ANALISI DEI DATI PER LA SICUREZZA
A.A 2023/2024

Cellammare Gabriel

“MALWARE DETECTION”

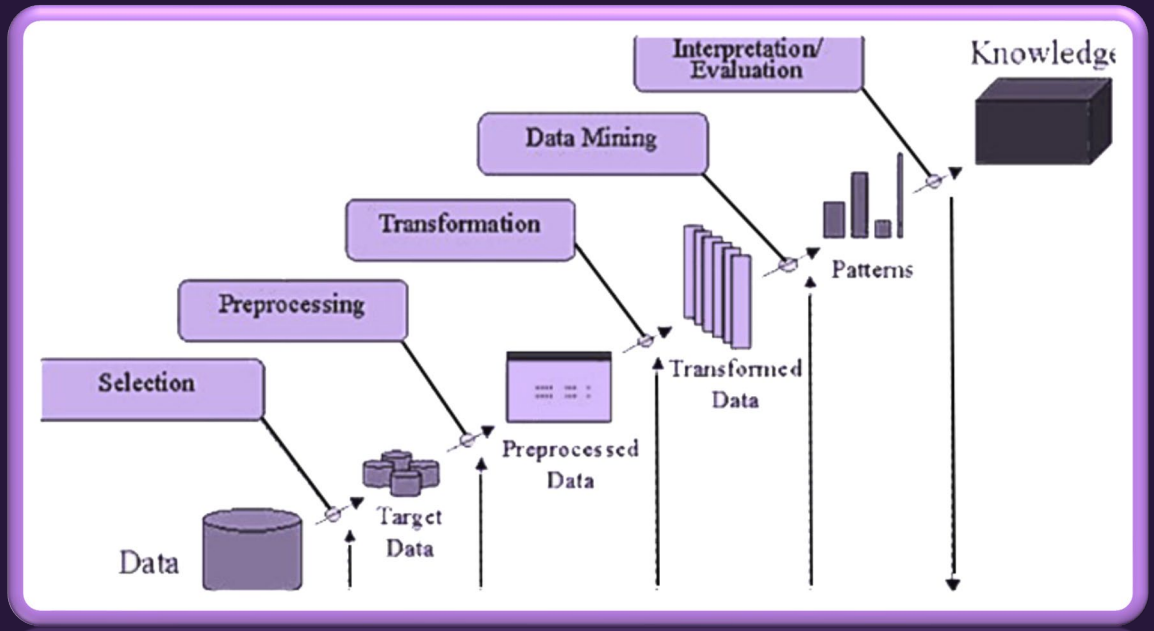
Matricola: 807350
Email: g.cellammare1@studenti.uniba.it

Obiettivo del progetto

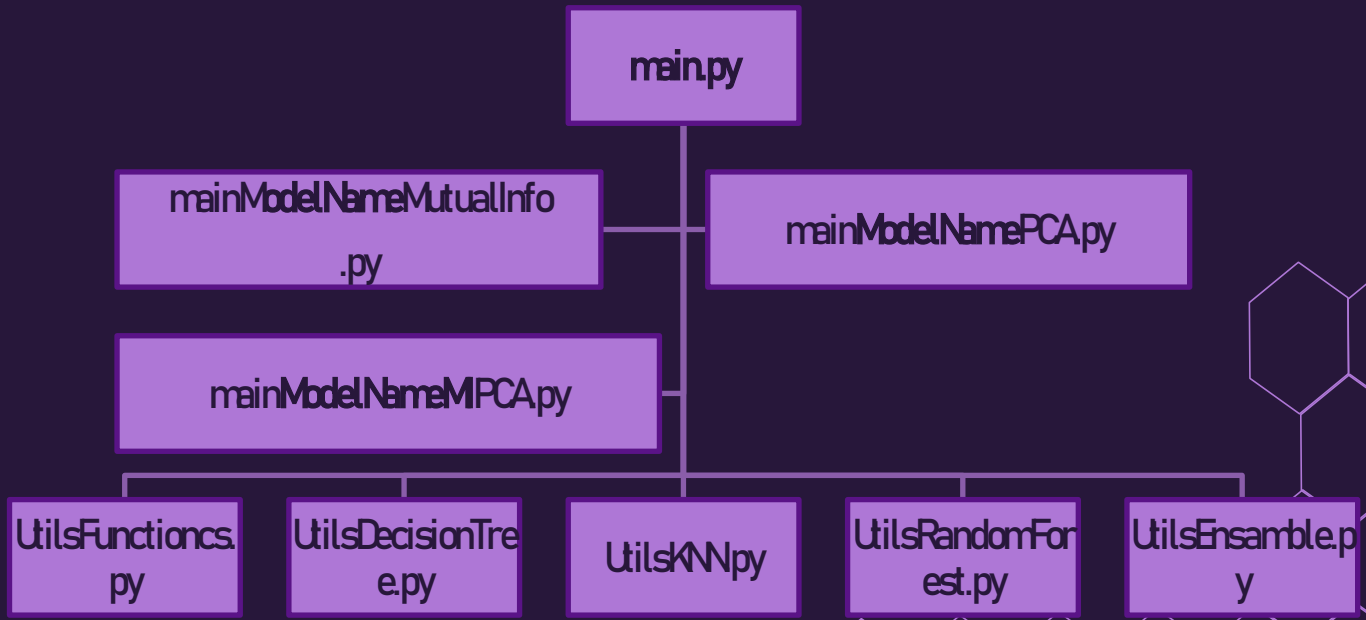
L'obiettivo del progetto è stato quello di individuare il modello migliore da addestrare e successivamente da utilizzare per la Malware detection, attraverso l'ausilio del **processo KDD.**

- **DATASET:** EMBER (https://github.com/elastic/ember)
- **Python 3.12**
- **Librerie**: Scikit-Learn, pandas, numpy, pickle
- **Data labeled** (0 Goodware – 1 Malware)
- **Train Data** (12000, 2381)
- **Test Data** (3000, 2381)

Il processo KDD

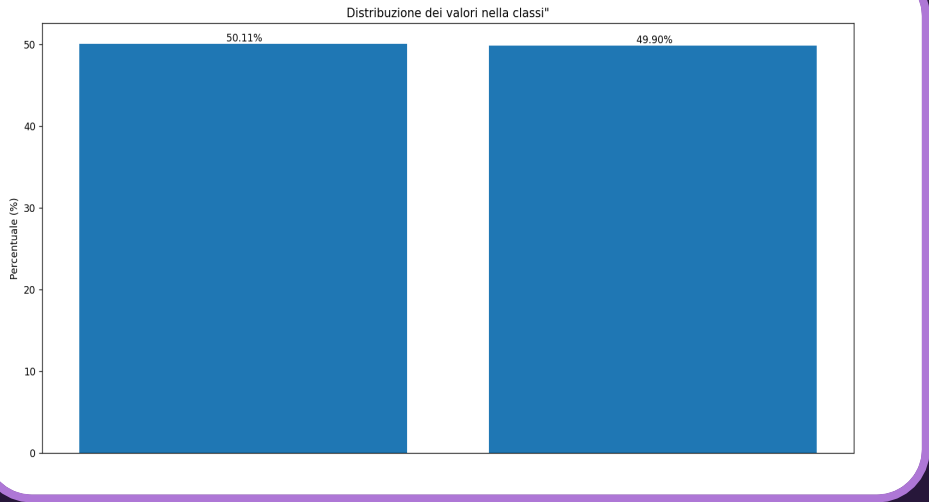


Strutturazione del codice



Domain and data understanding

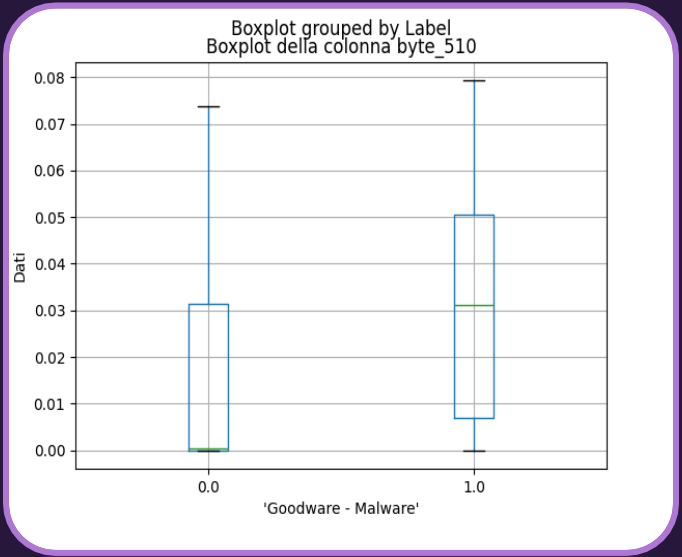
Durante questa fase, sono state visualizzate alcune caratteristiche dei dati



In questo caso, è stato visualizzato il numero degli esempi per ogni classe, evidenziando una proporzionalità tra le due.
Queste informazioni sono necessarie per strutturare le fasi successive del ***Processo***.

Domain and data understanding: Data exploration

Successivamente, è stata analizzata una delle variabili più significative

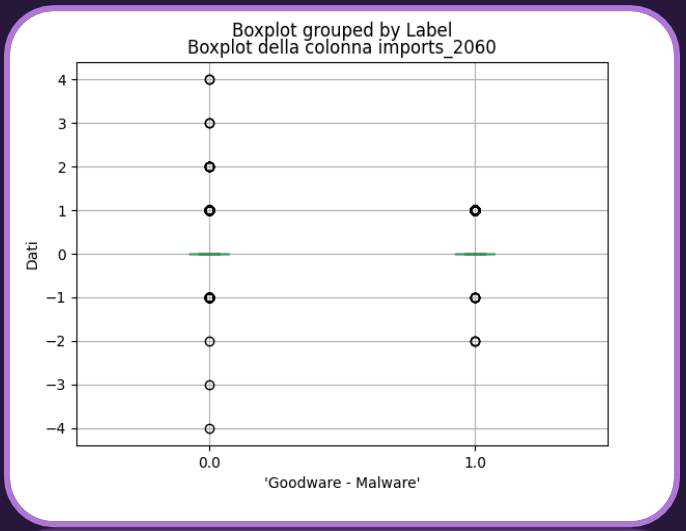


Statistiche complessive della colonna 'byte_510':

- count 12000.000000
- mean 0.022943
- std 0.023579
- min 0.000000
- 25% 0.000077
- 50% 0.013214
- 75% 0.050528
- max 0.079272

Domain and data understanding: Data exploration

In seguito, è stata paragonata alla variabile suddetta, che non discrimina nello specifico le due classi, come la precedente



Statistiche complessive della colonna 'imports_2060':

- count 12000.000000
- mean 0.024167
- std 0.205960
- min -4.000000
- 25% 0.000000
- 50% 0.000000
- 75% 0.000000
- max 4.000000

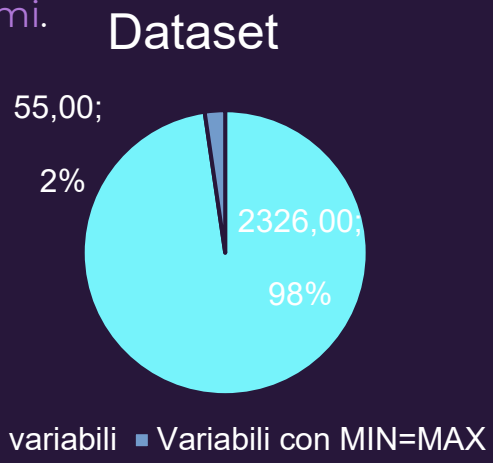
Data selection: Rimozione variabili indipendenti

Nelle fasi precedenti, i dati sono stati soltanto visualizzati e analizzati.
In questa fase invece sono state eliminate dapprima le variabili che possedevano valori minimi uguali a quelli massimi.

```
Shape di Train_x before removing min=max: (12000, 2381)
```

```
Nuovo dataset di training con dimensione: '(12000, 2326)'
```

Le medesime colonne, dopo essere state salvate, sono state coerentemente rimosse anche nel dataset di test



Perché rimuovere tali variabili?

Data selection: Automatically feature selection (Mutual Info)

Sono state confrontate due tecniche diverse: Mutual Info e PCA

```python
def mutualInfoRank(X, Y):
    print("Computing mutual info ranking...")
    # Restituisce i nomi delle colonne
    independentList = list(X.columns.values)
    res = dict(zip
                (independentList,
                    mutual_info_classif
                    (X,  np.ravel(Y), discrete_features=False, random_state=seed)))
    # La funzione zip accoppia ciascun nome di colonna con il corrispondente valore di informazione mutua, creando una se

    sorted_x = sorted(res.items(), key=lambda kv: kv[1], reverse=True)

    print("Computing mutual info ranking...completed")
    # ritorna un dizionario
    return sorted_x
```

Data selection: Automatically feature selection (Mutual Info)

Successivamente, sono stati salvati i box plot delle prime n variabili indipendenti con un valore di mutual info molto alto e le ultime n con uno molto basso

```python
def BoxPlotAnalysisDataMutualInfo(x, y, boxPlotDir, mutualInfo, n_print=10):
    # Ottieni la lista dei file nella cartella
    print("\nSaving Mutual info variables Box Plot in 'BoxPlotMutualInfo' Folder...\n")

    boxPlotDirMutualInfoFirst = boxPlotDir / "MoreSignificant"
    boxPlotDirMutualInfoLast = boxPlotDir / "LessSignificant"

    if (not os.listdir(boxPlotDirMutualInfoFirst)):
        i = 0
        if (mutualInfo):
            x['Label'] = y['Label']
            for tupla in mutualInfo:
                for col in x.columns:
                    # capire se va 0 considerato o meno
                    if (tupla[0] == col and tupla[1] != 0):
                        if (i < n_print):
                            x.boxplot(column=col, by='Label')
                            # Aggiungi titoli e label per gli assi, se necessario
                            plt.title(f'Boxplot della colonna {col}')
                            plt.xlabel(goodMalware)
                            plt.ylabel('Dati')
                            file_name = os.path.join(
                                boxPlotDirMutualInfoFirst, f'boxplot_{col}.png')
                            plt.savefig(file_name)
                            plt.close()
                            i += 1
```

Data trasformation: (PCA)

In seguito, è stata applicata la PCA, che permette di trasformare le variabili attraverso combinazioni lineari, riducendo la loro dipendenza dalle altre

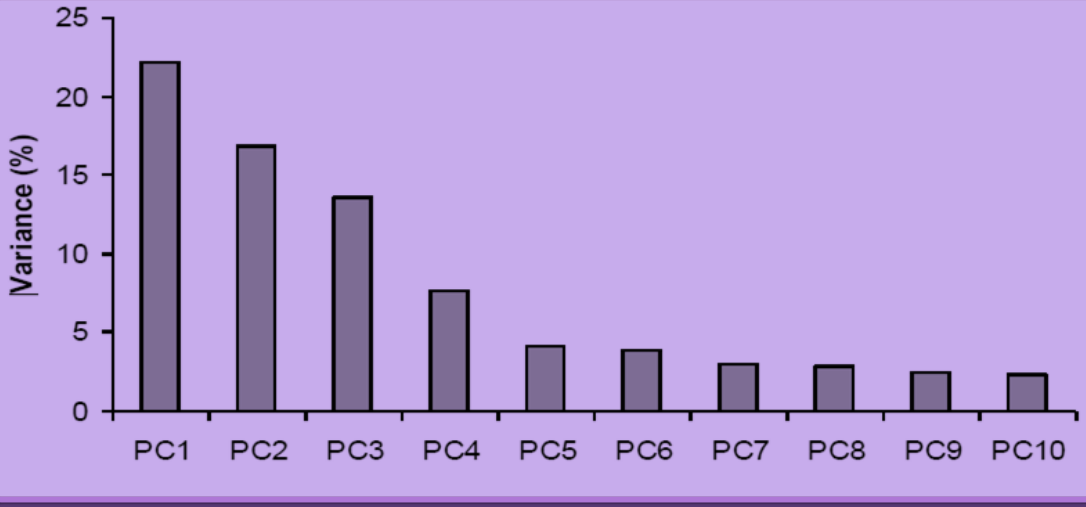
```python
def pca(X):
    print("\nTraining PCA...\n")
    pca = PCA(n_components=len(X.columns))
    pca.fit(X)
    # Nome delle nuove feature del tipo pca0...pcan
    feature_names = pca.get_feature_names_out()
    print("\nCompleted!\n")
    return pca, feature_names, pca.explained_variance_ratio_
```

```python
def applyPCA(X, pca, pcalist):
    print("\nApplying PCA...\n")
    # Trasforma il DataFrame utilizzando PCA
    transformed = pca.transform(X)
    # Crea un nuovo DataFrame con le componenti principali
    df_pca = pd.DataFrame(transformed, columns=pcalist)
    print("\nCompleted!\n")
    return df_pca
```

```python
def NumberOfTopPCSelect(explained_variance, thresold):
    cumulative_variance = 0.0
    num_components = 0

    for variance in explained_variance:
        cumulative_variance += variance
        num_components += 1

        if cumulative_variance > thresold:
            break

    return num_components
```

Data trasformation: (PCA)

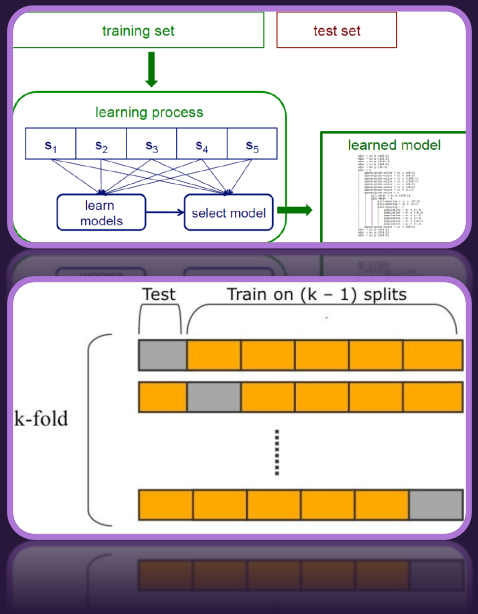
La PCA è una tecnica che permette di rimuovere la dipendenza dalle variabili e non di effettuare feature selection, ma in questo caso sono state selezionate le prime n variabili poiché «spiegano» la maggior parte della varianza del set di dati



Determine Best Configuration

Per determinare la migliore configurazione di ogni algoritmo, e successivamente per rendere più efficiente la fase di valutazione, è stata utilizzata la tecnica K-Fold Cross Validation, che:

- Mescola il set di dati in modo **casuale;**
- Suddivide il set di dati in k «**fold**»
- Per ogni **fold**:
- Prende il **fold** come set di test;
- Prende i **fold** rimanenti come set di dati di **training;**
- Addestra il modello scelto con il set di **training** e lo valuta con il set di **test**



Determine Best Configuration

Per effettuare tale procedura, è stata utilizzata la classe **StratifiedKFold** di scikit-learn

```
def stratifiedKFold(X, Y, folds=5):
    skf = StratifiedKFold(n_splits=folds, shuffle=True, random_state=seed)
    ListXTrain = []
    ListXTest = []
    ListYTrain = []
    ListYTest = []
    for i, (train_index, test_index) in enumerate(skf.split(X, Y)):

        ListXTrain.append(pd.DataFrame(X, index=train_index))
        ListYTrain.append(pd.DataFrame(Y, index=train_index))
        ListXTest.append(pd.DataFrame(X, index=test_index))
        ListYTest.append(pd.DataFrame(Y, index=test_index))
```

Classification models: Decision Tree

Il primo modello utilizzato è stato il **Decision Tree**

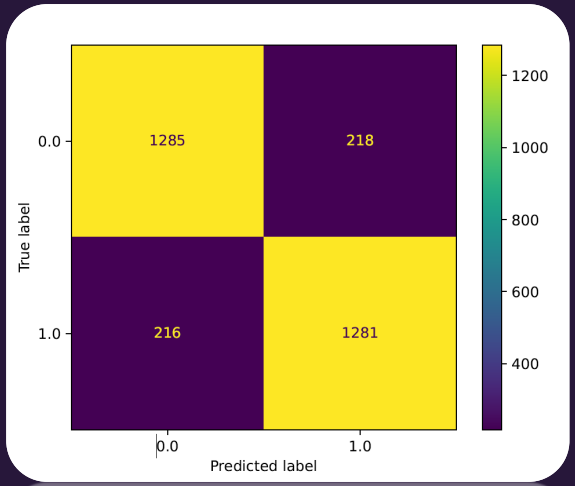
Il parametro **criterion** indica Gini o Entropy

```python
def decisionTreeLearner(X, Y, c):
    clf = DecisionTreeClassifier(criterion=c, random_state=seed)
    clf.min_samples_split = 500  # Numero minimo di esempi per mettere uno split
    # Tree_ dentro c'è l'albero addestrata
    clf.fit(X, Y)
    # print(f"Number of nodes: {clf.tree_.node_count}")
    # print(f"Number of leaves: {clf.get_n_leaves()} ")
    # Return the number of leaves of the decision tree.)
```

```python
def showTree(clf, script_pathTreeFolder, TreeType):

    plt.figure(figsize=(15, 10))
    tree.plot_tree(clf,
                   filled=True,
                   rounded=True)
    file_name = os.path.join(script_pathTreeFolder,
                             f'TreeFigOutput{TreeType}.pdf')
    plt.savefig(file_name,
                format='pdf', dpi=1000)
    plt.show()
```

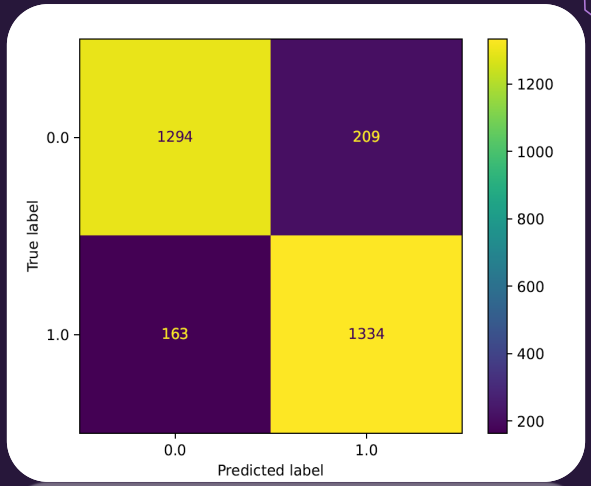
Attraverso la funzione showTree è stato possibile salvare in pdf l'intero albero con delle specifiche ricevute come parametri della stessa

Evaluation models: Decision Tree



PCA

F-Score
86%



Mutual Info

F-Score
88%

Classification models: Instance-based

Il secondo modello utilizzato è stato un modello **Instance-Based**, nello specifico l'algoritmo **KNN**

```python
def knnLearner(x, y, n_neighbors):
    knn = KNeighborsClassifier(n_neighbors=n_neighbors)
    knn.fit(x, np.ravel(y))
    return knn
```

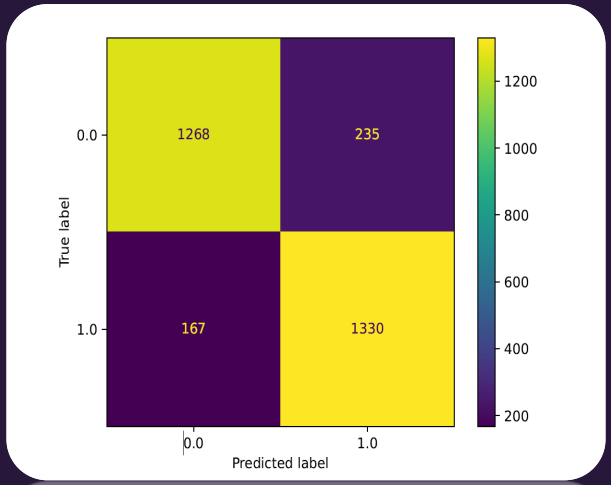
La variabile **K** indica il numero dei vicini considerati

Il comando ravel di numpy, trasforma l'input in un array monodimensionale contiguo

```python
report = classification_report(y_test, y_pred, target_names=target_names)
print(report)

nome_file = "classification_report(KNN_MutualInfo).txt"
script_pathClassification = script_path.parent.parent / \
    "ClassificationReport" / "KNN"

percorso_file = os.path.join(script_pathClassification, nome_file)

# Apre il file in modalità scrittura
with open(percorso_file, 'w') as file:
    # Scrive il classification report nel file
    file.write(report)

script_pathFolder = script_path.parent.parent / "ConfusionMatrix" / "KNN"
ConfusionMatrixBuilder(
    KNN, y_pred, y_test, script_pathFolder, "KNNMutualInfo")
```

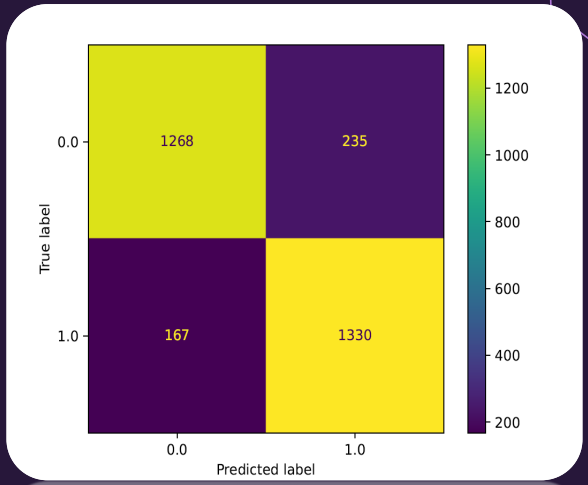
Così come il **Decision Tree**, è stato salvato il **classification report** con la relativa **matrice di Confusione** nella cartella selezionata

Evaluation models: Instance-based



PCA

F-Score
87%



Mutual Info

F-Score
87%

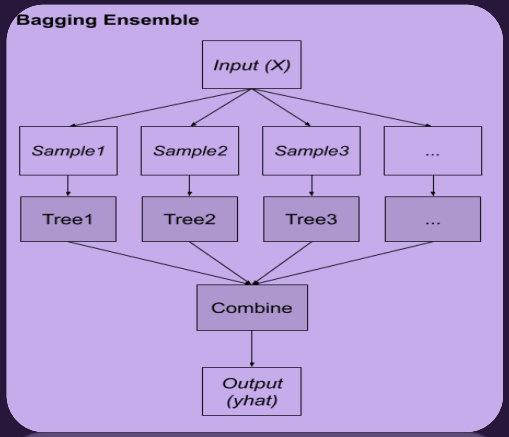
Classification models: Random forest

Il modello che ha restituito un livello di accuratezza più alto, è stato il Random Forest

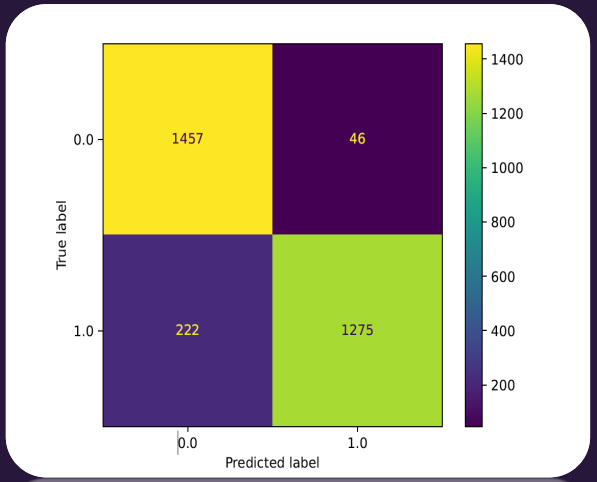
```
def randomForestLearner(x, y, n_tree, c, rand, bootstrap_s):
    rlf = RandomForestClassifier(n_estimators=n_tree, criterion=c,
                                 max_features=rand, max_samples=bootstrap_s, random_state=seed)
    # Min samples risultati inferiori
    rlf.fit(x, np.ravel(y))

    return rlf
```

I parametri indicano:

- **n_estimators:** Numero degli alberi della foresta
- **Criterion**: Gini o Entropy
- **max_features**: sqrt o log2 il numero delle feature da considerare per il calcolo del best split
- **max_samples**: Numero di campioni utilizzati in ogni decision tree
- **random_state**

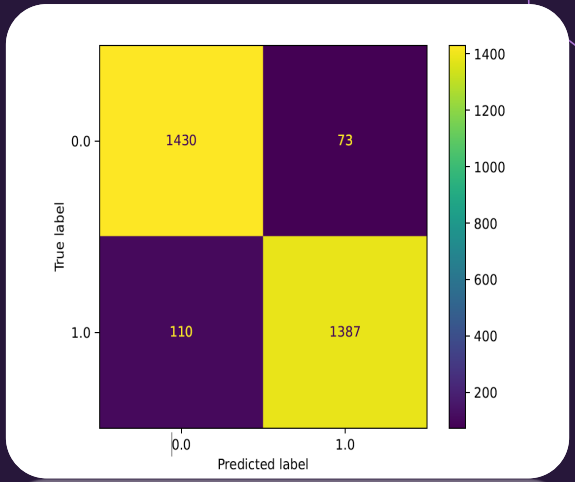


Evaluation models: Random forest



PCA

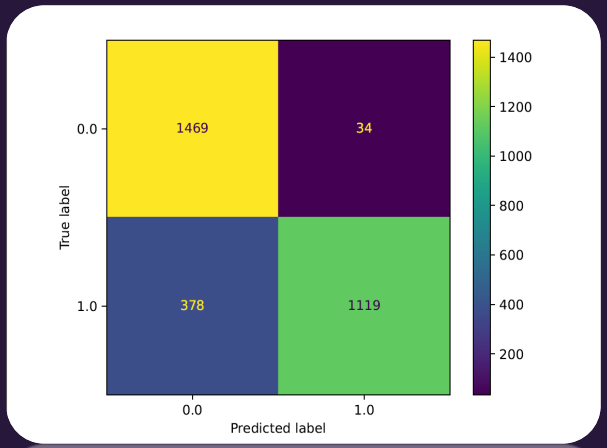
F-Score
92%



Mutual Info

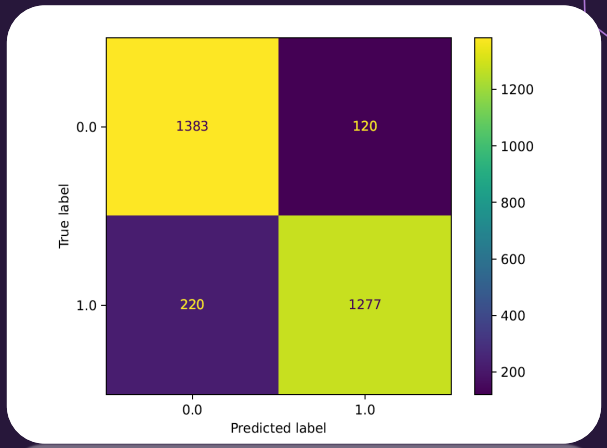
F-Score
94%

Evaluation models: Random forest (Min samples 500)



PCA

F-Score
86%



Mutual Info

F-Score
88%

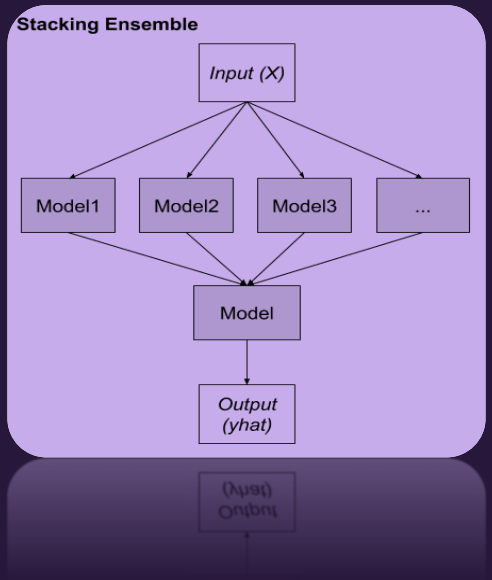
Classification models: Ensamble

Nella fase finale è stato valutato l'ensemble, formato dal Decision Tree, KNN e Random Forest

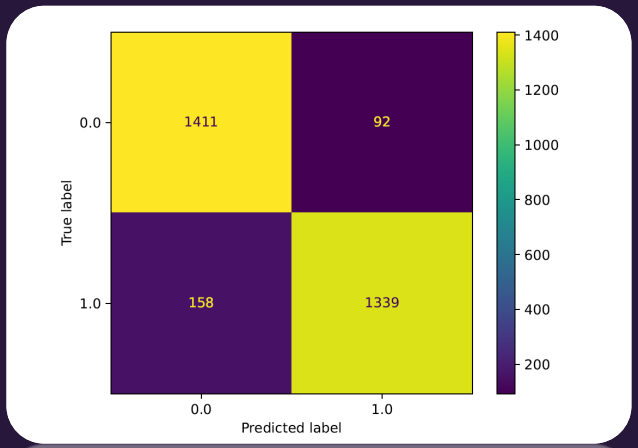
```
def EnsambleLearner(x, y, clf1, clf2, clf3):
    eclf = VotingClassifier(
        estimators=[('dt', clf1), ('rf', clf2), ('knn', clf3)], voting='hard')

    eclf.fit(x, np.ravel(y))
    return eclf
```

Il parametro Voting:

- **Soft,** si prenderà **l'argmax** delle somme delle probabilità **previste** (consigliato con modelli calibrati)
- **Hard,** si farà riferimento alla **classe** maggioritaria

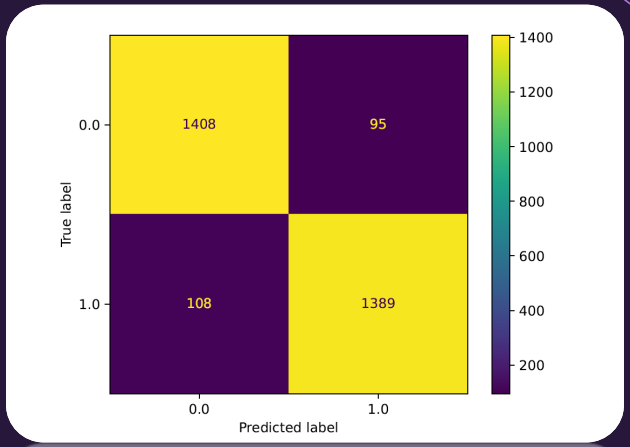


Evaluation models: Ensamble



PCA

F-Score
92%



Mutual Info

F-Score
93%

Determine Best Configuration

Per ogni modello, è stato sviluppato un metodo che permettesse di scegliere iperparametri migliori attraverso il confronto dell'FSCORE

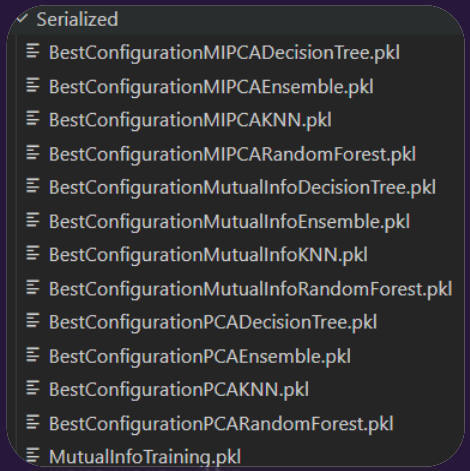
```
best_criterion = None
best_TH = None
bestN = None
best_fscore = 0

criterion = ['gini', 'entropy']

for criteria in criterion:
    for thre in np.arange(min_t, max_t, step):
        avg_fscore = 0
        fscores = []
        selectedFeatures = topFeatureSelect(rank, thre)
        if (len(selectedFeatures) > 0):
            # Utilizzo la lunghezza di ListXTrain poichè è la stessa di ListXTest
            for i in range(len(ListXTrain)):
                x_train_feature_selected = ListXTrain[i].loc[:,
                                                             selectedFeatures]
                x_test = ListXTest[i].loc[:, selectedFeatures]
                clf = decisionTreeLearner(
                    x_train_feature_selected, ListYTrain[i], criteria)

                y_pred = clf.predict(x_test)
                fscores.append(f1_score(ListYTest[i], y_pred))

            if (len(fscores) > 1):
                avg_fscore = np.mean(fscores)
                print(f"Average F1 score: '{avg_fscore}'")
                if avg_fscore == best_fscore:
                    if (len(selectedFeatures) < len(bestN)):  # Criterio aggiuntivo
                        best_fscore = avg_fscore
                        best_criterion = criteria
                        best_TH = thre
                        bestN = selectedFeatures

                if avg_fscore > best_fscore:
```

Serialization

Per migliorare l'intero **processo** è stata utilizzata la serializzazione attraverso il modulo **Pickle**, che implementa protocolli binari per la **serializzazione** e **deserializzazione** di oggetti.

```
erialize_dir = script_path.parent.parent / \
    "Serialized" / "MutualInfoTraining.pkl"

# Verifica se il file esiste
if os.path.exists(serialize_dir):
    # Se il file esiste, leggi i parametri
    with open(serialize_dir, "rb") as f:
        serialize_dir = pickle.load(f)
        rank = serialize_dir

else:
    rank = mutualInfoRank(x, y)
    MutualInfoTraining = rank
    # print(f"X mutual_info: '{rank}'\n")
    # Salva il dizionario in un file usando pickle
    with open(serialize_dir, "wb") as f:
        pickle.dump(MutualInfoTraining, f)
```

Questo ha permesso di salvare tutte quante le **configurazioni** calcolate per ogni **modello** e i valori di **mutual info** riguardo ogni variabile indipendente



Risultati finali e Workflow

Data Exploration

- Visualizzazione **box plot** e valutazione **variabili**

Data Selection

- Ricerca e rimozione di variabili con **valori con minimi e massimi uguali**
- Calcolo del **Mutual Info**

Data Trasformation

- Applicazione **PCA**

Data Mining

- Utilizzo di tutti i **modelli** analizzati con **algoritmi** specifici
- Ricerca dei **parametri** migliori attraverso la valutazione **dell'FSCORE** di diverse configurazioni

Evaluation

- Utilizzo della **K-Fold Cross Validation**
- Confronto dei **risultati** ottenuti

Random Forest (Mutual Info)

	Precision	Recall	F1-Score	Support
Goodware	0,93	0,95	0,94	1503
Malware	0,95	0,93	0,94	1497

	Accuracy
Macro AVG	0,94
Weighted AVG	0,94

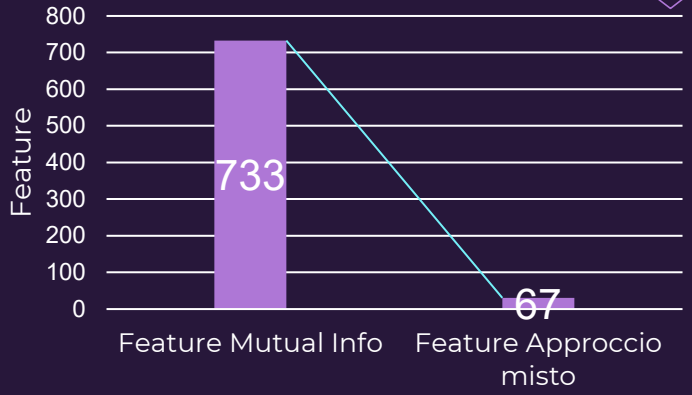
Idea: Utilizzo combinato di Mutual Info e PCA

Per ottimizzare ulteriormente il **processo**:

- È stata effettuata dapprima la feature selection con **Mutual Info** sull'intero training set (*Utilizzando il thresold precedentemente calcolato nella «best configuration» della Random Forest*)
- Successivamente, è stata applicata la PCA, ricercando il livello di thresold e i relativi parametri tramite «best configuration»

Cosa comporta la riduzione?

Feature selezionate

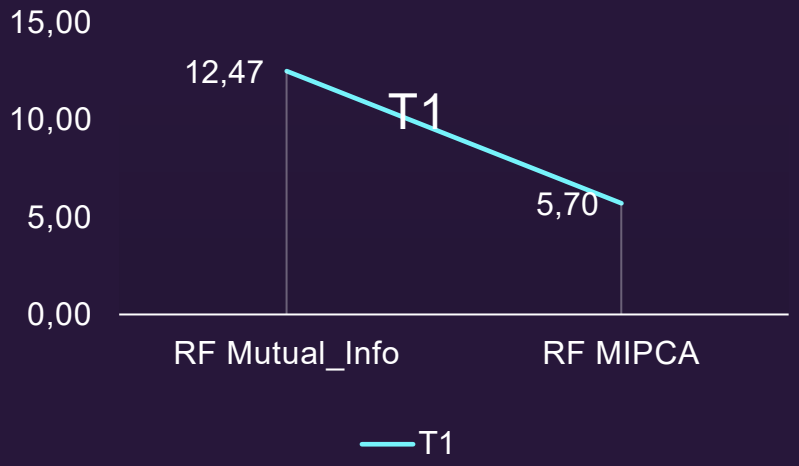


Del totale delle feature precedentemente utilizzate, è stato possibile ottenere gli stessi risultati soltanto con il 9.14% del totale (67 feature), con una riduzione del numero di feature del 90.86%

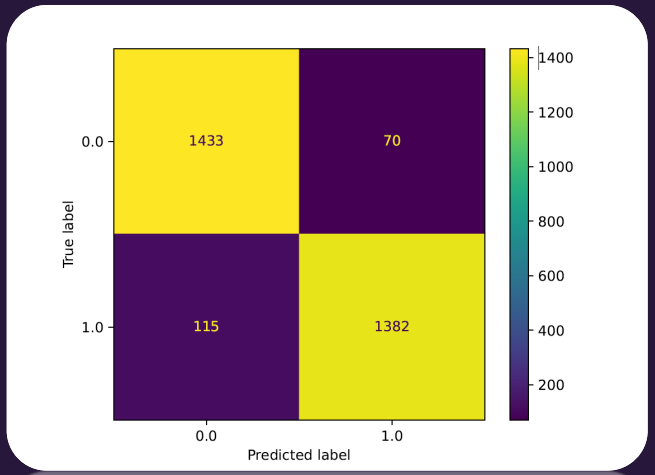
Idea: Utilizzo combinato di Mutual Info e PCA

Le perfomance sono state testate anche dal punto di vista temporale, passando dai 12,47 secondi di addestramento ai 5,70 con una riduzione circa del 55%.

Significherebbe per esempio, passare da un ipotetico ***addestramento*** *di* ***72 ore*** *ad uno di* ***39.6 ore****, mantenendo le medesime* ***prestazioni*** *e con un* ***impatto*** ***economico e infrastrutturale minore***



Idea: Utilizzo combinato di Mutual Info e PCA



Mutual Info
F-Score
94%

	Precision	Recall	F1-Score	Support
Goodware	0,93	0,95	0,94	1503
Malware	0,95	0,92	0,94	1497

	Accuracy
Macro AVG	0,94
Weighted AVG	0,94

ANALISI DEI DATI PER LA SICUREZZA
A.A 2023/2024

Cellammare Gabriel

GRAZIE PER L'ATTENZIONE!

Matricola: 807350
Email: g.cellammare1@studenti.uniba.it