sui modelli di Naive bayes, evidenziando una soglia critica (intorno al 50%) oltre la quale le prestazioni crollano drasticamente, arrivando ad avere un classificatore inverso.

Abbiamo però una peculiarità nella variabilità della posizione di rank della feature più importante. Questo dato in questo caso indica una dinamica diversa rispetto quella vista precedentemente. Vediamo questo aspetto più nel dettaglio.

#### CONSIDERAZIONI SU ENTRAMBI I GRAFICI:

Sia l'andamento dell'F1-score che del MCC-score sono molto simili a quelli analizzati nei modelli di regressione logistica precedente. Cioè abbiamo che fino a circa il 45-50% di dati sporchi le performance di classificazione rimangono sempre alte, è però notevole osservare come con modello di Naive Bayes queste performance tendano a diminuire molto di più velocemente rispetto ai modelli di regressione logistica. Ciò ci da conferma che, come aveva indicato fin dall'inizio, i modelli di Naive Bayes sono dunque molto più suscettibili alla presenza di rumore.

Aumentando poi il rumore ad una soglia vicina a 50% si arriva, come nell'algoritmo precedente, ad avere un classificatore casuale. Infine nella strada tra il 50% ed il 100% si va a tendere verso il classificatore opposto (per gli stessi motivi indicati nel capitoletto delle conclusioni di Logistic Regression).

Una cosa però molto interessante da notare è che, differentemente dal classificatore precedente, arrivando più vicini verso il 100% dei dati sporchi il classificatore perde nuovamente affidabilità passando da un classificatore opposto ad uno più simile ad un classificatore casuale. Questo comportamento particolare si rispecchia anche nella significatività della feature più importante (nel grafico a destra) che, differentemente dal caso precedente, una volta raggiunto il 50% non torna ad essere più significativa sulla strada dal 50% al 100%. Questo comportamento potrebbe essere spiegato dal fatto che:

1. l'algoritmo di Bernulli Naive Bayes si basa sull'assunzione di indipendenza tra le feature: Quando i dati sono sporcati, questa assunzione viene violata, il che porta a una diminuzione delle prestazioni del modello ma anche alla perdita di significatività delle feature che precedentemente erano considerate come "più discriminanti".
2. l'algoritmo di Bernulli Naive Bayes si comporta bene quando molte feature all'interno del dataset assumono valore 0 (che è il nostro caso prima di sporcare i dati), infatti la nostra funzione di ranking delle feature basandosi sul logaritmo tende a mettere a dare un score di importanza maggiore (ossia posizioni più basse, come posizione 1,2,...) a quelle feature che contengono un alto numero di 0. Inserendo però valori inversi per il 100% dei dati arriviamo ad avere un dataset con feature prevalentemente composte da 1. Ergo la nostra funzione per il ranking delle feature

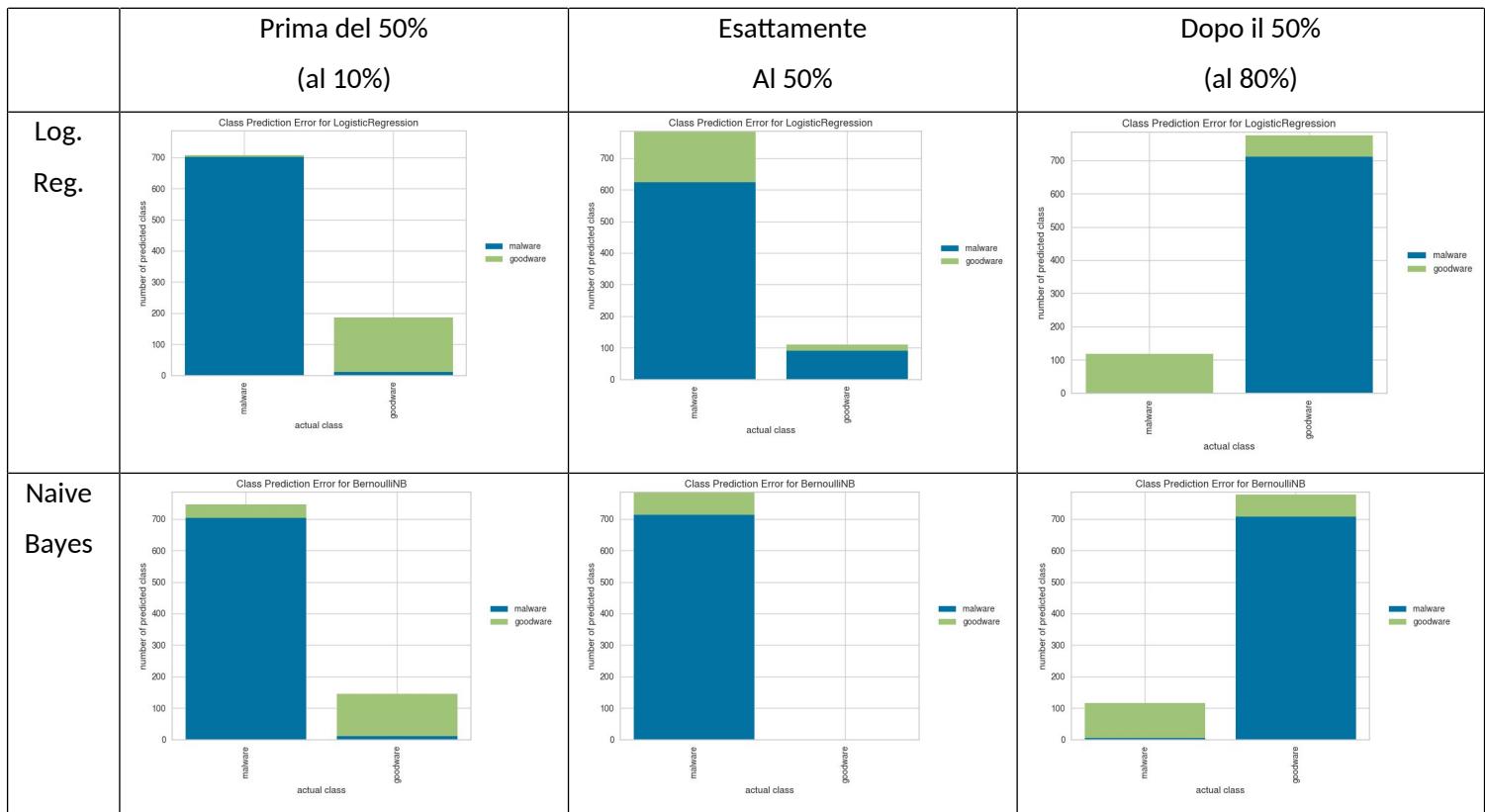
farà passare alle prime posizioni quelle feature che prima avevano il maggior numero di uni ed il classificatore farà molta più fatica ad ottenere una classificazione corretta. Ha quindi perfettamente senso che invertendo tutti i valori, quella che nella baseline era la feature più importante, magicamente scali all'ultima posizione, questa infatti era prima quella che conteneva più zeri di tutti ed ora sarà quindi quella che contiene più uni di tutti.

### 6.3.2.3. CONSIDERAZIONI SULLA SIGNIFICATIVITÀ NEL DOMINIO APPLICATIVO:

Abbiamo dunque detto che, sporcando tutte le feature, ambi i classificatori evidenziando una soglia critica (intorno al 50%) oltre la quale le prestazioni crollano drasticamente:

- Prima di questa soglia abbiamo una classificazione “normale”, dove più aumenta il rumore più aumentano gli errori di classificazione.
- Esattamente sul soglia abbiamo una classificazione casuale
- Dalla soglia in poi arriviamo ad avere un classificatore inverso.

Dedichiamo questa sezione ad esplorare queste casistiche dal punto di vista del dominio applicativo. Come vengono classificati i dati?



Si nota che in tutte casistiche, generalmente, inserendo i valori opposti si tende ad avere una missclassificazione sui valori di goodware che vengono erroneamente etichettati come malware. Questo tipo di errore è molto meno influente, perché, come abbiamo già analizzato in precedenza, equivale a dare un semplice falso allarme all'utente. Di fatto comunque la macchina dell'utente rimane sempre al sicuro. Quindi, ironicamente, questi risultati (o almeno quelli dallo 0% al 50%) a livello di significatività nel dominio sono molto più accettabili che quelli visti sul inserimento di valori nulli.



## 7. RECAP CONCLUSIVO

### 7.1. RECAP SUGLI OBIETTIVI DI CLASSIFICAZIONE:

Se facciamo dunque un recap sui nostri obiettivi di classificazione abbiamo che tendenzialmente riusciamo a raggiungere tutti i nostri obiettivi, infatti abbiamo che:

#### 1. La base:

Utilizzando i modelli Logistic Regression e Bernoulli Naive Bayes siamo riusciti a predire correttamente se un applicazione è un malware o goodware in base ai permessi e le api call fornite dal dataset con soglie di F1-score e MCC-score molto alte.

#### 2. Feature selection:

Per la fase di feature selection riusciamo a diminuire drasticamente la numerosità di feature del dataset passando da 241 a 30 feature ed inoltre abbiamo sviluppato anche delle funzioni accurate per individuare sia per Logistic Regression sia per Naive Bayes l'elenco di feature che più discriminano.

#### 3. Sporcare le feature:

- Per inserimento valori casuali:

Siamo giunti alla conclusione che non riusciamo a rispettare il vincolo da noi imposto di riproducibilità dati ma inserendo valori casuali, ad alcuni valori, casualmente, viene riassegnato il valore corretto. Ergo entrambi i modelli possono poi imparare sulle colonne più pulite ed avere comunque una classificazione corretta. Inoltre si è mostrato quanto i risultati possano variare in quanto questo punto non rispetta il vincolo di riproducibilità dei dati da noi imposto. Per confermare ciò abbiamo dimostrato con un esperimento che all'interno del dataset di train per entrambi i modelli c'erano sia feature con alta correlazione con il target sia feature altamente discriminatorie che non erano state sporcate in maniera sufficiente.

- Per inserimento valori nulli:

Siamo giunti alle stesse conclusioni dello scenario precedente sia per quanto riguarda il vincolo di riproducibilità dei dati (siccome anche qui usiamo il random per selezionare le celle nel quale inserire dati nulli non abbiamo ancora piena riproducibilità) che per quanto riguarda la previsione dei dati. Ossia diventa facile osservare in come i modelli possano facilmente imparare dai dati che non vengono sporcati. Inoltre in questa fase usiamo l'imputer per la moda per ricostruire i dati che fa' un ottimo lavoro, porta ad un numero molto basso la soglia di dati differenti tra prima di aver inserito valori nulli e a dopo aver fatto l'imputing. In seguito a queste conclusioni ci siamo focalizzati quindi a fare delle considerazioni sui tempi di esecuzione e train dei modelli e notiamo che (come ci si poteva aspettare) Logistic regression per quanto più accurato impiega molto più tempo a trainare e predire. Questo comportamento è dovuto alla ricorsione applicata dal modello, cosa che Naive Bayes non deve fare (impiegando tempi minori).

- Per inserimento valori casuali:

Sviluppiamo appositamente questa metodologia di inserimento di rumore nel dataset per provare le nostre ipotesi per implementare la riproducibilità dei dati. Le nostre ipotesi vengono dunque confermate ma addestrando i modelli e sporcando una sola feature alla volta si ricade ancora nell'intoppo di avere valori di score molto alti, in quanto i modelli possono imparare da tutte le altre feature. Per questo motivo si decide quindi di sporcare tutte le colonne del dataset contemporaneamente creando un 2 nuovi modelli (e registrandone gli score) per ogni 1% da 0% a 100%. Analizzando le performance ottenute in questo modo risulta che entrambi i modelli evidenziando una soglia critica (intorno al 50% di dati opposti) oltre la quale le prestazioni crollano drasticamente, si ha quindi che:

- Prima di questa soglia abbiamo una classificazione "normale", dove più aumenta il rumore più aumentano gli errori di classificazione.
- Esattamente sul soglia abbiamo una classificazione casuale
- Dalla soglia in poi arriviamo ad avere un classificatore inverso.

## 7.2. RECAP SULLA SIGNIFICATIVITÀ DELLE PREDIZIONI SUL DOMINIO:

Negli step di inserimento di rumore all'interno del dataset abbiamo anche dato delle spiegazioni sul significato che le predizioni ottenute hanno sul dominio, di seguito se ne riporta il succo:

- **Con l'inserimento di valori casuali:**

Abbiamo detto che non sarebbe possibile fare un analisi di significatività del genere, in quanto, per definizione, i valori sono inseriti casualmente. Non avremmo mai modo di stabilire con consistenza se venissero inseriti con certezza falsi positivi o falsi negativi che aumentano la classe di goodware o malware

- **Con l'inserimento di valori nulli:**

Si registra generalmente un aumento il numero di istanze di malware erroneamente classificate come goodware. Questo è un caso sicuramente molto più importante ed incisivo sullo studio che stiamo facendo piuttosto che quando invece dei goodware vengono classificati come malware. Se un buon software viene etichettato come malvagio è solo un falso allarme, non succederebbe nulla ad un ipotetico enduser, nel caso opposto, se un malware viene classificato come goodware allora il nostro ipotetico enduser mette la sua macchina a rischio perché potrebbe installare dei programmi malware che non gli sono stati segnalati come tali. Dunque non sarebbe giusto accettare di avere performance del genere, seppur statisticamente discretamente buone.

- **Con l'inserimento dei valori opposti:**

Si registra il caso opposto al precedente, generalmente, inserendo i valori opposti si tende ad avere una missclassificazione sui valori di goodware che vengono erroneamente etichettati come malware. Questo tipo di errore è molto meno influente, perchè, come abbiamo già analizzato in precedenza, equivale a dare un semplice falso allarme all'utente. Di fatto comunque la macchina dell'utente rimane sempre al sicuro. Quindi, ironicamente, questi risultati a livello di significatività nel dominio sono molto più accettabili che quelli visti sul inserimento di valori nulli.