Versione 1

21/06/2024

Relazione di Progetto:

ARCHITETTURA DATI

GoodWare VS Malware:

Conseguenze dello sporcare un DataSet sui modelli predittivi

Presented by: cavallini Francesco

Matricola: 920835

UNIVERSITÀ MILANO BICOCCA

# 0. Sommario:

[0. Sommario: 3](#_Toc1)

[1. Introduzione: goodware vs malaware 4](#_Toc2)

[1.1. Premessa: Scelta del tema: 4](#_Toc3)

[1.2. Il Dataset: 4](#_Toc4)

[1.3. Obbiettivi del progetto: 5](#_Toc5)

[1.4. Fasi del Progetto: 6](#_Toc6)

[2. Caricare il dataset + Data Exploration 8](#_Toc7)

[2.1. Caricamento del dataset: 8](#_Toc8)

[2.2. Data exploation: 8](#_Toc9)

[2.2.1. Verifica dataset sbilanciato: 8](#_Toc10)

[2.2.2. Verifica presenza valori nulli: 9](#_Toc11)

[2.2.3. Verifica sparsità dei dati: 9](#_Toc12)

[2.2.4. Verifica correlazione con il target: 10](#_Toc13)

[3. Scelta dei modelli: Baseline e Modelli di validazione: 12](#_Toc14)

[3.1. Scelta modelli ed implementazione di funzioni di train: 12](#_Toc15)

[3.2. Modelli di baseline 14](#_Toc16)

[4. Preparazione dati: Feature selection 18](#_Toc17)

[4.1 Analisi bivariata della correlazione - Eliminazione feature riNdondanti: 18](#_Toc18)

[4.2. Analisi bivariata di correlazione - Highlight feature più correlate al target 22](#_Toc19)

# 1. Introduzione: goodware vs malaware

## 1.1. Premessa: Scelta del tema:

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

## 1.2. Il Dataset:

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

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

## 1.3. Obbiettivi del progetto:

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

1. **La base:**

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

1. **Feature selection:**

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

1. **Sporcare le feature:**

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

1. **Riproducibilità dei dati:**

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

## 1.4. Fasi del Progetto:

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

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

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

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

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

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

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

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

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

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

1. **Sporcare il dataset:**

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

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

# 2. Caricare il dataset + Data Exploration

## 2.1. Caricamento del dataset:

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

## 2.2. Data exploation:

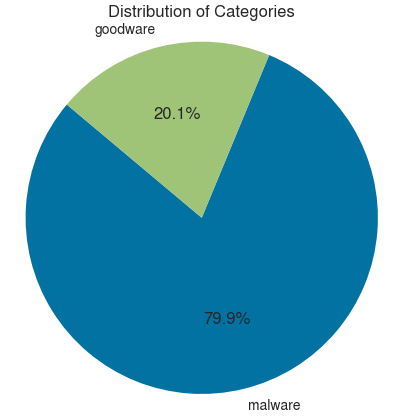
In questa prima fase di esplorazione del dataset andremo a scoprire quali sono le qualità intrinseche del dataset per formulare un idea di quali modelli utilizzare e cosa aspettarci dalla utilizzo di sopracitati modelli. Ovviamente dovremmo fare utilizzo di più passaggi per verificare più qualità del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

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

Questi passaggi vengono approfonditi nelle sezioni sottostanti:

### 2.2.1. Verifica dataset sbilanciato:

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



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

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

### 2.2.2. Verifica presenza valori nulli:

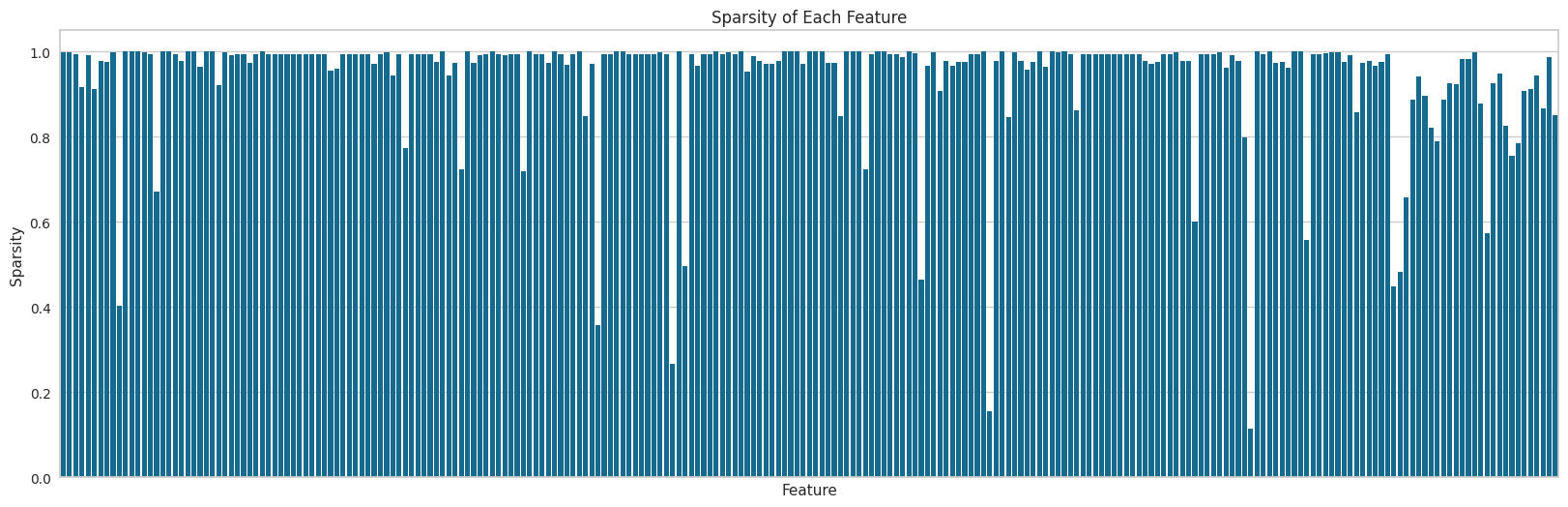
Controlliamo ora se il dataset contiene valori mancanti. Dopo un primo controllo sembra che tutte le features contengano almeno un valore nullo, Ma in seguito ad un analisi più approfondita (tramite il richiamo di una funzione che ci andrà ad elencare tutte le instanze all'interno di ogni features con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

Decidiamo dunque eliminare la riga dal dataset, in quanto

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

### 2.2.3. Verifica sparsità dei dati:

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



Considerando che:

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

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

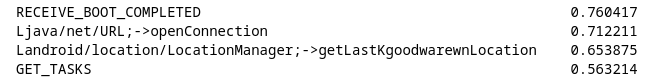
### 2.2.4. Verifica correlazione con il target:

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

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

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

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



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

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

## 3.1. Scelta modelli ed implementazione di funzioni di train:

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

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

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

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

* **Logistic Regression**:

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

* **Bernoulli Naive bayes**:

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

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

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

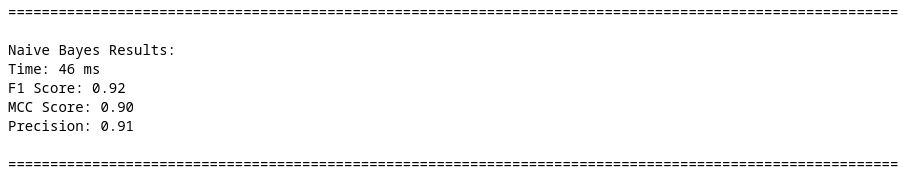
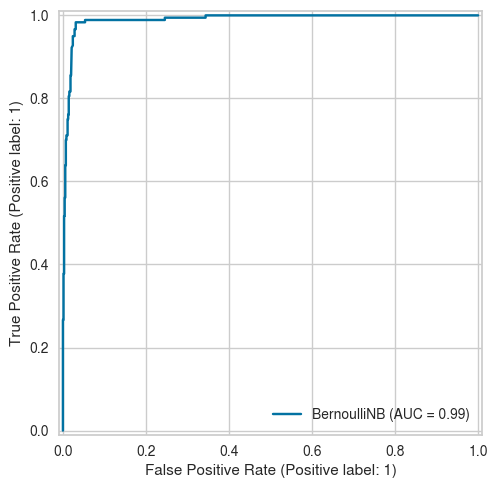
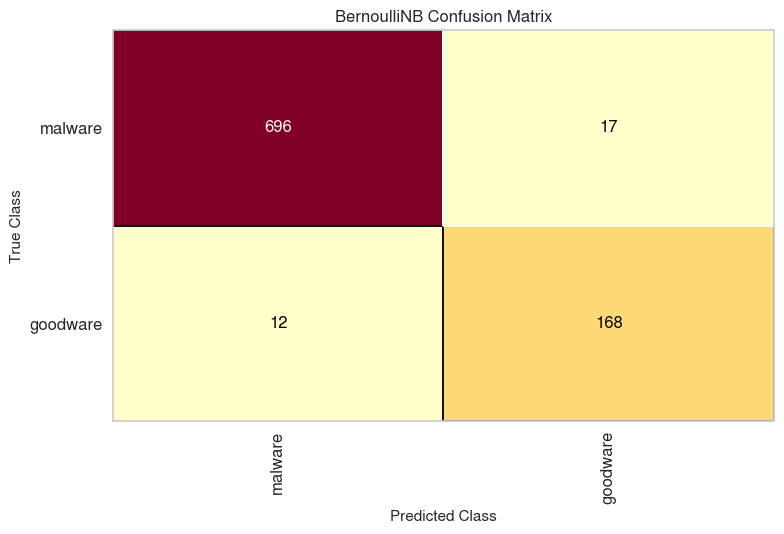
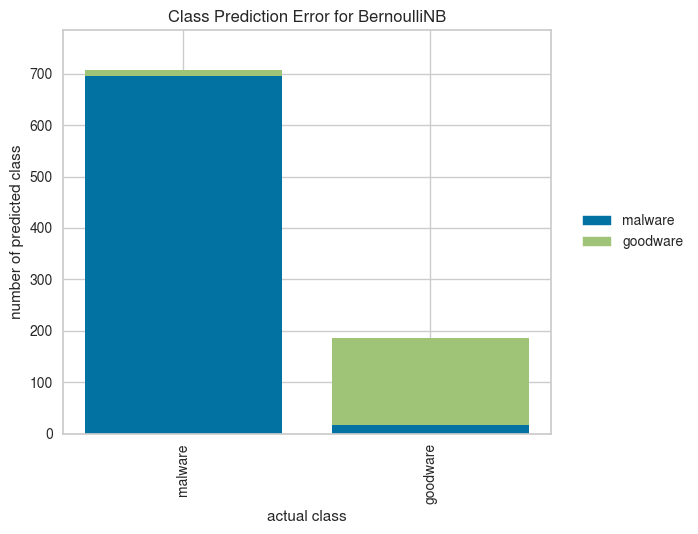
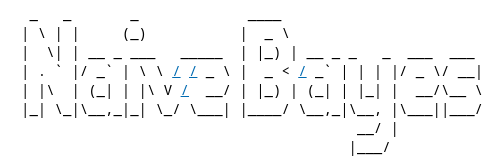
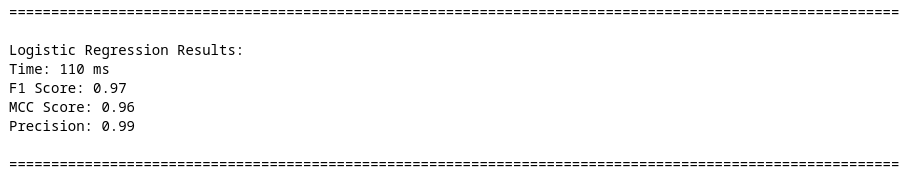
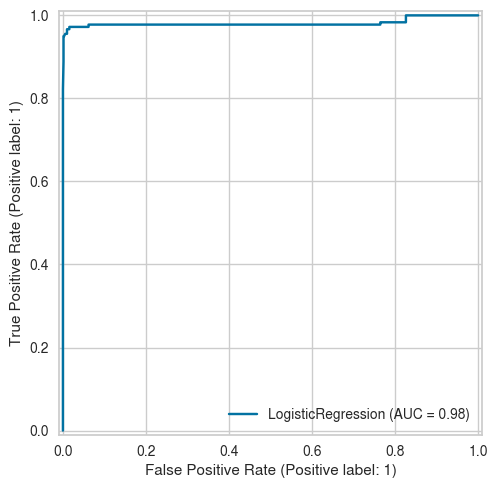
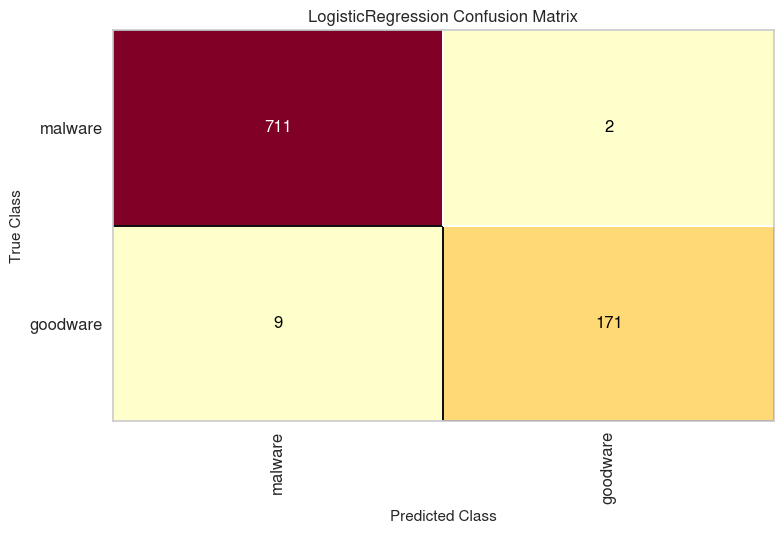
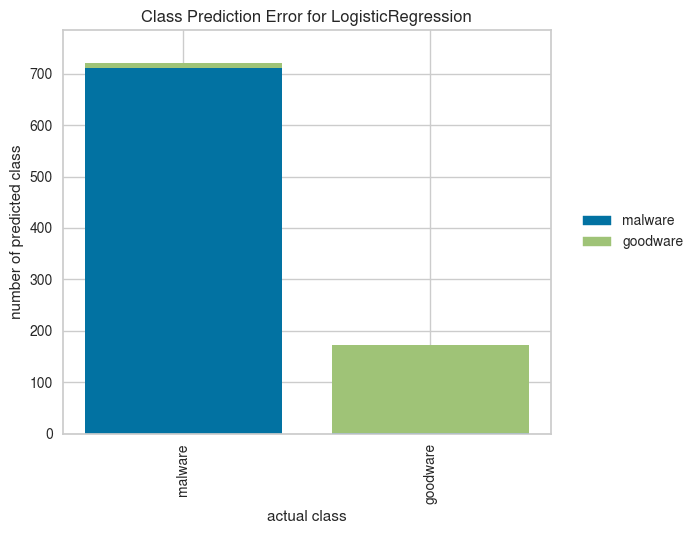
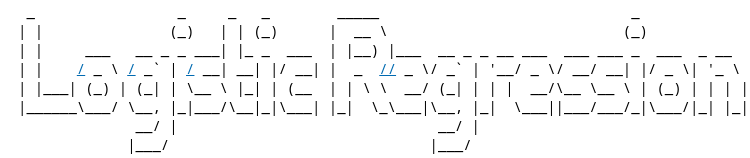
* F1-score
* MCC-score

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

## 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 trainati 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 valture 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 classificazzione 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% infeririori.

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.

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

# 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 velicizzare 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 extracion 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 fatures, 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.

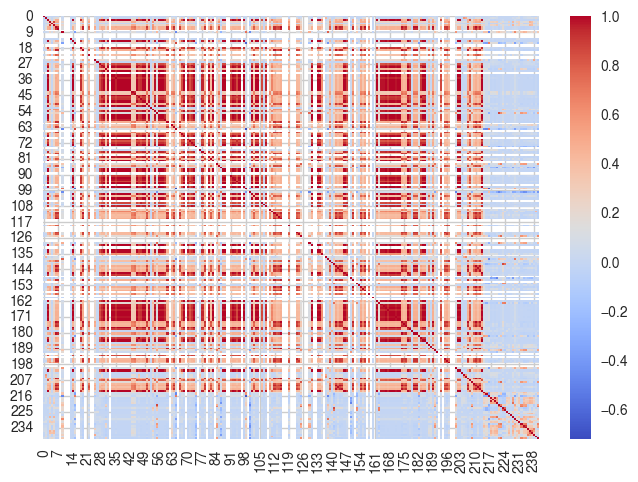
Dunque i passaggi che andremo ad eseguire per effettuare feature extraction sono:

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 procerede con l'eliminazione delle feature rindondanti. Questo perchè 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.

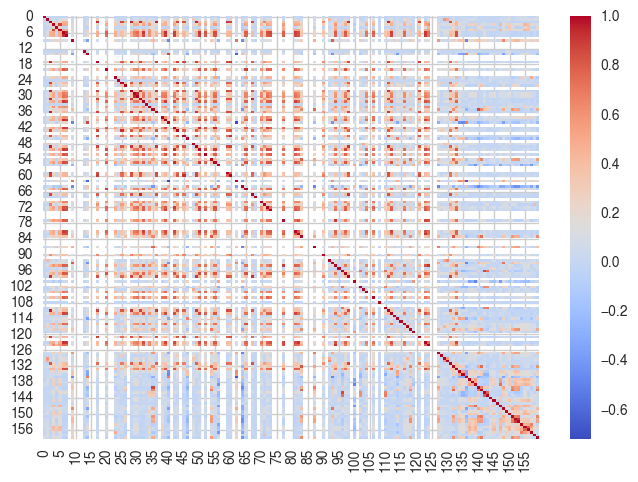
Si sviluppa quindi una funzione che permette di selezionare feature altamente correlate, più precisamente la funzione tornerà in output l’elenco di tutti i feature names la quale feature è correlata con un altra feature per almeno X%, dove X è un parametro preso in input (nel nostro caso avremo del X=95%). 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, ossia 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 perchè volgiamo applicare il metodo appena definito **solo sul train set** per poi verificare, tramite validazione con i modelli, si 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 allora possiamo applicare la rimozione delle stesse feature alla copia contenente tutto il dataset e procedere allo step successivo.

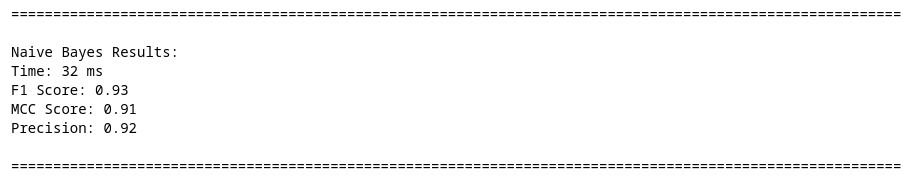
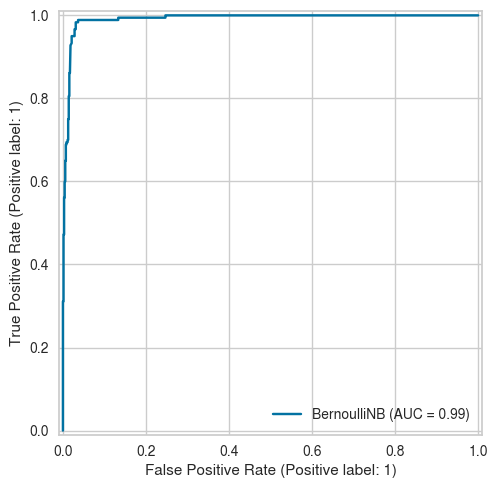
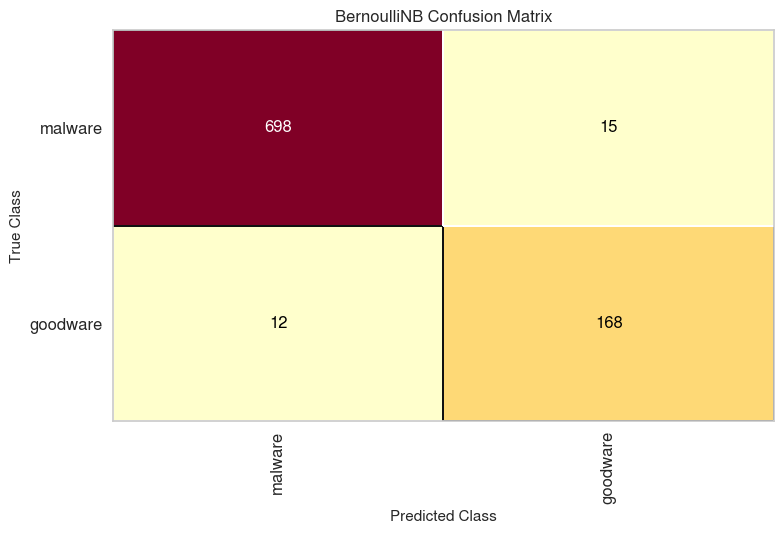
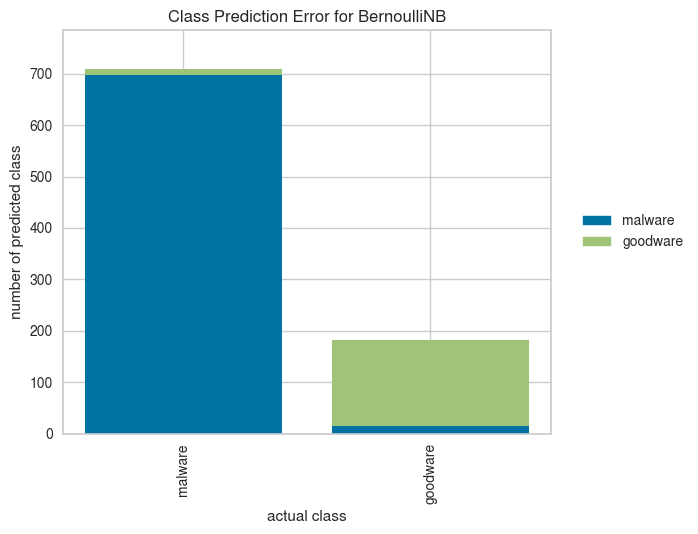
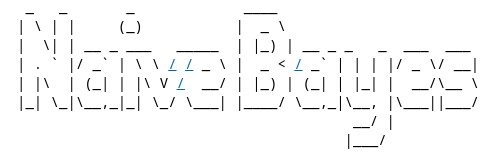
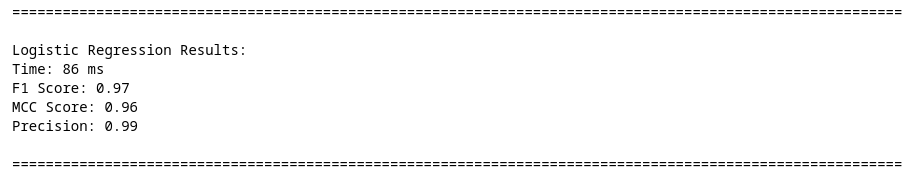
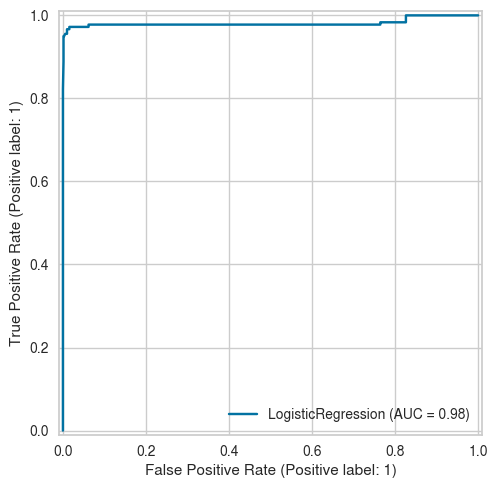
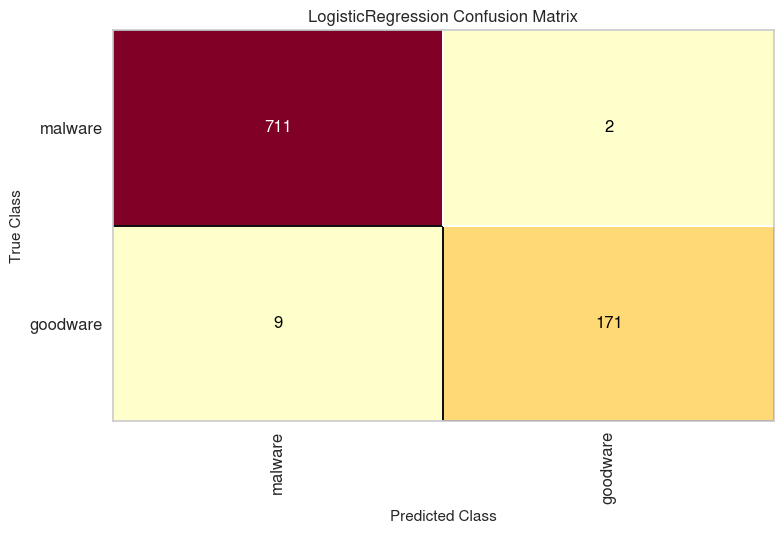
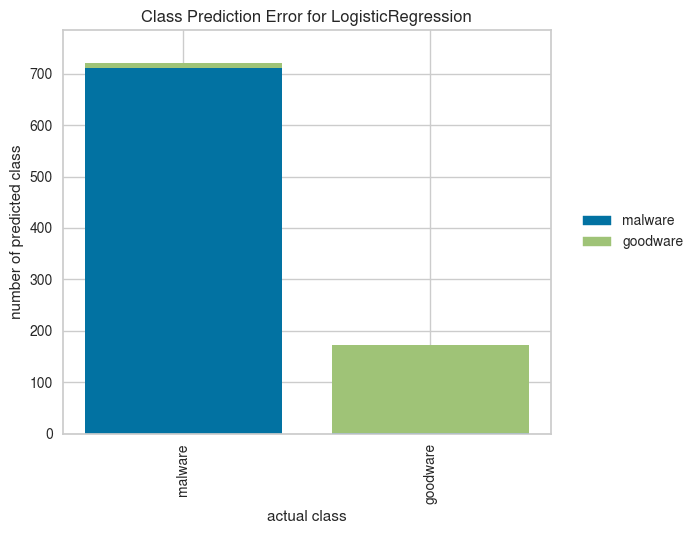
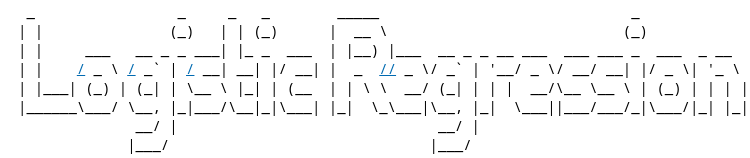
Se non effettuiamo questa operazione di validazione corriamo dunque il rischio di andare ad eliminare feature molto importanti e peggiorare di molto la classificazione. Per questo motivo seguiremo questa procedura di validazione divisione del dataset in train e test per validare le performance su anche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

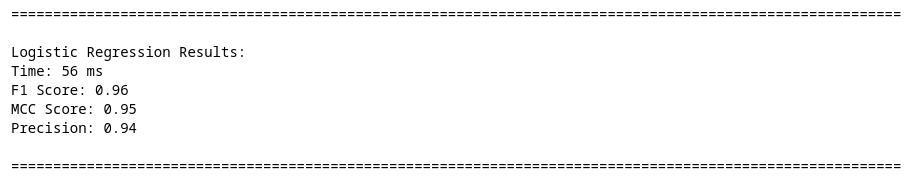
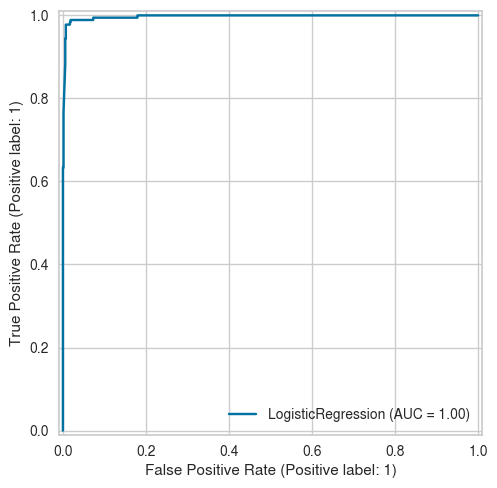
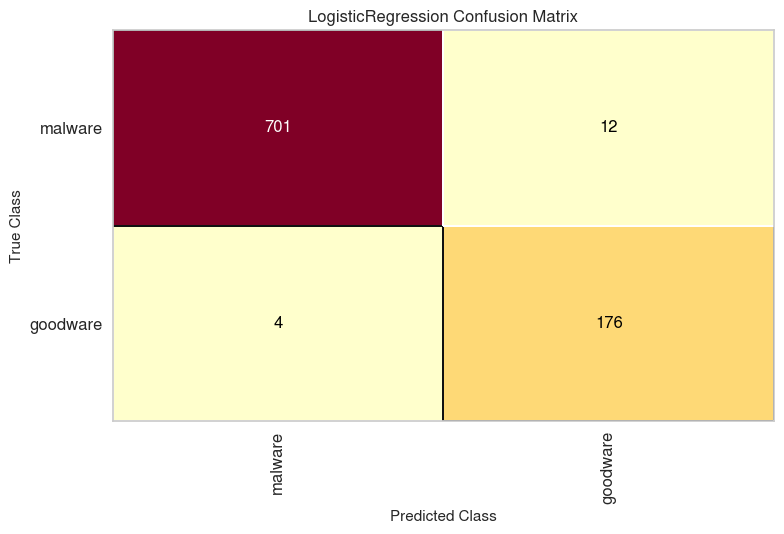
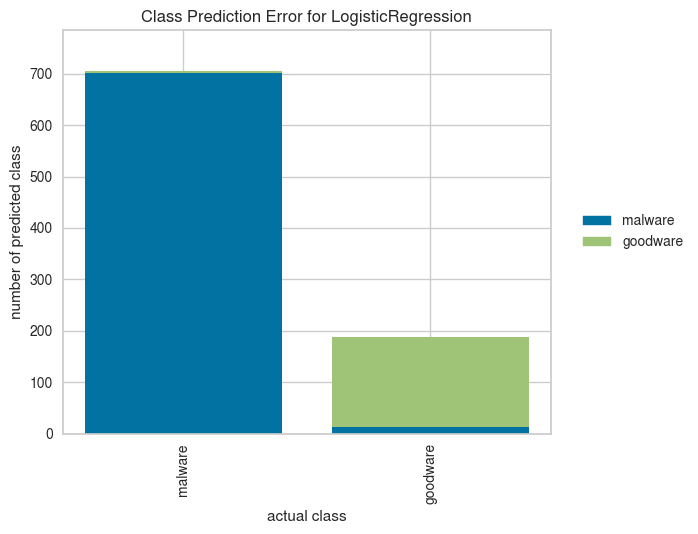
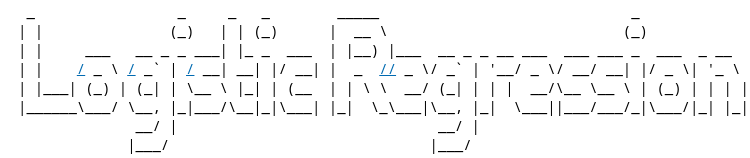
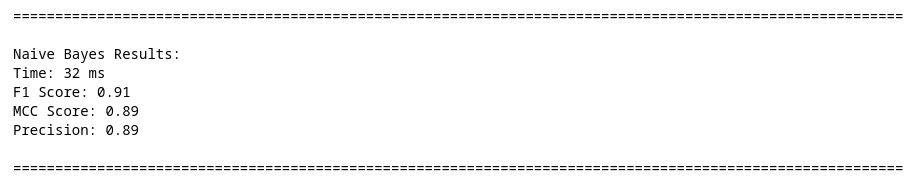
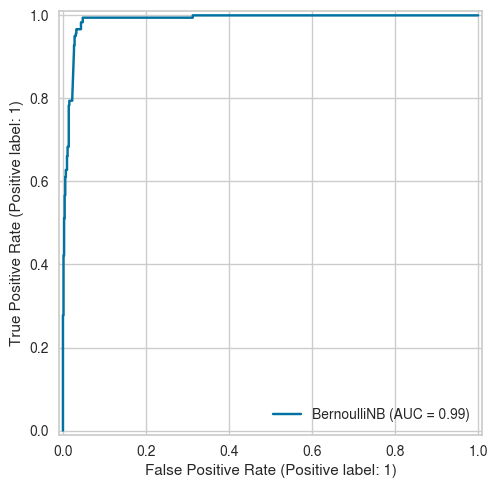
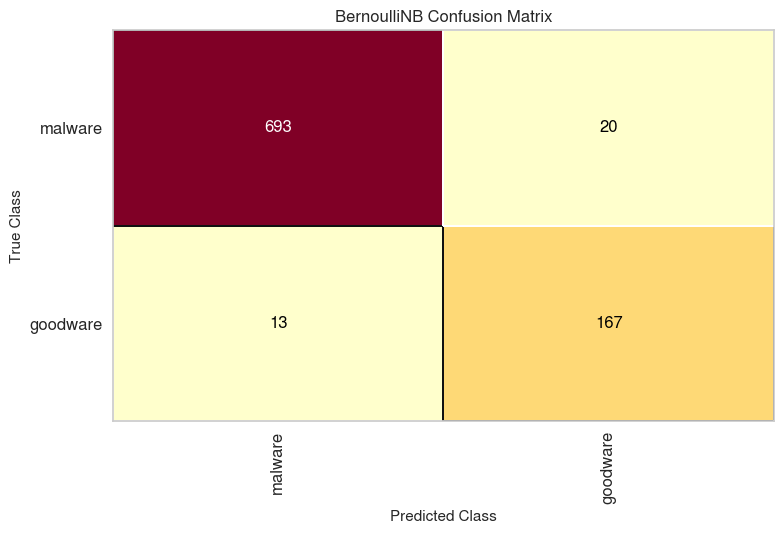
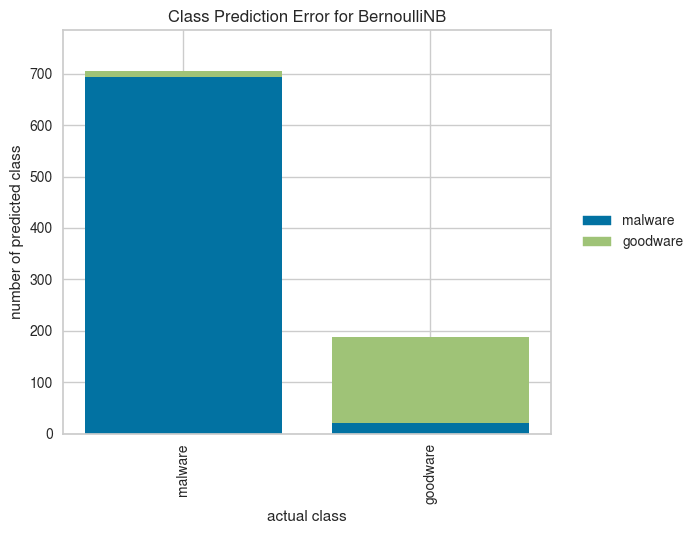
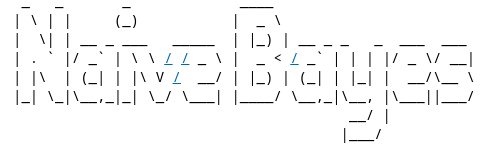
Anche qui dunque si sviluppa una nuova funzione. Questa calcolerà la correlazione di ciascuna feature con il target e poi restituirà le il nome colonne che hanno una correlazione strettamente inferiore alla soglia percentuale (X) scelta. Siccome dall’analisi del dataset fatta precedentemente abbiamo notao che ci sono molte poche feature con un alta correlazione con il target, per evitare di eliminare feature che potrbbero 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 suddviisione 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 165 feature con una correlazione infeariore al 5%, dopo l’eliminazione rimarrebbero solo 76 features.

I risultati sembrano abbastanza allarmanti, 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 misreo 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 dabase 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 osservazioni 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). Noi avremmo potuto applicare RFE fin da subito (senza applicare le elaborazioni precedenti) ma in quel caso i tempi di computazione sarebbero diventanti non accettabili (intorno ai 40 minuti). Invece applicando RFE dopo le 2 fasi precededenti 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 perchè questo è un modello giocattolo.