MALWARE DTECTION

ANALISI DEI DATI PER LA SICUREZZA

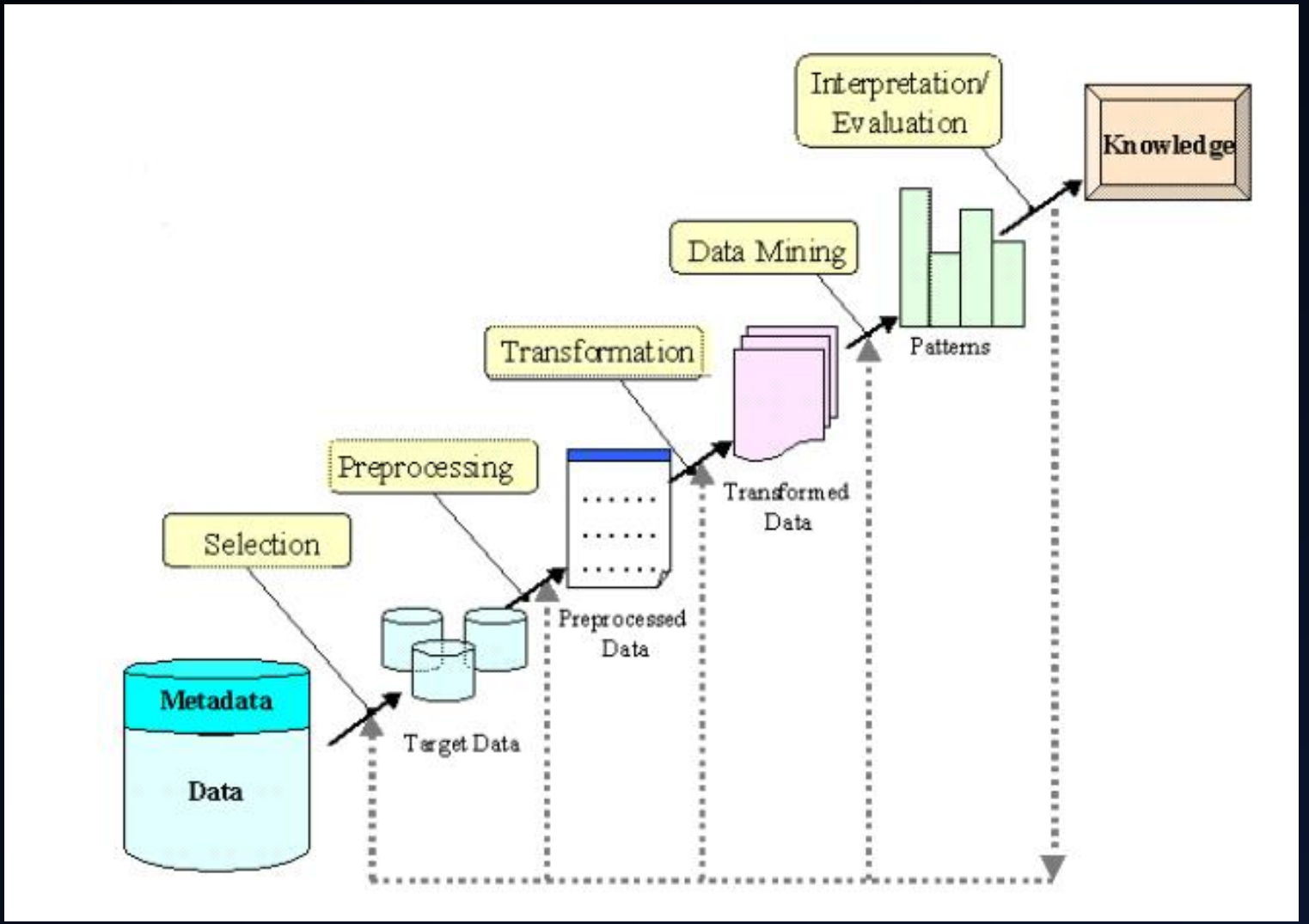
A.A 2023/2024

Giacomo Pagliara

Obiettivo del Progetto

- L’obiettivo del progetto è individuare, il modello migliore da addestrare e poi utilizzare per la Malware Detection.
- Dataset: Ember Dataset (https://github.com/elastic/ember)
- Dati Etichettati : (**0 Goodware**) – (**1 Malware**)
- Train Data Size: (12000, 2381)
- Test Data Size: (3000, 2381)

Il Processo KDD



Strutturazione del codice

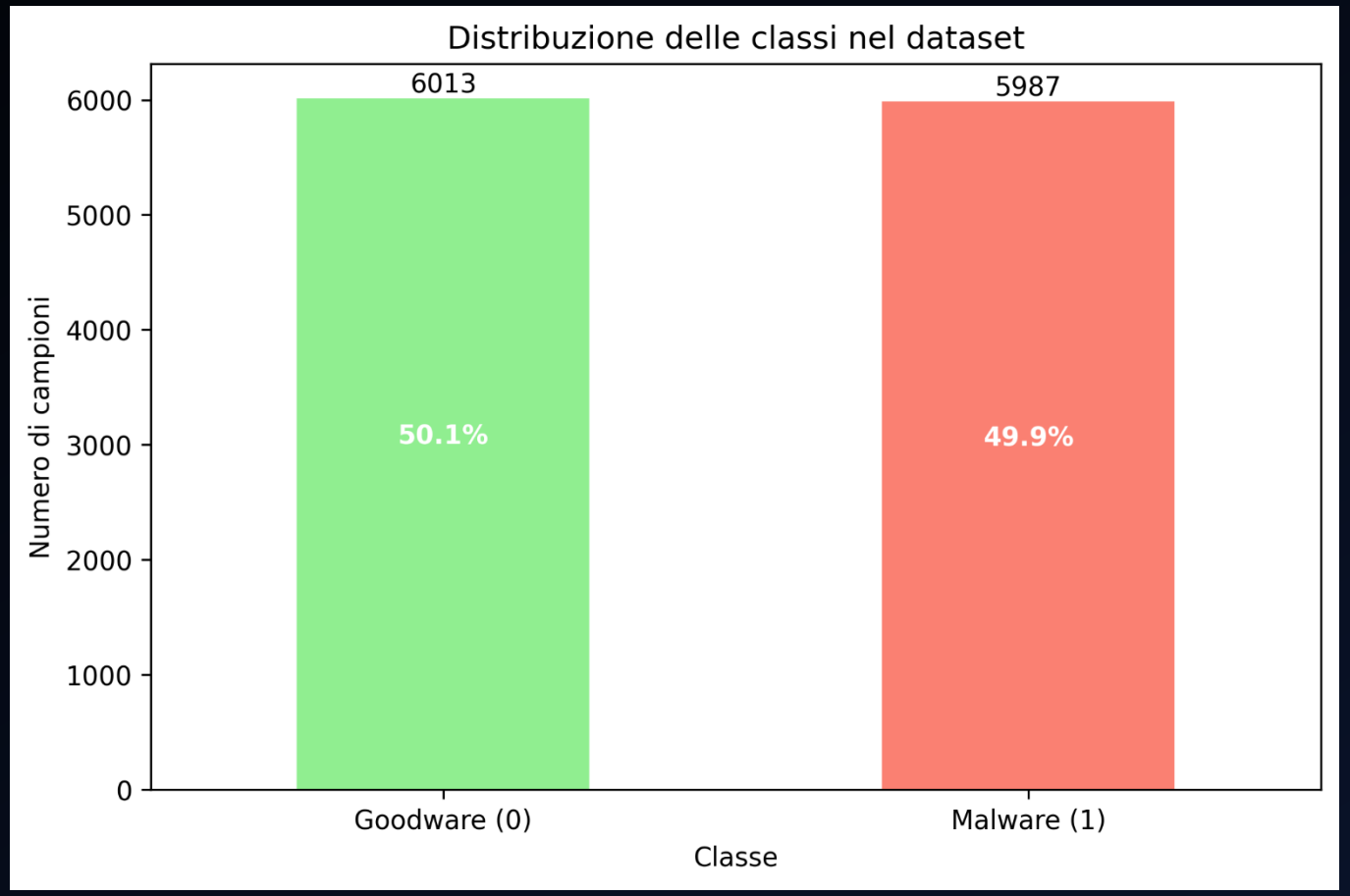
- Main.py
- DataSelection.py
- DataPreprocessing.py
- DataTransformation.py
- DataMining.py
- Utils.py

- Ciascun file fa riferimento a una fase precisa del KDD

Bilanciamento del dataset

```
def DistribuzioneClassi(y, plot=False):
```

```
Occorrenze per ogni classe:  Label
0.0    6013
1.0    5987
```



Ci troviamo ad analizzare un task di apprendimento supervisionato, dove ai dati di addestramento è stata già segnata un etichetta Malware o Goodware.

Dall'istogramma, si possono vedere le percentuali di distribuzione delle classi nel dataset.

Il dataset risulta ben bilanciato.

Rimozione Attributi

Per ottimizzare il dataset, è stata effettuata un'analisi sulle colonne(attributi) del training set. Sono stati rimossi gli attributi che presentavano lo stesso valore per tutti gli esempi, ovvero quelli in cui il valore minimo e massimo coincidevano.

Le stesse colonne sono state rimosse anche dal dataset di test per garantire coerenza.

Dimensione iniziale di x_train: (12000, 2381)

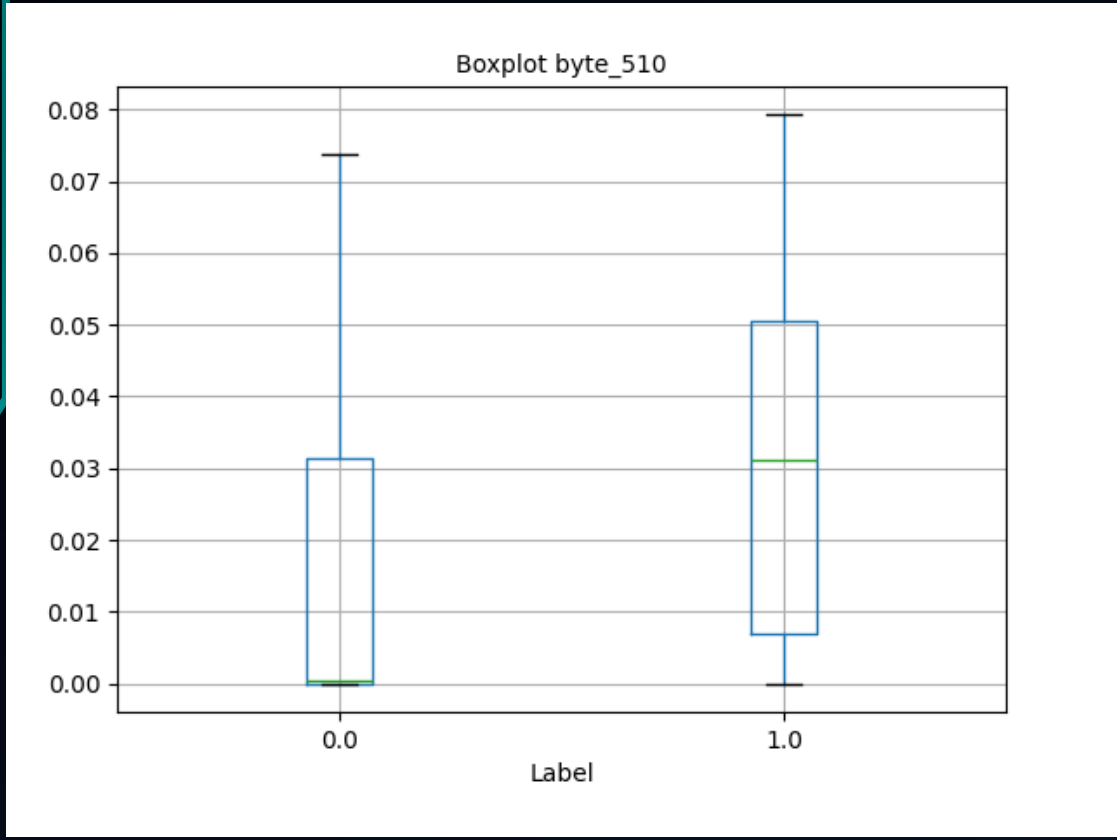
Dopo rimozione… ->

```
def removeColumns(X):
```

```
Dimensione di x_train_cleaned:  (12000, 2326)
```

Variabile Significativa

Subito dopo, sono state analizzate due variabili fondamentali.
La più significativa e la meno significativa.



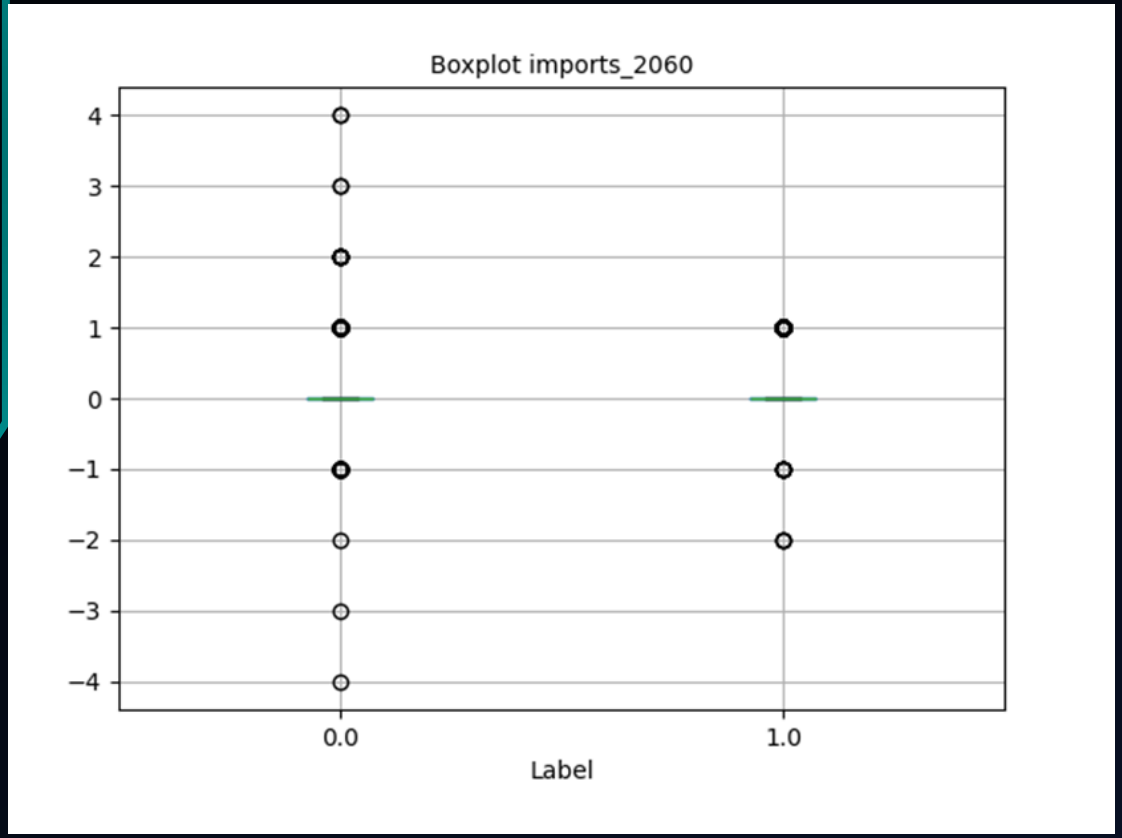
Mostra una differenza significativa nella distribuzione tra le due classi.

I malware (1) presentano valori mediani più alti rispetto ai goodware (0), i malware tendono ad avere valori più alti per questa caratteristica.

Proprio per questo, risulta un buon predittore e quindi importante per la classificazione.

Variabile con alta capacità discriminante

Variabile Non Significativa



Per i goodware (0) si notano molti outlier con valori sia positivi che negativi.

Per i malware (1) gli outlier sono più limitati.

E’ difficile quindi distinguere un malware da un goodware

Variabile con bassa capacità discriminante

MUTUAL INFO E PCA

MUTUAL INFO

La Mutual Info è una misura statistica che quantifica la dipendenza tra due variabili.

Selezionando solo le caratteristiche con «MI >= soglia» le dimensioni del dataset sono state ridotte, mantenendo solo le informazioni rilevanti. **Utilizzata per la feature selection**.

```
def mutualInfoRank(X, Y):
```

```
def topFeatureSelect(rank, threshold):
```

MUTUAL INFO E PCA

PCA

E’ una tecnica che permette di rimuovere la dipendenza dalle variabili e non effettua feature selection.

Sono state selezionate le prime N componenti, le quali spiegano la maggior parte della varianza.

Permettono di trasformare i dati in uno spazio di dimensioni ridotte ma informativo.

```
def pca(X):
```

```
def applyPCA(X, pca_obj, pc_names):
```

```
def NumberOfTopPCSelect(explained_variance, threshold):
```

ALGORITMI DI CLASSIFICAZIONE UTILIZZATI

Per la Malware Detection, abbiamo testato diversi classificatori:

- Decision Tree
- Random Forest
- K-Nearest Neighbors (KNN)
- Ensemble (ovvero la combinazione dei modelli precedenti)

Ciascun algoritmo è stato valutato sia con MI che con PCA.

Stratifield K-Fold Cross Validation

Per trovare i migliori parametri per ciascun modello è stata utilizzata la K-Fold Cross Validation.

Per la scelta delle configurazioni migliori è stato effettuato il confronto dell'FSCORE, che rappresenta una misura di accuratezza, calcolata come la media armonica tra precision e recall.

La K-fold cross validation:

- Mescola il set di dati in modo casuale;
- Suddivide il set di dati in k fold;
- Per ogni fold:
  - Prende un fold come set di test
  - I rimanenti come set di training
  - E poi addestra il modello con il set di training e lo valuta con quello di test

```
def stratifiedKfold(X, Y, folds, seed):
```

DECISION TREE

Il Decision Tree è un modello predittivo che organizza i dati in una struttura ad albero, dove ogni nodo interno rappresenta una decisione basata su un attributo, i rami indicano gli esiti di queste decisioni e le foglie corrispondono alle previsioni finali, ovver gli attributi target (classi).

Subito dopo, viene determinato il miglior criterio da utilizzare per il DT, sia per DT_MI che per DT_PCA.

Infine, viene creato il DT finale con i parametri migliori, generando il relativo report di classificazione e la matrice di confusione.

```
def decisionTreeLearner(X, y, c):
```

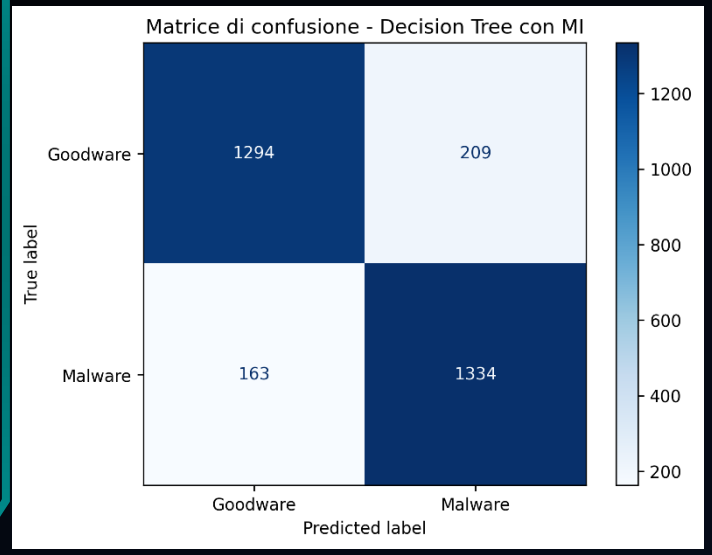
```
def determineDecisionTreekFoldConfiguration(ListXTrain, ListYTrain, ListXTest, ListYTest, rank, min_t, max_t,
                                            step,save_path):
```

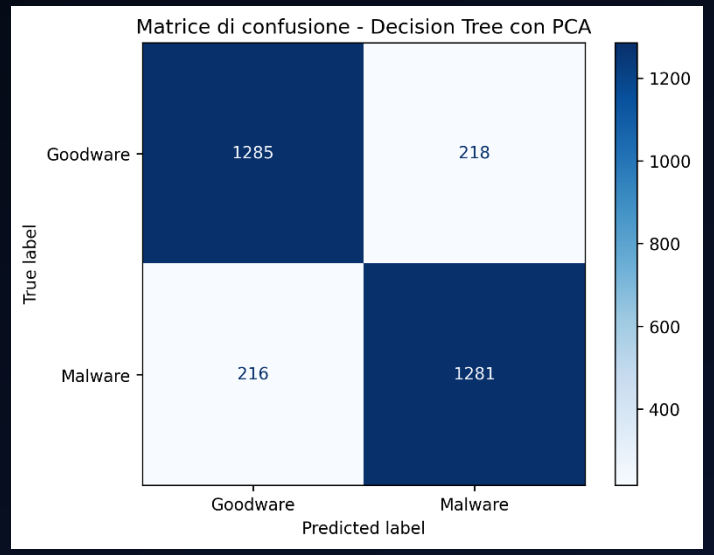
```
def determineDecisionTreekFoldConfigurationPCA(ListXTrain, ListYTrain, ListXTest, ListYTest, explained_variance,
                                               min_t,max_t, step, save_path):
```

DECISION TREE - RISULTATI

Feature Ranking by MI: Best criterion gini, best MI threshold 0.0, best N 2326, Best CV F 0.8721531952763458

PCA: Best criterion entropy, Best Threshold 1.0, Best components N 2326, Best CV F 0.8327404551160267





Classification Report:

	precision	recall	f1-score	support
0.0	0.89	0.86	0.87	1503
1.0	0.86	0.89	0.88	1497
accuracy			0.88	3000
macro avg	0.88	0.88	0.88	3000
weighted avg	0.88	0.88	0.88	3000

Classification Report:

	precision	recall	f1-score	support
0.0	0.86	0.85	0.86	1503
1.0	0.85	0.86	0.86	1497
accuracy			0.86	3000
macro avg	0.86	0.86	0.86	3000
weighted avg	0.86	0.86	0.86	3000

RANDOM FOREST

La Random Forest è un algoritmo di apprendimento automatico che combina l'output di più decision trees per raggiungere un unico risultato. La classe predetta sarà la classe più frequentemente predetta tra gli alberi della foresta.

Serve a cercare di risolvere il problema dell'overfitting.

```
def randomForestLearner(x, y, n_tree, c, rand, bootstrap_s, seed):
```

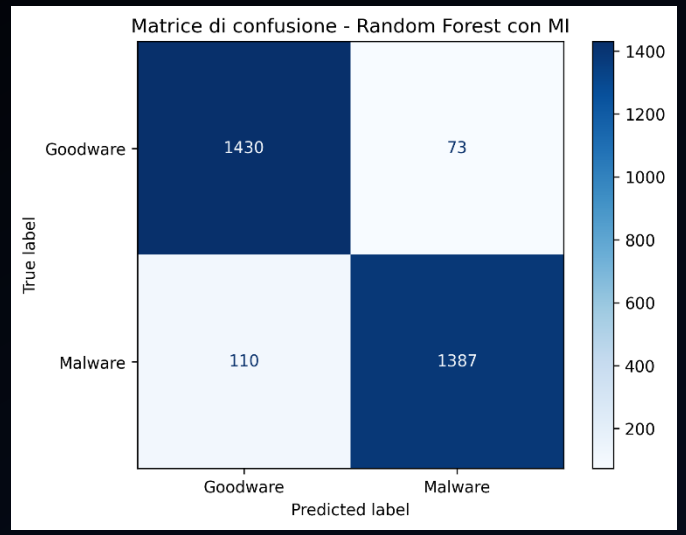
```
def determineRFkFoldConfiguration(ListXTrain, ListYTrain, ListXTest, ListYTest, rank, min_t, max_t, step,
                                  save_path):
```

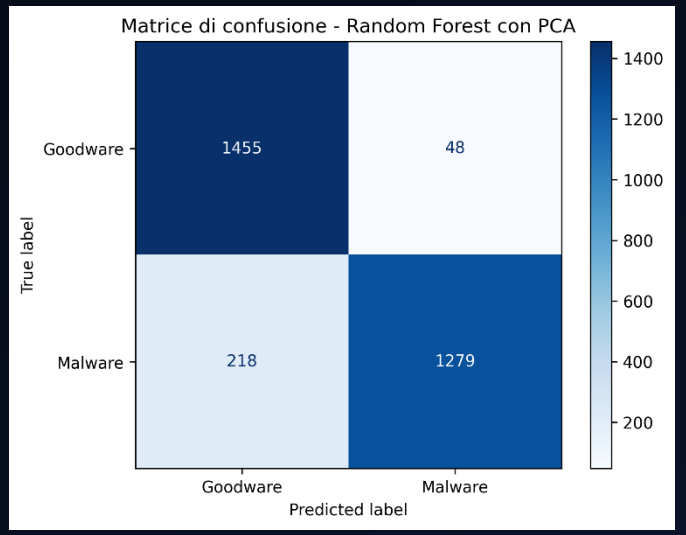
```
def determineRFkFoldConfigurationPCA(ListXTrain, ListYTrain, ListXTest, ListYTest, explained_variance,
                                     min_t, max_t, step, save_path):
```

RANDOM FOREST - RISULTATI

```
Random Forest: Best criterion gini, max_features sqrt, max_samples 0.8, n_estimators 30, MI threshold 0.02, features 733, Best CV F 0.93
54472770297916
```

```
Random Forest con PCA: Best criterion: entropy, Best Rand: sqrt, Best Bootstrap: 0.8, Best number Tree: 30, Best Threshold: 1.0, Best N
2326, Best CV F (Fscore): 0.8925158510979123
```





Classification Report:

	precision	recall	f1-score	support
0.0	0.93	0.95	0.94	1503
1.0	0.95	0.93	0.94	1497
accuracy			0.94	3000
macro avg	0.94	0.94	0.94	3000
weighted avg	0.94	0.94	0.94	3000

Classification Report:

	precision	recall	f1-score	support
0.0	0.87	0.97	0.92	1503
1.0	0.96	0.85	0.91	1497
accuracy			0.91	3000
macro avg	0.92	0.91	0.91	3000
weighted avg	0.92	0.91	0.91	3000

KNN

Il KNN viene tipicamente indicato come un algoritmo di apprendimento Lazzy learning. È definito pigro sostanzialmente perché non fa alcun apprendimento.

Predice la classe di un esempio in base ai «k» esempi più vicini. Per determinare la vicinanza viene utilizzata una funzione di distanza. Identificati i «k» più vicini, la classe più frequente tra questi viene assegnata all'esempio da classificare.

```
def KNNLearner(x, y, n_neighbors):
```

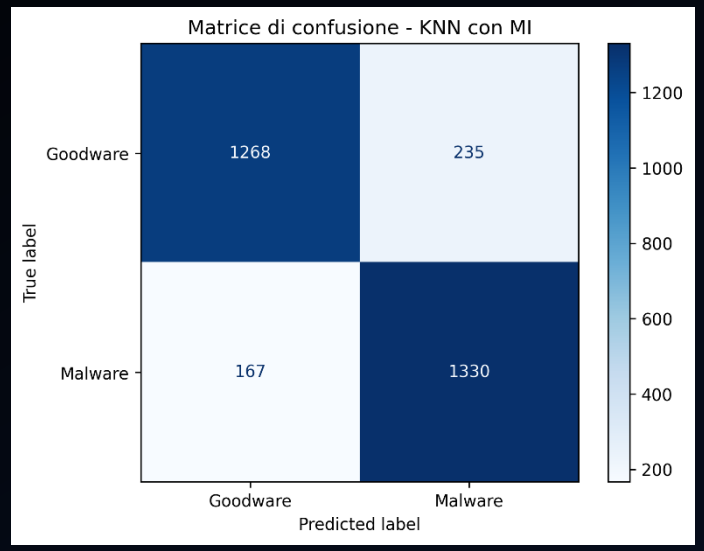
```
def determineKNNkFoldConfiguration(ListXTrain, ListYTrain, ListXTest, ListYTest, rank, min_t, max_t, step,
                                   save_path):
```

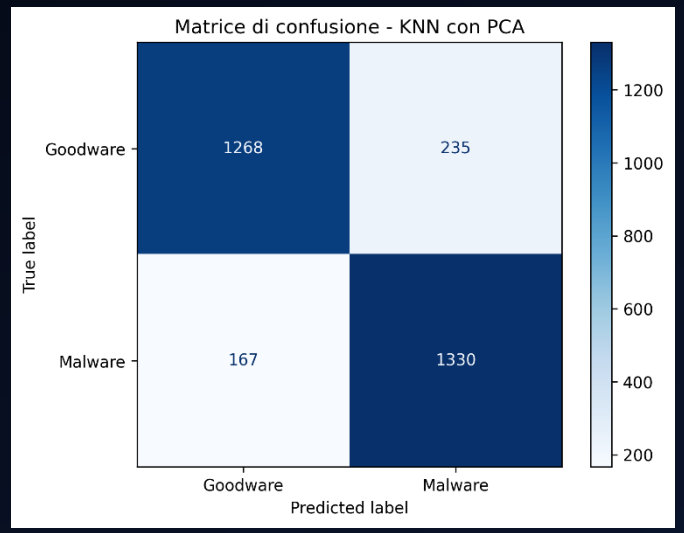
```
def determineKNNkFoldConfigurationPCA(ListXTrain, ListYTrain, ListXTest, ListYTest, explained_variance,
                                      min_t,
                                      max_t, step, save_path):
```

KNN - RISULTATI

```
KNN: Best k 1, MI threshold 0.0, features 2326, Best CV F 0.8673359393890653
```

```
KNN con PCA: Best k (vicini): 1, Best threshold: 1.0, Best N: 2326, Best CV F (fscore): 0.9099474627605055
```





Classification Report:

	precision	recall	f1-score	support
0.0	0.88	0.84	0.86	1503
1.0	0.85	0.89	0.87	1497
accuracy			0.87	3000
macro avg	0.87	0.87	0.87	3000
weighted avg	0.87	0.87	0.87	3000

Classification Report:

	precision	recall	f1-score	support
0.0	0.88	0.84	0.86	1503
1.0	0.85	0.89	0.87	1497
accuracy			0.87	3000
macro avg	0.87	0.87	0.87	3000
weighted avg	0.87	0.87	0.87	3000

ENSEMBLE

Nella fase finale sono stati combinati diversi modelli, tra cui: **Decision Tree, Random Forest e KNN.**

Nell'Ensemble è importante settare il parametro «voting», che può essere avvalorato con due valori:

- **Soft**: Ogni modello restituisce la **probabilità** di ogni classe, si fa la **media delle probabilità**, e poi si prende la classe con la **probabilità più alta** (ovvero, l'argmax).

- **Hard:** Ogni modello vota la **classe predetta** (es. 0 o 1),e poi viene scelta la **classe più votata** tra i modelli. (**PARAMETRO SCELTO**).

```
def EnsembleLearner(x, y, dt, rf, knn):
```

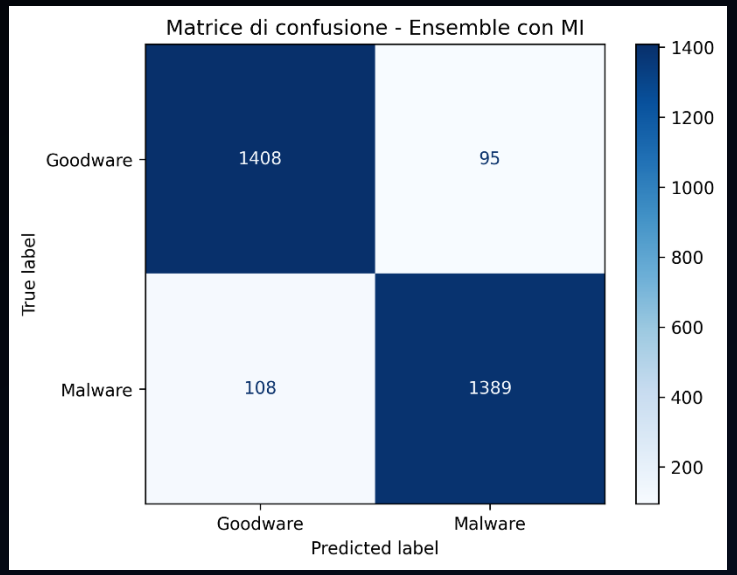
```
def determineEnsembleMIkFoldConfiguration(ListXTrain, ListYTrain, ListXTest, ListYTest, rank, min_t, max_t,
                                          step, DT_mutual_info, RFMI, KNNMI, save_path):
```

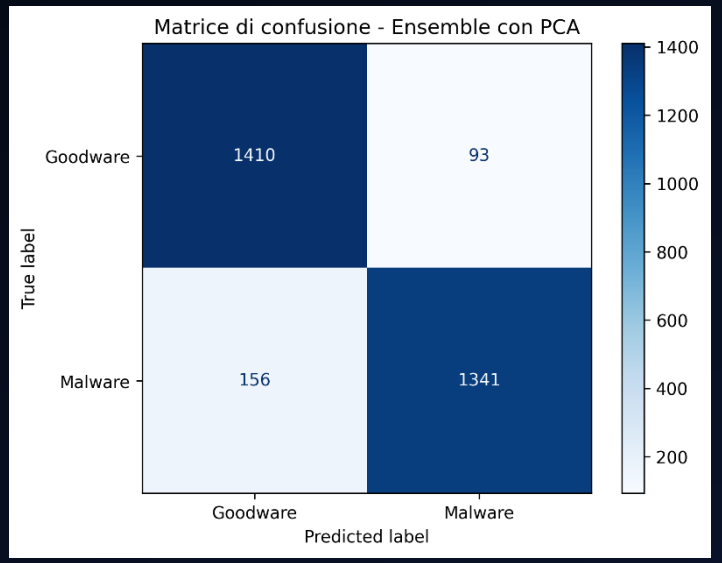
```
def determineEnsemblePCAkFoldConfiguration(ListXTrain, ListYTrain, ListXTest, ListYTest, explained_variance,
                                           min_t,max_t, step, DTPCA, RFPCA, KNNPCA, save_path):
```

ENSEMBLE - RISULTATI

```
Ensemble con MI: threshold 0.08, features 631, F1 0.9310004235477372
631 features selezionate con informazione mutua >= 0.08
```

```
Ensemble con PCA: Best threshold: 1.0, Best N: 2326, Best CV F (Fscore) 0.9033366870015506
```





Classification Report:

	precision	recall	f1-score	support
0.0	0.93	0.94	0.93	1503
1.0	0.94	0.93	0.93	1497
accuracy			0.93	3000
macro avg	0.93	0.93	0.93	3000
weighted avg	0.93	0.93	0.93	3000

Classification Report:

	precision	recall	f1-score	support
0.0	0.90	0.94	0.92	1503
1.0	0.94	0.90	0.92	1497
accuracy			0.92	3000
macro avg	0.92	0.92	0.92	3000
weighted avg	0.92	0.92	0.92	3000

CONCLUSIONI

Per poter valutare al meglio un determinato modello occorre:

- Utilizzare un numero minore di feature;
- Addestrare più dati in meno tempo;
- Avere impatti infrastrutturali minori.

MALWARE DTECTION

ANALISI DEI DATI PER LA SICUREZZA

A.A 2023/2024

Grazie per l’attenzione.